7.3.3

Joins in Action

The first round of employee lists Bobby generated were extensive, but not specific enough for our needs. Now that we've researched the different types available to us as well as when to use them, we'll be able to help Bobby create the lists he needs to present to his supervisor.

Joins can be very confusing, so we'll be practicing using them not only to make new tables fitting the bill for Bobby's task, but also so we can get more comfortable with them and even level up our skills a bit.

We aren't limited to only two tables when using joins, we can combine three or even four tables if needed. As you can imagine, the code can get a bit messy with that many moving parts. We'll practice not only joining multiple tables, but also a way to clear up code by using aliases, where we assign a nickname to a table.

Use Inner Join for Departments and dept-manager Tables

Let's create a query that will return each department name from the Departments table as well as the employee numbers and the from- and to- dates from the dept_manager table. We'll use an inner join because we want all of the matching rows from both tables.

The code for our inner join would look like the following:

```
-- Joining departments and dept_manager tables
SELECT departments.dept_name,
     dept_manager.emp_no,
     dept_manager.from_date,
     dept_manager.to_date
FROM departments
INNER JOIN dept_manager
ON departments.dept_no = dept_manager.dept_no;
```

This code tells Postgres the following:

- The `SELECT` statement selects only the columns we want to view from each table.
- The `FROM` statement points to the first table to be joined, Departments (Table 1).
- `INNER JOIN` points to the second table to be joined, dept_manager (Table 2).
- `ON departments.dept_no = managers.dept_no;` indicates where Postgres should look for matches.

As Bobby is working through this first join, he realizes that he overlooked something important: start and end dates. Some of the folks from our original list may not even work with the company anymore. Our original retirement_info table only included individuals with certain birth- and hire-dates—how many of these people have already left the company?

Use Left Join to Capture retirement-info Table

We'll need to help Bobby recreate this list. Think about what we need to have a fully accurate retirement_info table:

- Employee number
- Employee name (first and last)
- If the person is presently employed with PH

Which tables have this information? Our current retirement_info is already filtered to list only the employees born and hired within the correct time frame. The dept_emp table has the last bit we need. We'll need to perform a join to get this information into one spot. Let's get started.

First, we start with the `SELECT` statement.

```
-- Joining retirement_info and dept_emp tables
SELECT retirement_info.emp_no,
    retirement_info.first_name,
retirement_info.last_name,
    dept_emp.to_date
```

Next, we assign the left table with `FROM`.

```
FROM retirement_info
```

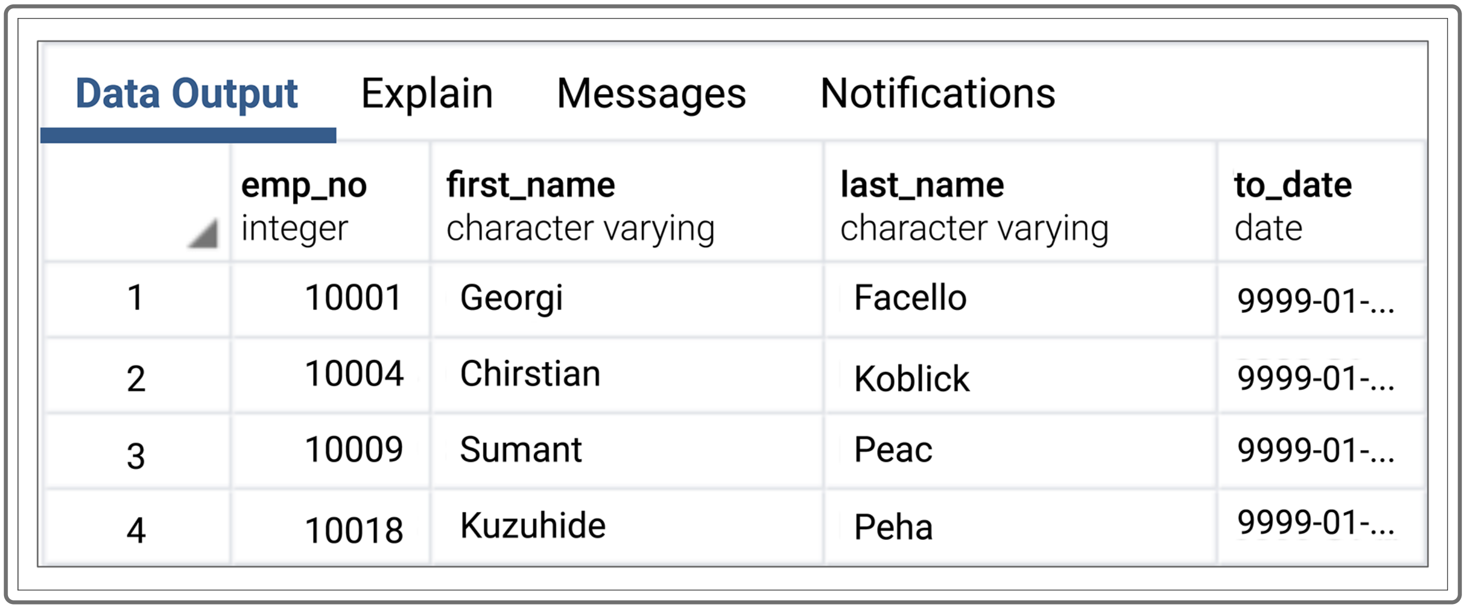
Then we specify the join we'll use. This time, use a `LEFT JOIN` to include every row of the first table (retirement_info). This also tells Postgres which table is second, or on the right side `(dept_emp)`.

```
LEFT JOIN dept_emp
```

Now we need to tell Postgres where the two tables are linked with the `ON` clause.

```
ON retirement_info.emp_no = dept_emp.emp_no;
```

Run the code to see what the data looks like. The "Data Output" tab should contain each of the four columns specified earlier: emp_no (employee number), first_name, last_name, and the to_date. Data from two different tables have been successfully merged into one!



	emp_no integer	first_name character varying	last_name character varying	to_date date
1	10001	Georgi	Facello	9999-01-...
2	10004	Chirstian	Koblick	9999-01-...
3	10009	Sumant	Peac	9999-01-...
4	10018	Kuzuhide	Peha	9999-01-...

Use Aliases for Code Readability

Joining tables can get messy. There are several different table and column name combinations to keep track of, and they can get lengthy as the query is created.

SQL has a method to shorten the code and provide greater readability by using an alias instead of a full tablename.

IMPORTANT

An alias in SQL allows developers to give nicknames to tables. This helps improve code readability by shortening longer names into one-, two-, or three-letter temporary names. This is commonly used in joins because multiple tables and columns are often listed.

Practice on the join we performed earlier:

```
-- Joining retirement_info and dept_emp tables
SELECT retirement_info.emp_no,
    retirement_info.first_name,
retirement_info.last_name,
    dept_emp.to_date
FROM retirement_info
LEFT JOIN dept_emp
ON retirement_info.emp_no = dept_emp.emp_no;
```

Each table name can be shortened to a nickname (e.g., retirement_info becomes "ri"). Let's start with updating the `SELECT` statement.

```
SELECT ri.emp_no,
    ri.first_name,
ri.last_name,
    de.to_date
```

This is already less cluttered. There are considerably fewer characters to type (and read) and less space is taken up in the editor. But how does SQL know what the nickname refers to? We still need to define the new aliases, right? That takes place in the next two lines:

```
FROM retirement_info as ri
LEFT JOIN dept_emp as de
```

And we can continue using the aliases to finish the code.

```
ON ri.emp_no = de.emp_no;
```

These aliases only exist within this query and aren't committed to that database.

Update our other join. Go back to your query editor and locate this block of code:

```
-- Joining departments and dept_manager tables
SELECT departments.dept_name,
     dept_manager.emp_no,
     dept_manager.from_date,
     dept_manager.to_date
FROM departments
INNER JOIN dept_manager
ON departments.dept_no = dept_manager.dept_no;
```

Using the same alias method and syntax as before, rename departments to "d" and dept_manager to "dm."

Starting with the `SELECT` statement, update the table names.

```
SELECT d.dept_name,
     dm.emp_no,
     dm.from_date,
     dm.to_date
```

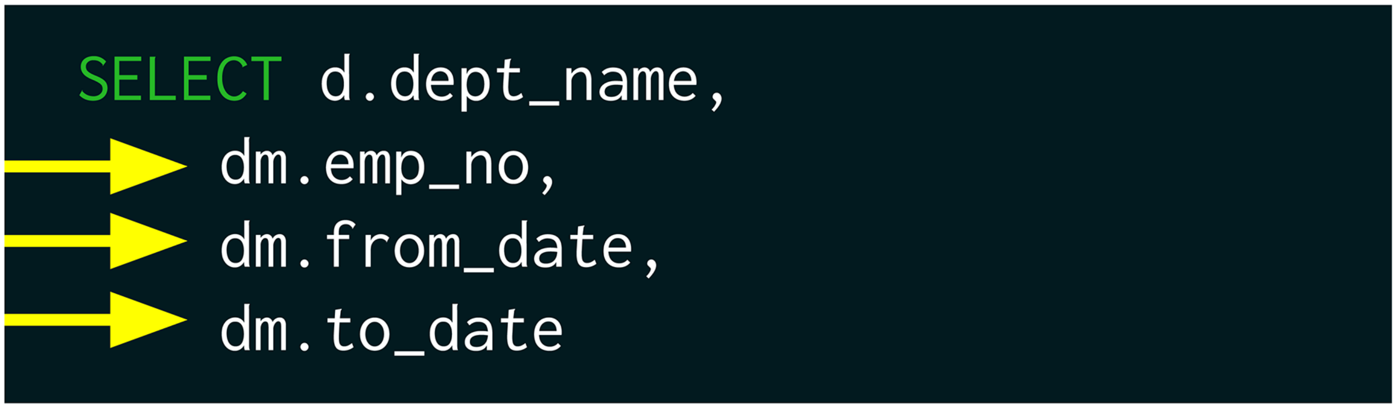
That's much cleaner already. Continue with the last few lines.

```
FROM departments as d
INNER JOIN dept_manager as dm
ON d.dept_no = dm.dept_no;
```

Did you notice a slight difference in the spacing in our new tables? Instead of each table's name and column being stretched into a single line, they are now on their own lines. The spacing is similar to when we created tables, too.

NOTE

Joins aren't always completed instantly; the performance can vary based on different factors. For example, joining more than three tables will be slower than joining two tables. The size of the datasets being joined is another factor to consider. Even the physical design of the table can play a part!



```
SELECT d.dept_name,
       dm.emp_no,
       dm.from_date,
       dm.to_date
```

This is all to help with code readability. Not only is the code neat and tidy, but it's best practice to maintain the syntax like this when there is more than one column listed in the `SELECT` statement.

Use Left Join for retirement_info and dept_emp tables

Now that we have a list of all retirement-eligible employees, it's important to make sure that they are actually still employed with PH. To do so, we're going to perform another join, this time between the retirement_info and dept_emp tables. The basic information to include in the new list is:

- Employee number
- First name
- Last name
- To-date

In the pgAdmin query editor, let's begin by specifying these columns and tables.

```
SELECT ri.emp_no,
    ri.first_name,
    ri.last_name,
de.to_date
```

Next, we need to create a new table to hold the information. Let's name it "current_emp."

```
INTO current_emp
```

The next step is to add the code that will join these two tables.

```
FROM retirement_info as ri
LEFT JOIN dept_emp as de
ON ri.emp_no = de.emp_no
```

Finally, because this is a table of current employees, we need to add a filter, using the WHERE keyword and the date 9999-01-01.

```
WHERE de.to_date = ('9999-01-01');
```

When this block of code is executed, a new table containing only the current employees who are eligible for retirement will be returned.