NATURAL JOINs

by Sophia

WHAT'S COVERED

In this lesson, you will use a NATURAL JOIN between tables, in two parts. Specifically, this lesson will cover:

1. Getting Started

NATURAL JOINs in PostgreSQL combine rows from two or more tables based on columns with matching names and data types. NATURAL JOINs differ from other JOIN types in that PostgreSQL determines the matching columns automatically rather than you defining them explicitly. Using this method simplifies query syntax since columns do not have to be explicitly specified.

BIG IDEA

Despite their efficiency, NATURAL JOINs also pose certain risks and limitations. Changing column names or adding new columns could unintentionally affect query results owing to the JOIN's exclusive reliance on column names. A NATURAL JOIN can also produce incorrect results if two or more columns share the same name but contain different data. In order to avoid these potential pitfalls, many database professionals prefer explicitly defined JOIN conditions in order to control the JOIN process and ensure a more accurate and predictable result.

Behind the scenes, a NATURAL JOIN goes through three stages:

- The first stage creates the product of the two tables together.
- The next stage takes the output from the prior stage and only displays the rows in which the common attributes are equal to one another.
- In the last stage, a PROJECT is performed on the results to have a single copy of each attribute, which removes the duplicate column.

The user executing the JOIN operation doesn't have to issue commands for each of those stages, though; a single NATURAL JOIN clause handles it all. The final result based on the NATURAL JOIN provides a result set that only has the matches between the two tables. It does not include any unmatched pairs from either table.

The syntax of a NATURAL JOIN looks like the following:

```
SELECT <column_list>
FROM <table1>
NATURAL [INNER, LEFT, RIGHT] JOIN <table2>
```

With a NATURAL JOIN, the default type is an INNER JOIN, but it can also perform a LEFT or RIGHT JOIN. Instead of a column list, we can also use the asterisk (*). This will include all columns from both tables that have the same name.

Let's build a simple set of tables of product and categories:

```
CREATE TABLE category ( category_id serial PRIMARY KEY, category_name VARCHAR (100) NOT NULL );
CREATE TABLE product ( product_id serial PRIMARY KEY, product_name VARCHAR (100) NOT NULL, category_id INT NOT NULL, FOREIGN KEY (category_
INSERT INTO category (category_name)
VALUES ('Game'), ('Movie'), ('CD');
INSERT INTO product (product_name, category_id)
VALUES ('Call of Duty', 1), ('Final Fantasy', 1), ('Wizard of Oz', 2), ('Jaws', 2), ('Great Hits', 3), ('Journey', 3);
```

To join them using a NATURAL JOIN, we can set it as the following:

```
SELECT *
FROM category
NATURAL JOIN product;
```

Query Results

Row count: 6

category_id	category_name	product_id	product_name
1	Game	1	Call of Duty
1	Game	2	Final Fantasy
2	Movie	3	Wizard of Oz
2	Movie	4	Jaws
3	CD	5	Great Hits
3	CD	6	Journey

Note that we could also swap the two tables in the query:

```
SELECT *
FROM product
NATURAL JOIN category;
```

Query Results

Row count: 6

category_id	product_id	product_name	category_name
1	1	Call of Duty	Game
1	2	Final Fantasy	Game
2	3	Wizard of Oz	Movie
2	4	Jaws	Movie
3	5	Great Hits	CD
3	6	Journey	CD

In both cases, the result set will display the common column first, which in this example is the category_id. Then, it will display all of the columns from the first table (other than the common column) and all of the columns from the second table (other than the common column).

TERM TO KNOW

NATURAL JOINs

SQL JOINs based on columns with identical names in the related tables. They are easier to set up than other JOINs but can result in unintended connections when the tables share multiple common column names.

2. Problems With NATURAL JOINs

There are potential issues to be aware of with a NATURAL JOIN, and they can sometimes create unexpected results. Let's recreate these tables, but instead of having product_name and category_name, let's just call these columns "name" in both tables:

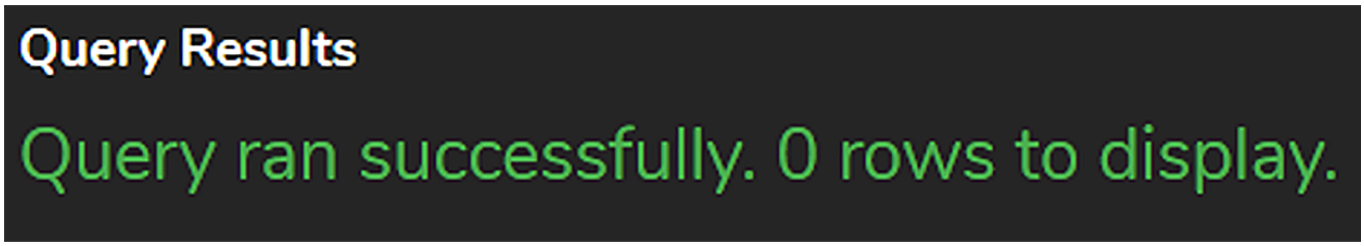
```
CREATE TABLE category ( category_id serial PRIMARY KEY, name VARCHAR (100) NOT NULL );
CREATE TABLE product ( product_id serial PRIMARY KEY, name VARCHAR (100) NOT NULL, category_id INT NOT NULL, FOREIGN KEY (category_id) REF
INSERT INTO category (name)
VALUES ('Game'), ('Movie'), ('CD');
```

```
INSERT INTO product (name, category_id)
VALUES ('Call of Duty', 1), ('Final Fantasy', 1), ('Wizard of Oz', 2), ('Jaws', 2), ('Great Hits', 3), ('Journey', 3);
```

Consider if we now tried to run the NATURAL JOIN:

```
SELECT *
FROM product
NATURAL JOIN category;
```

We get the following result:



Although we have the category_id in both tables, we also have name columns in both tables. This common column has different meanings in each table, and since the values in the two do not match, no rows are returned.

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 in SQL, a NATURAL JOIN combines rows from two or more tables based on columns with the same name or data type. When you use a NATURAL JOIN, the columns to join on are automatically determined by comparing their names, unlike other types of JOINs. With this approach, you don't have to specify the JOIN conditions explicitly, which is especially useful when dealing with tables that share attributes.

NATURAL JOINs are convenient, but they also present certain **problems**. When joining tables that contain columns with the same names but different meanings or data types, the JOIN might lead to unexpected or incorrect results. Any future changes to column names could also affect query behavior. Some database professionals prefer using explicit JOIN conditions because they give them more control and clarity over the joining process. When used carefully, NATURAL JOINs can be powerful tools, but there are limitations and dangers to be aware of.

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

NATURAL JOINs
SQL JOINs based on columns with identical names in the related tables. They are easier to set up than other JOINs but can result in unintended connections when the tables share multiple common column names.