Use JOINs effectively

In this reading, you will review how `JOIN`s are used and will be introduced to some resources that you can use to learn more about them. A `JOIN` combines tables by using a primary or foreign key to align the information coming from both tables in the combination process. `JOIN`s use these keys to identify relationships and corresponding values across tables.

If you need a refresher on primary and foreign keys, refer to the glossary for this course, or go back to Databases in data analytics.

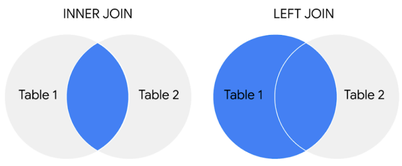
The general `JOIN` syntax

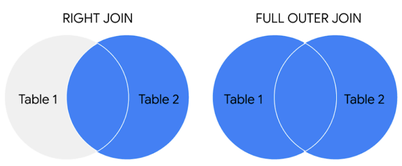
```
SELECT
   -- table columns from tables are inserted here
   table_name1.column_name
   table_name2.column_name
FROM
   table_name1
JOIN
   table_name2
ON table_name1.column_name = table_name2.column_name
```

As you can see from the syntax, the `JOIN` statement is part of the FROM clause of the query. `JOIN` in SQL indicates that you are going to combine data from two tables. `ON` in SQL identifies how the tables are to be matched for the correct information to be combined from both.

Type of `JOIN`s

There are four general ways in which to conduct `JOIN`s in SQL queries: `INNER`, `LEFT`, `RIGHT`, and `FULL OUTER`.





Here is what these different `JOIN` queries do.

`INNER JOIN`

INNER is *optional* in this SQL query because it is the default as well as the most commonly used `JOIN` operation. You may see this as `JOIN` only. `INNER JOIN` returns records if the data lives in both tables. For example, if you use `INNER JOIN` for the `customers` and `orders` tables and match the data using the `customer_id` key, you would combine the data for each `customer_id` that exists in both tables. If a `customer_id` exists in the `customers` table but not the `orders` table, data for that `customer_id` isn't joined or returned by the query.

```
SELECT
   customers.customer_name,
   orders.product_id,
   orders.ship_date
FROM
   customers
INNER JOIN
   orders
ON customers.customer_id = orders.customer_id
```

The results from the query might look like the following, where customer_name is from the customers table and product_id and ship_date are from the orders table:

customer_name	product_id	ship_date
Martin's Ice Cream	043998	2021-02-23
Beachside Treats	872012	2021-02-25
Mona's Natural Flavors	724956	2021-02-28
... etc.	... etc.	... etc.

The data from both tables was joined together by matching the `customer_id` common to both tables. Notice that `customer_id` doesn't show up in the query results. It is simply used to establish the relationship between the data in the two tables so the data can be joined and returned.

`LEFT JOIN`

You may see this as `LEFT OUTER JOIN`, but most users prefer `LEFT JOIN`. Both are correct syntax. `LEFT JOIN` returns all the records from the left table and only the matching records from the right table. Use `LEFT JOIN` whenever you need the data from the entire first table and values from the second table, if they exist. For example, in the query below, `LEFT JOIN` will return `customer_name` with the corresponding `sales_rep`, if it is available. If there is a customer who did not interact with a sales representative, that customer would still show up in the query results but with a `NULL` value for `sales_rep`.

```
SELECT
   customers.customer_name,
   sales.sales_rep
FROM
   customers
LEFT JOIN
   sales
ON customers.customer_id = sales.customer_id
```

The results from the query might look like the following where `customer_name` is from the `customers` table and `sales_rep` is from the `sales` table. Again, the data from both tables was joined together by matching the `customer_id` common to both tables even though `customer_id` wasn't returned in the query results.

customer_name	sales_rep
Martin's Ice Cream	Luis Reyes
Beachside Treats	NULL
Mona's Natural Flavors	Geri Hall
...etc.	...etc.

`RIGHT JOIN`

You may see this as `RIGHT OUTER JOIN` or `RIGHT JOIN`. `RIGHT JOIN` returns all records from the right table and the corresponding records from the left table. Practically speaking, `RIGHT JOIN` is rarely used. Most people simply switch the tables and stick with `LEFT JOIN`. But using the previous example for `LEFT JOIN`, the query using `RIGHT JOIN` would look like the following:

```
SELECT
   sales.sales_rep,
   customers.customer_name
FROM
  sales
RIGHT JOIN
  customers
ON sales.customer_id = customers.customer_id
```

The query results are the same as the previous `LEFT JOIN` example.

customer_name	sales_rep
Martin's Ice Cream	Luis Reyes
Beachside Treats	NULL
Mona's Natural Flavors	Geri Hall
...etc.	...etc.

`FULL OUTER JOIN`

You may sometimes see this as `FULL JOIN`. `FULL OUTER JOIN` returns all records from the specified tables. You can combine tables this way, but remember that this can potentially be a large data pull as a result. `FULL OUTER JOIN` returns all records from *both* tables even if data isn't populated in one of the tables. For example, in the query below, you will get all customers and their products' shipping dates. Because you are using a `FULL OUTER JOIN`, you may get customers returned without corresponding shipping dates or shipping dates without corresponding customers. A `NULL` value is returned if corresponding data doesn't exist in either table.

```
SELECT
   customers.customer_name,
   orders.ship_date
FROM
   customers
FULL OUTER JOIN
   orders
ON customers.customer_id = orders.customer_id
```

The results from the query might look like the following.

customer_name	ship_date
Martin's Ice Cream	2021-02-23
Beachside Treats	2021-02-25
NULL	2021-02-25
The Daily Scoop	NULL
Mountain Ice Cream	NULL
Mona's Natural Flavors	2021-02-28
...etc.	...etc.

For more information

`JOIN`s are going to be useful for working with relational databases and SQL—and you will have plenty of opportunities to practice them on your own. Here are a few other resources that can give you more information about `JOIN`s and how to use them:

- **SQL JOINs:** This is a good basic explanation of `JOIN`s with examples. If you need a quick reminder of what the different `JOIN`s do, this is a great resource to bookmark and come back to later.
- **Database JOINs - Introduction to JOIN Types and Concepts:** This is a really thorough introduction to `JOIN`s. Not only does this article explain what `JOIN`s are and how to use them, but it also explains the various scenarios in more detail of when and why you would use the different `JOIN`s. This is a great resource if you are interested in learning more about the logic behind `JOIN`ing.
- **SQL JOIN Types Explained in Visuals:** This resource has a visual representation of the different `JOIN`s. This is a really useful way to think about `JOIN`s if you are a visual learner, and it can be a really useful way to remember the different `JOIN`s.
- **SQL JOINs: Bringing Data Together One Join at a Time:** Not only does this resource have a detailed explanation of `JOIN`s with examples, but it also provides example data that you can use to follow along with their step-by-step guide. This is a useful way to practice `JOIN`s with some real data.
- **SQL JOIN:** This is another resource that provides a clear explanation of `JOIN`s and uses examples to demonstrate how they work. The examples also combine `JOIN`s with aliasing. This is a great opportunity to