Hands-On Activity: COUNT and COUNT DISTINCT

coursera.org/learn/analyze-data/quiz/IDPTu/hands-on-activity-count-and-count-distinct/attempt

Congratulations! You passed!

Grade received 100%

To pass 100% or higher

Activity Overview

You have learned that spreadsheets and SQL have a lot in common. In a spreadsheet, the *COUNT* function is used to count the number of cells that contain numerical values in a specified range or in an array of cells. In SQL, *COUNT* and *COUNT DISTINCT* are similar tools. The *COUNT* function returns the number of records that are returned by a query. *COUNT DISTINCT* performs the same function as *COUNT*, but it also removes both duplicate rows of the same data and null values from the result set. In this activity, you will practice using both *COUNT* and *COUNT DISTINCT* in your queries.

By the time you complete this activity, you will be able to use *COUNT* and *COUNT DISTINCT* in your queries to determine the amounts of things. Remember, *COUNT* and *COUNT DISTINCT* will return numerical values found within a dataset, helping you answer questions like, “How many customers did this?” Or, “How many transactions were there this month?” Or, “How many dates are in this dataset?”

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.

For this activity, you will need the warehouse dataset uploaded into your BigQuery project space. If you haven’t already uploaded this data, follow the instructions in the Upload the warehouse dataset to BigQuery reading.

In this scenario, you are a junior data analyst working for a company that manufactures socks. You have access to data on the company’s customers, orders, warehouses, and products. Within the dataset, there are two tables: *warehouse* and *orders*. Begin by examining the *warehouse* table:

1. In line 1, enter *SELECT*, then press **Enter/Return** on your keyboard.
2. In line 2, press **Tab** on your keyboard. Then, add an asterisk (*), then press **Enter/Return**.

3. In line 3, press **Backspace** to remove the indentation, then enter *FROM*. Then, press **Enter/Return**.
4. In line 4, press **Tab**, then enter `*your-project-name.warehouse_orders.warehouse*`. (Replace *your-project-name* with the unique name of your project).
5. Select **RUN**.

You can also copy and paste the following query into the query window instead, making sure to replace *your-project-name* with your unique project name.

```
1
2
3
4
SELECT
  *
FROM
  `your-project-name.warehouse_orders.warehouse`
```

After running the query, the five columns from the warehouse table will load in the **Query results** window:

- **warehouse_id:** indicates the ID number of each warehouse
- **warehouse_alias:** indicates the alias, or name, of each warehouse
- **maximum_capacity:** indicates the maximum capacity at each warehouse
- **employee_total:** indicates the total number of employees at each warehouse
- **state:** indicates the postal abbreviation for the U.S. state each warehouse is located in

---

Next, create a new query to retrieve the first 100 rows of the orders table. Use LIMIT to limit the number of rows returned. This is useful if you're working with large datasets, especially if you just want to explore a small sample of that dataset.

1. Start a new query.
2. In line 1, enter *SELECT*, then press **Enter/Return**.
3. In line 2, press Tab on your keyboard. Then, add an asterisk (*), then press **Enter/Return**.
4. In line 3, press **Backspace** to remove the indentation, then enter *FROM*. Then, press **Enter/Return**.

5. In line 4, press **Tab**, then enter *`your-project-name.warehouse_orders.orders`*. (Replace *your-project-name* with the unique name of your project). Then press **Enter/Return**.
6. In line 5, press **Backspace** to remove the indentation, then enter *LIMIT 100*.
7. Select **RUN**.

You can also copy and paste the following query into the query window instead, making sure to replace *your-project-name* with your unique project name.

```
1
2
3
4
5
SELECT
  *
FROM
  `your-project-name.warehouse_orders.orders`
LIMIT 100
```

After running the query, the five columns from the orders table will load in the **Query results** window:

- **order_id:** indicates the ID number of each order
- **customer_id:** indicates the ID number of each customer
- **warehouse_id:** indicates the ID number of each warehouse
- **order_date:** indicates the date on which the order was placed
- **shipper_date:** indicates the date on which the order was shipped

Also notice the number of results returned in the query: 100 results.

Aliasing involves temporarily naming a table or column in your query to make it easier to read and write. Because these names are temporary, they only last for the given query. Use a *FROM* statement to write in what the tables' aliases are going to be. This will save time in other parts of the query. To alias the *warehouse_orders.orders* table as *orders*:

1. In line 1, enter *SELECT*, then press **Enter/Return**.
2. In line 2, press **Tab** on your keyboard. Then add an asterisk (*), then press **Enter/Return**.

3. In line 3, press **Backspace** to remove the indentation, then enter **FROM**. Then press **Enter/Return**.

4. In line 4, press **Tab**, then enter *`your-project-name.warehouse_orders.orders` orders*. (Replace *your-project-name* with the unique name of your project). Then press **Enter/Return**.

Note: An alternate syntax uses the *AS* keyword to assign an alias name:

4

1

2

3

```
    `your-project-name.warehouse_orders.orders` AS orders
SELECT
    *
FROM
```

Queries can run with or without the *AS* keyword, but using *AS* enables an alias to stand out so the query is easier to read.

Perhaps you need both the warehouse details and the order details so you can report on the distribution of orders by state. Use *JOIN* to join the two tables together to get data from both of them and alias the warehouse table in the process. In this case, use *JOIN* as shorthand for *INNER JOIN* to get corresponding data from both tables.

6. In line 5, press **Backspace**, then enter *JOIN*, then press **Enter/Return**.

7. In line 6, press **Tab**, then enter *warehouse_orders.warehouse warehouse ON orders.warehouse_id = warehouse.warehouse_id*.

Your query text should read as follows:

6

3

4

5

1

2

```
    `your-project-name.warehouse_orders.warehouse` warehouse ON orders.warehouse_id = warehouse.warehouse_id
```

```
FROM
    `your-project-name.warehouse_orders.orders` AS orders
JOIN
SELECT
    *
```

With the aliases in place and the two tables joined, circle back and update the *SELECT* statement at the beginning of the query:

8. In line 1, press **Enter/Return** to create a new line after *SELECT*.

9. In line 2, press Tab, then enter *orders.*,* (that is, “orders” followed by a **period**, **asterisk**, and **comma**). Press **Enter/Return**.

10. In line 3, enter *warehouse.warehouse_alias* followed by a **comma**, then press **Enter/Return**.

11. In line 4, enter *warehouse.state*.

12. Line 5 should now contain a single **asterisk**. **Delete** this line before running the query.

13. Select **RUN**.

After running the query, the data from both tables are now joined together in the **Query results** window with these seven columns:

- **order_id:** indicates the ID number of each order.
- **customer_id:** indicates the ID number of each customer.
- **warehouse_id:** indicates the ID number of each warehouse.
- **order_date:** indicates the date on which the order was placed.
- **shipper_date:** indicates the date on which the order was shipped.
- **warehouse_alias:** indicates the names given to each warehouse as an alias.
- **state:** indicates which state the warehouse is located in.

You can also copy and paste the completed query text below into the query window. Just remember to replace *your-project-name* with your unique project name.

```
1
2
3
4
5
```

```
6
7
8
SELECT
    orders.*,
    warehouse.warehouse_alias,
    warehouse.state
FROM
    `your-project-name.warehouse_orders.orders` AS orders
JOIN
    `your-project-
name.warehouse_orders.warehouse` warehouse ON orders.warehouse_id = warehouse.warehouse_id
```

As a data analyst, you might be interested in finding the number of states with warehouses that have shipped orders.

First, you'll try to do this with the COUNT function. Begin by modifying the query you wrote in the previous step to create aliases and *JOIN* the tables.

1. Delete lines 2-4. Line 2 should be blank, in between *SELECT* in line 1 and *FROM* in line 3.
2. In line 2, press **Tab** to create an indentation. Then enter *COUNT(warehouse.state) as num_states*.
3. Select **RUN**.

Your query text should read as follows:

```
1
2
3
```

4

5

6

```
SELECT
   COUNT(warehouse.state) as num_states
FROM
   `your-project-name.warehouse_orders.orders` AS orders
JOIN
   `your-project-name.warehouse_orders.warehouse` warehouse ON orders.warehouse_id = warehouse.warehouse_id
```

Notice how the query returned more than 9,000 results. There are only 50 states, so this is clearly not the answer you're looking for! This is because the query counted every single record (row) that included a state, regardless of duplicates or null values.

Don't worry! You can modify the existing query to remove duplicates and null values, and only count the distinct states with *COUNT DISTINCT*:

1. In line 2, enter *DISTINCT* after the open parenthesis and add a space before *warehouse.state* so that line 2 reads: *COUNT (DISTINCT warehouse.state) as num_states* . This will modify *COUNT* to operate as *COUNT DISTINCT*, and it will remove all the repeated instances from the results.
2. Select **RUN**.

According to the results, there are three distinct states in the *orders* data. Nice work! You've successfully used *COUNT DISTINCT*.

Next, you might want to find the number of orders shipped from warehouses in each state, instead of the number that shipped orders. You can find this information by using *GROUP BY* to group the *state* column in the warehouse table. Use *JOIN* and *GROUP BY* in the *FROM* statement.

1. In line 7, enter *GROUP BY*, then press **Enter/Return**.
2. In line 8, press **Tab** to create an indentation, then enter *warehouse.state*.
3. Select *RUN*.

The complete query text should read as follows:

1

2

3

```
4
5
6
7
8
9
SELECT
    state,
    COUNT(DISTINCT order_id) as num_orders
FROM
    `your-project-name.warehouse_orders.orders` AS orders
JOIN
  `your-project-
name.warehouse_orders.warehouse` warehouse ON orders.warehouse_id = warehouse.warehouse_id
GROUP BY
    warehouse.state
```

After running the query, there are now three rows listed in the results table: one for each state represented within the *orders* data. The **Query results** window now displays two columns:

- Column one is **state**, which indicates which state the warehouse is located in.
- Column two is **num_orders**, which indicates the number of orders. These three numbers add up to the count that you ran previously—9,999.

Congratulations! You have successfully executed queries using *COUNT* and *COUNT DISTINCT*, as well as *SELECT*, *FROM*, and *GROUP BY* statements to create aliases and *JOIN* tables, return numerical values within a specific range, and group by specific columns within a table.

You'll find yourself using *COUNT* and *COUNT DISTINCT* during every stage of the data analysis process. Understanding what these queries are and how they are different is crucial in your role as a data analyst.

1.

Question 1

Reflection

What is the key difference between *COUNT* and *COUNT DISTINCT* in a database query?

Status: [object Object]

1 / 1 point

Correct

2.

Question 2
In the text box below, write 2-3 sentences (40-60 words) in response to each of the following questions.

- What are some of the benefits of using *COUNT* and *COUNT DISTINCT* when working with larger datasets?
- In which ways are the SQL and spreadsheet functions that you've been learning similar?

Status: [object Object]

1 / 1 point
COUNT and COUNT DISTINCT are essential for large datasets. They efficiently summarize data and optimize queries. SQL and spreadsheet functions often share similar purposes and syntax.

Correct

Great work completing this activity! In this activity, you executed queries using *COUNT* and *COUNT DISTINCT*, as well as *SELECT*, *FROM*, and *GROUP BY* statements to create aliases and *JOIN* tables, return numerical values within a specific range, and group by specific columns within a table. An effective response might include that *COUNT* and *COUNT DISTINCT* can help you answer questions about how many in a dataset or identify the number of non-null or null values in a specified range.