Hands-On Activity: Queries for JOINS

coursera.org/learn/analyze-data/quiz/vnDR2/hands-on-activity-queries-for-joins/view-attempt

Congratulations! You passed!

Grade received 100%

To pass 100% or higher

Activity Overview

You've just learned about *JOIN*s and aliasing. Now, you'll practice writing queries that join multiple tables to build more robust datasets and use aliasing to make your SQL queries clearer. In this activity, you will load and examine two tables from the World Bank's International Education dataset in order to identify and understand their keys. You will also review *JOIN*s and write your own query that includes *JOIN*s and aliases. Finally, you'll use a *JOIN* to answer a specific question about the data.

By learning how to apply *JOIN* statements and aliasing, you'll be able to fully harness the power of relational databases by combining data from tables linked by keys.

Follow the instructions to complete each step of the activity. Then answer the questions at the end of the activity before going to the next course item.

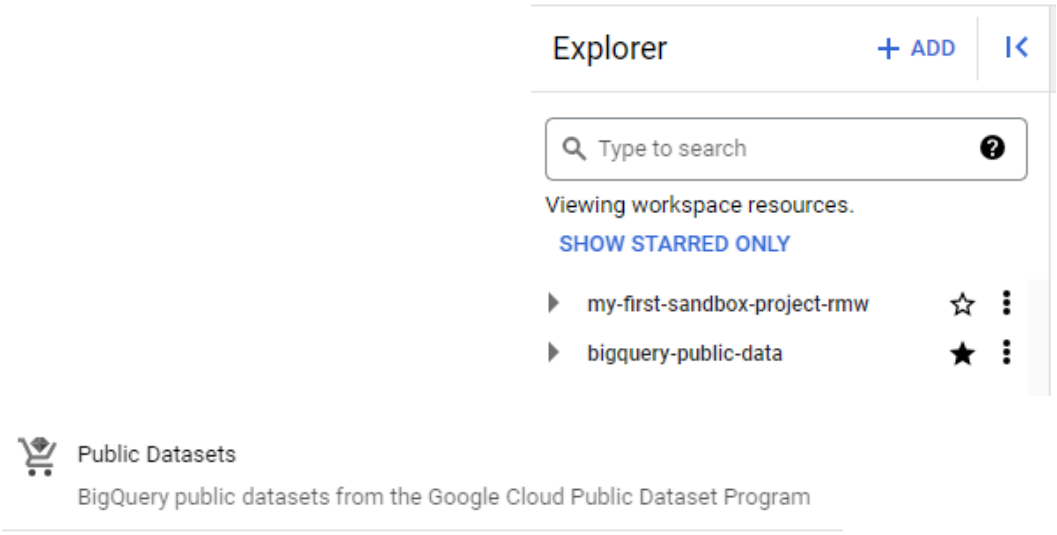
---

1. Log in to BigQuery Sandbox. If you have a free trial version of BigQuery, you can use that instead. On the BigQuery page, select the **Go to BigQuery** button.

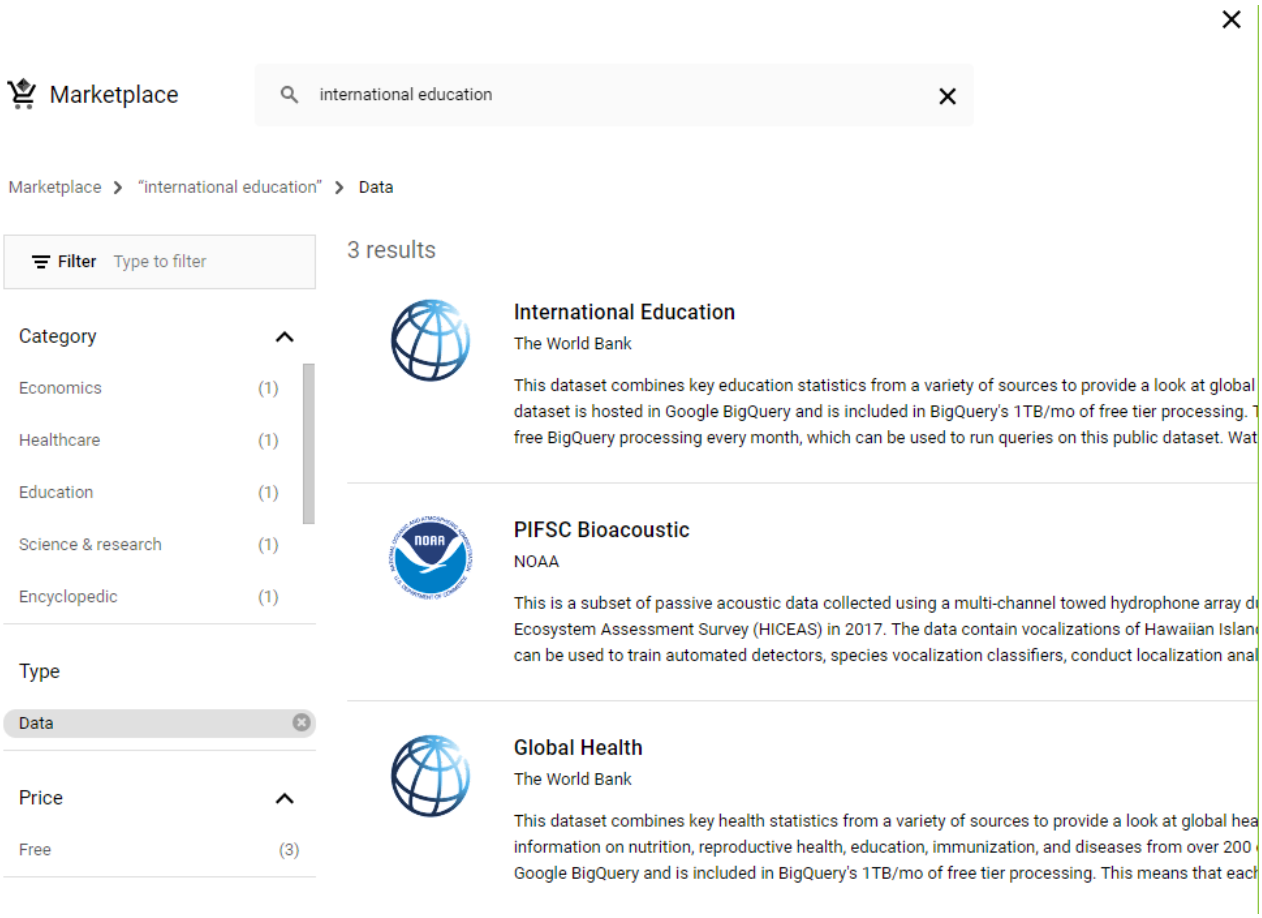
Note: BigQuery Sandbox frequently updates its user interface. The latest changes may not be reflected in the screenshots presented in this activity, but the principles remain the same. Adapting to changes in software updates is an essential skill for data analysts, and it's helpful for you to practice troubleshooting. You can also reach out to your community of learners on the discussion forum for help.

2. Now, you'll find the **Editor** interface. There are three major menus: the BigQuery navigation menu, the **Explorer** pane where you can search for datasets, and the details pane.

3. Select the **+ ADD** button in the Explorer pane, then scroll down and select the **Public Datasets** option.



4. In the **Search Marketplace** text box, enter **international education** and press Return to find the search results.

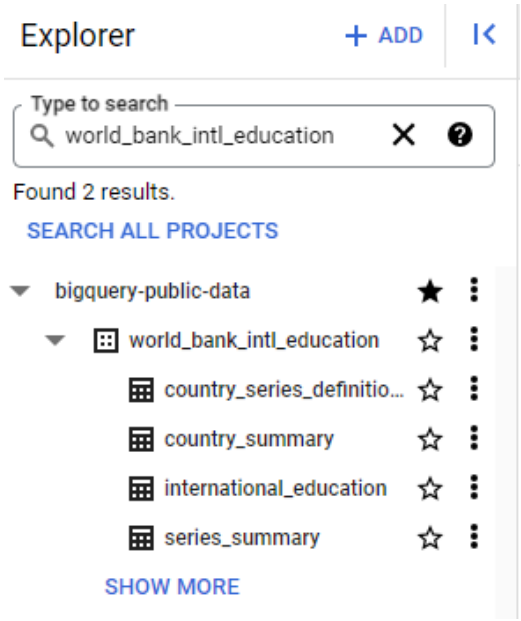


5. Select the World Bank's **International Education** dataset, which is the first result.

6. Select the **View Dataset** button to open the dataset in BigQuery in a new tab.

Note: You may want to star the **bigquery-public-data** resource in the **Explorer** pane. That way, you can access and browse the public datasets and tables more easily without having to navigate to the marketplace in the future.

7. In the Explorer pane, search for **world_bank_intl_education**. Expand the dataset to explore the tables it contains. You may also need to click on the **SHOW MORE** button for the additional tables to display.



Before you begin joining tables together, take a moment to consider how JOINs work: Two tables must be connected by their primary and foreign keys in order to join them. Keys are the most important elements of *JOIN*s—*JOIN*s function by combining tables based on those shared fields.

When designing a *JOIN* statement, these keys are listed in the *ON* statements as references to specific columns or fields within each table from the join: primary and foreign keys.

Consider two tables in the world_bank_intl_education dataset: **international_education** and **country_summary**. In order to understand how you might join these tables, take a moment to identify which columns you could use to combine them from each table. You can do that by examining the table schemas.

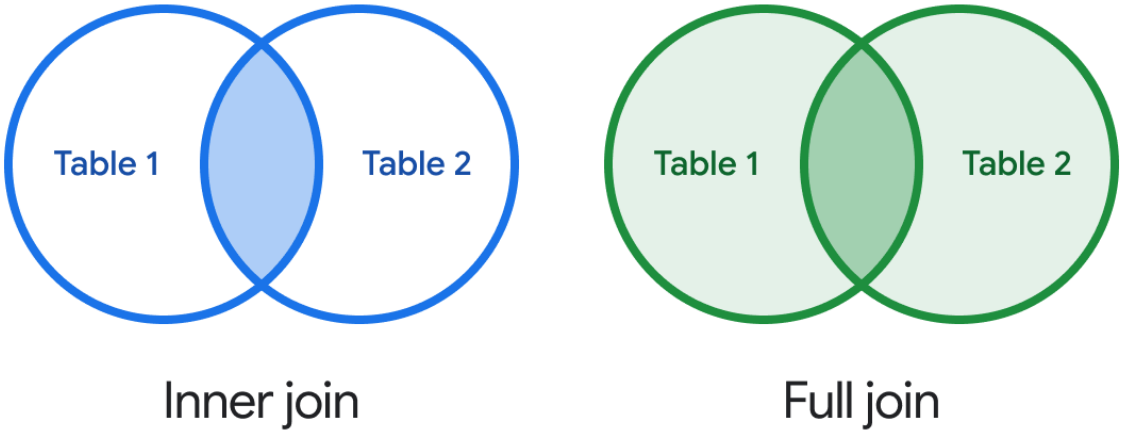
1. In the **Explorer** pane, select the **international_education** table. This will bring up the table's schema in the details pane. If the schema doesn't appear, select the **Schema** tab in the details pane. You might also check out the preview to view the data in the table.

2. Next, select the **country_summary** table and examine its schema. You'll find that the country_code column appears in both table schemas. You might also check out the preview.

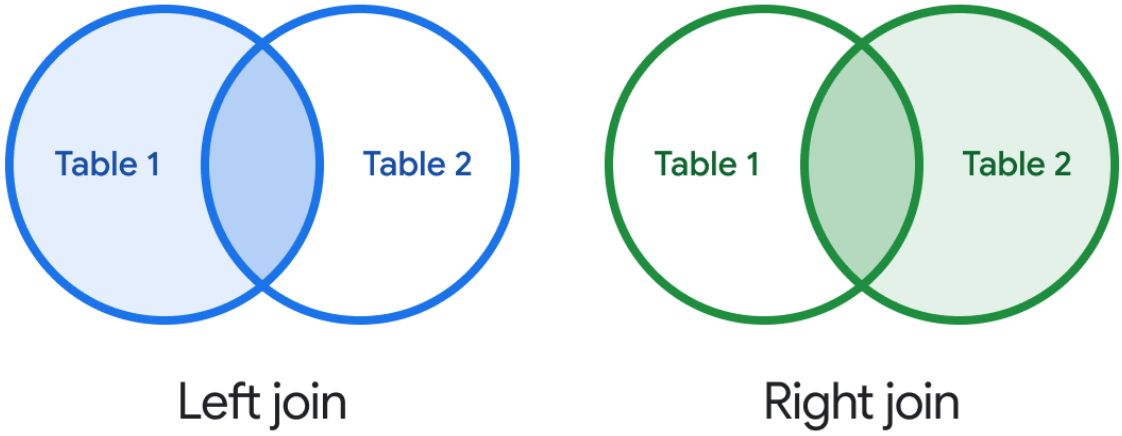
What do you notice about the columns from each table?

Both of these tables share a field name: *country_code*. As you continue with this activity, you will use this common field for your JOIN as both your primary and foreign key. It's important to understand that foreign keys don't always have the same names across tables. If you're ever unsure if the columns are the same, you can always double-check. To do so, select the **Details** tab for each table and confirm that they contain the same kinds of information.

Now that you have identified the key field for joining these tables, take a moment to review the various kinds of *JOIN* statements.



Circles represent table 1 and table 2. In the inner join, where the tables are joined is highlighted in blue, which means the results will display only the rows from the two tables where the primary key in table 1 matches the foreign key in table 2. In the full join, tables 1 and 2 are completely green. This means the query results will display all rows from both tables regardless of whether their primary key and foreign key match.



Circles represent table 1 and table 2. In the left join, table 1 is highlighted in blue and the overlap indicates that the join will show all rows in table 1 and only the related rows from table 2. In the right join, table 2 is highlighted green. The overlap shows that the results will contain all rows from table 2 and only the related rows from table 1.

The two most common kinds of JOIN statements are *INNER JOIN* and *OUTER LEFT JOIN* (also known simply as *LEFT JOIN*). As a review:

- *INNER JOIN*: Returns only the rows where the target appears in both tables.
- *LEFT JOIN*: Returns every row from the left table, as well as any rows from the right table with matching keys found in the left table.

Before you work with the BigQuery sample dataset, here's an example to help solidify your understanding of how to select the appropriate **type** of *JOIN*.

Imagine that you have one table with a list of customers (*Table 1 = CUSTOMER*) and another with a list of sales (*Table 2 = SALES*). You want to return all of the *SALES* fields, plus the customer name (*field_name = CUST_NAME*) and the state (*field_name = STATE*) that the customer lives in. The query

final query should appear like this:

```
1
2
3
4
5
6
SELECT SALES.*
  , CUSTOMER.CUST_NAME
  , CUSTOMER.STATE
FROM SALES
LEFT JOIN CUSTOMER
  ON SALES.CUST_ID = CUSTOMER.CUST_ID
```

In this case, *SALES.CUST_ID* is the primary key and *CUSTOMER.CUST_ID* is the foreign key. This will show you all sales, regardless of whether there is an associated customer ID (*CUST_ID*).

Now, consider the following: If you only want to query SALES where there is a corresponding customer ID in the CUSTOMER table, how might you change the previous query?

One way to accomplish this is to simply change the **type** of *JOIN*. *LEFT JOIN CUSTOMER to INNER JOIN CUSTOMER*.

With that simple change, all of the rows from the SALES table that were missing a corresponding customer ID from the table would NOT show.

The type of *JOIN* you use is important. You can inadvertently add or lose records with a simple incorrect selection of your *JOIN* type. When writing a query, it can help to draw out a Venn diagram like the example graphic above to help you decide which sort of *JOIN* you need.

Now, it's time to actually query the dataset. In this section, you're going to practice using *JOIN*s and explore how aliasing can help develop complex queries.

As a starting point, this is a *JOIN* that does not use any aliasing. In the details pane, compose a new query to open a query window. In the query window, copy, paste, and run the following query:

```
1
2
3
```

```
4
5
6
SELECT `bigquery-public-data.world_bank_intl_education.international_education`.country_name,
    `bigquery-public-data.world_bank_intl_education.country_summary`.country_code,
    `bigquery-public-data.world_bank_intl_education.international_education`.value
FROM `bigquery-public-data.world_bank_intl_education.international_education`
INNER JOIN `bigquery-public-data.world_bank_intl_education.country_summary`
ON `bigquery-public-data.world_bank_intl_education.country_summary`.country_code = `bigquery-public-data.world_bank_intl_education.international_education`.country_code
```

This basic query joins the *'country_summary'* and *'international_education'* tables on the *'country_code'* key and returns the country name, country code, and value column. It's a lot of effort to write out the full schema, table, field names every time you want to reference them. You can make these much easier to work with by using aliasing.

Here is the exact same query, but this time it uses an alias for each table:

```
1
2
3
4
5
6
7
8
9
SELECT
    edu.country_name,
    summary.country_code,
    edu.value
FROM
    `bigquery-public-data.world_bank_intl_education.international_education` AS edu
INNER JOIN
```

```
    `bigquery-public-data.world_bank_intl_education.country_summary` AS summary
ON edu.country_code = summary.country_code
```

This query is much easier to read and understand. You can set aliases for tables and fields by specifying the alias for the table after the table's name in *FROM* or *JOIN* statements. In this case, the international_education table was renamed as **edu**, and the country_summary table as **summary**. Using descriptive aliases is a best practice and will help you keep your queries clean, readable, and easy to work with.

Now that you've confirmed that the *JOIN* statement works, answer the following question: In 2015, how many people were of the official age for secondary education broken down by region of the world?

For this query, you will need to perform some additional tasks before returning the result:

- Exclude rows with a missing region.
- Use the *SUM(value)* to calculate the total population for a given grain size.
- Sort by highest population region first.

In a query window, copy, paste, and run the following query:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
SELECT
```

```
summary.region,
SUM(edu.value) secondary_edu_population
FROM
    `bigquery-public-data.world_bank_intl_education.international_education` AS edu
INNER JOIN
    `bigquery-public-data.world_bank_intl_education.country_summary` AS summary
ON edu.country_code = summary.country_code --country_code is our key
    WHERE summary.region IS NOT NULL
    AND edu.indicator_name = 'Population of the official age for secondary education, both sexes (number)'
    AND edu.year = 2015
GROUP BY summary.region
ORDER BY secondary_edu_population DESC
```

Query results

JOB INFORMATION | RESULTS | CHART | PREVIEW

Row	region	secondary_edu_popu
1	South Asia	237541684.0
2	East Asia & Pacific	172016129.0
3	Sub-Saharan Africa	135639085.0
4	Europe & Central Asia	70181959.0
5	Latin America & Caribbean	67937467.0
6	Middle East & North Africa	44318682.0
7	North America	27003321.0

Notice how, in this query, an alias is also set to give the *SUM(edu.value)* a more descriptive name for the temporary table the query returns.

Also note that the *WHERE* statement excludes rows with any null information. This is necessary to present the data succinctly and display only seven rows for the seven regions present in the data. However, this *WHERE* statement means that the results will return the same regardless of which *JOIN* you use. In the next section, you'll explore a situation where you need to use a specific kind of join in your query.

In the last query, you used an *INNER JOIN* to find the total population of the official age for secondary education in 2015 broken down by region. Because of the *WHERE* statement that removed null values in our previous query, using any other *JOIN* methods (left outer join, right outer join, full outer) would have produced the same result.

Now, you will write a *LEFT JOIN*, a type of *OUTER JOIN*, for a situation where the type of query you use will change the result you return.

Consider this scenario: You have been tasked to provide data for a feature sports article on NCAA basketball in the 1990s. The writer wants to include a funny twist about which Division 1 team mascots were the winningest.

You'll need a list of all NCAA Division I colleges and universities; their mascots, if applicable; and their number of wins and losses. You can find this information by typing **ncaa_basketball** in the Explorer tab dataset search bar on BigQuery.

Next, start a new query tab by clicking on the **blue +** button. Your query should join the season statistics from one table with the mascot information from another. You need to use a *LEFT JOIN* instead of an *INNER JOIN* because not all teams have mascots. If you use an *INNER JOIN*, you would exclude teams with no mascot.

To demonstrate this, copy, paste, and run the following query:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

```
SELECT
 seasons.market AS university,
 seasons.name AS team_name,
 mascots.mascot AS team_mascot,
 AVG(seasons.wins) AS avg_wins,
 AVG(seasons.losses) AS avg_losses,
 AVG(seasons.ties) AS avg_ties
FROM `bigquery-public-data.ncaa_basketball.mbb_historical_teams_seasons` AS seasons
LEFT JOIN `bigquery-public-data.ncaa_basketball.mascots` AS mascots
ON seasons.team_id = mascots.id
WHERE seasons.season BETWEEN 1990 AND 1999
 AND seasons.division = 1
 GROUP BY 1,2,3
ORDER BY avg_wins DESC, university
```

This is an example of when a *LEFT JOIN* is more helpful than an *INNER JOIN*. With this query, you can review college basketball statistics to get a better sense of which teams (and mascots) were winning most in the 1990s.

1.

Question 1

Reflection

In the last query, you use a *LEFT JOIN* instead of an *INNER JOIN* to find the information you needed. Beneath the query results, you'll find that the number of rows in your joined table is 320. If you rerun the query with an *INNER JOIN* instead of a *LEFT JOIN*, how many rows does it return?

Status: [object Object]

1 / 1 point

Correct

The query would return 317 rows. The row count drops from 320 with the *LEFT JOIN* to 317 with the *INNER JOIN*, because there are three rows where the team mascot is null. The *INNER JOIN* drops those rows which is why there is a difference between the two joins in this application.

2.

Question 2
In this activity, you used *JOIN* statements to combine data from multiple tables. In the text box below, write 2-3 sentences (40-60 words) in response to each of the following questions.

- Why do you think *JOIN* statements are important for working with databases?
- What are the consequences of choosing the wrong type of *JOIN* for your application?
- For what types of activities will aliasing be most useful?

Status: [object Object]

1 / 1 point
JOIN statements combine data from multiple tables, creating comprehensive views. Incorrect JOINs lead to inaccurate results. Aliasing simplifies complex queries and improves readability.

Correct

Congratulations on completing this hands-on activity! A thoughtful response would include that *JOIN* allows you to combine data from linked tables, helping you make comparisons and answer business questions. You might also mention that aliasing helps organize your SQL statements into a structure that is easier to read and follow.

Mastering *JOIN* statements is one of the most important parts of SQL, as combining data from multiple database tables is a core skill for data analysts.