**Functions with multiple conditions**

In this reading, you will learn more about conditional functions and how to construct functions with multiple conditions. Recall that conditional functions and formulas perform calculations according to specific conditions. Previously, you learned how to use functions like **SUMIF** and **COUNTIF** that have one condition. You can use the **SUMIFS** and **COUNTIFS** functions if you have two or more conditions. You will learn their basic syntax in Google Sheets, and check out an example.

Refer to the resources at the end of this reading for information about similar functions in Microsoft Excel.

**SUMIF to SUMIFS**

The basic syntax of a SUMIF function is: **=SUMIF(range, criterion, sum\_range)**

The first range is where the function will search for the condition that you have set. The criterion is the condition you are applying and the sum\_range is the range of cells that will be included in the calculation.

For example, you might have a table with a list of expenses, their cost, and the date they occurred.

Column A: A1 - Expense A2 - Fuel A3 - Food A4 - Taxi A5 - Coffee A6 - Fuel A7 - Taxi A8 - Coffee A9 - Food Column B: B1 - Price B2 - $48.00 B3 - $12.34 B4 - $21.57 A5 - $2.50 A6 - $36.00 A7 - $15.88 A8 - $4.15 A9 - $6.75 Column C: C1 - Date C2 - 12/14/2020 C3 - 12/14/2020 C4 - 12/14/2020 C5 - 12/15/2020 C6 - 12/15/2020 C7 - 12/15/2020 C8 - 12/15/2020 C9 - 12/15/2020

You could use SUMIF to calculate the total price of fuel in this table, like this:

Formula bar for Cell A11. =SUMIF(A1:A9, "Fuel", B1:B9)

But, you could also build in multiple conditions by using the SUMIFS function. SUMIF and SUMIFS are very similar, but SUMIFS can include multiple conditions.

The basic syntax is: **=SUMIFS(sum\_range, criteria\_range1, criterion1, [criteria\_range2, criterion2, ...])**

The square brackets let you know that this is optional. The ellipsis at the end of the statement lets you know that you can have as many repetition of these parameters as needed. For example, if you wanted to calculate the sum of the fuel costs for one date in this table, you could create a SUMIFS statement with multiple conditions, like this:

Formula bar for Cell A12. =SUMIFS(B1:B9, A1:A9, "Fuel", C1:C9, "12/15/2020")

This formula gives you the total cost of every fuel expense from the date listed in the conditions. In this example, C1:C9 is our second criterion\_range and the date 12/15/2020 is the second condition. As long as you follow the basic syntax, you can add up to 127 conditions to a SUMIFS statement!

**COUNTIF to COUNTIFS**

Just like the SUMIFS function, COUNTIFS allows you to create a COUNTIF function with multiple conditions.

The basic syntax for COUNTIF is: **=COUNTIF(range, criterion)**

Just like SUMIF, you set the range and then the condition that needs to be met. For example, if you wanted to count the number of times Food came up in the Expenses column, you could use a COUNTIF function like this:

Formula bar for Cell A13. =COUNTIF(A1:A9, "Food")

COUNTIFS has the same basic syntax as SUMIFS: **=COUNTIFS(criteria\_range1, criterion1, [criteria\_range2, criterion2, ...])**

The criteria\_range and criterion are in the same order, and you can add more conditions to the end of the function. So, if you wanted to find the number of times Coffee appeared in the Expenses column on 12/15/2020, you could use COUNTIFS to apply those conditions, like this:

Formula bar for Cell A14. =COUNTIFS(A1:A9, "Coffee", C1:C9, "12/15/2020")

This formula follows the basic syntax to create conditions for “Coffee” and the specific date. Now we can find every instance where both of these conditions are true.

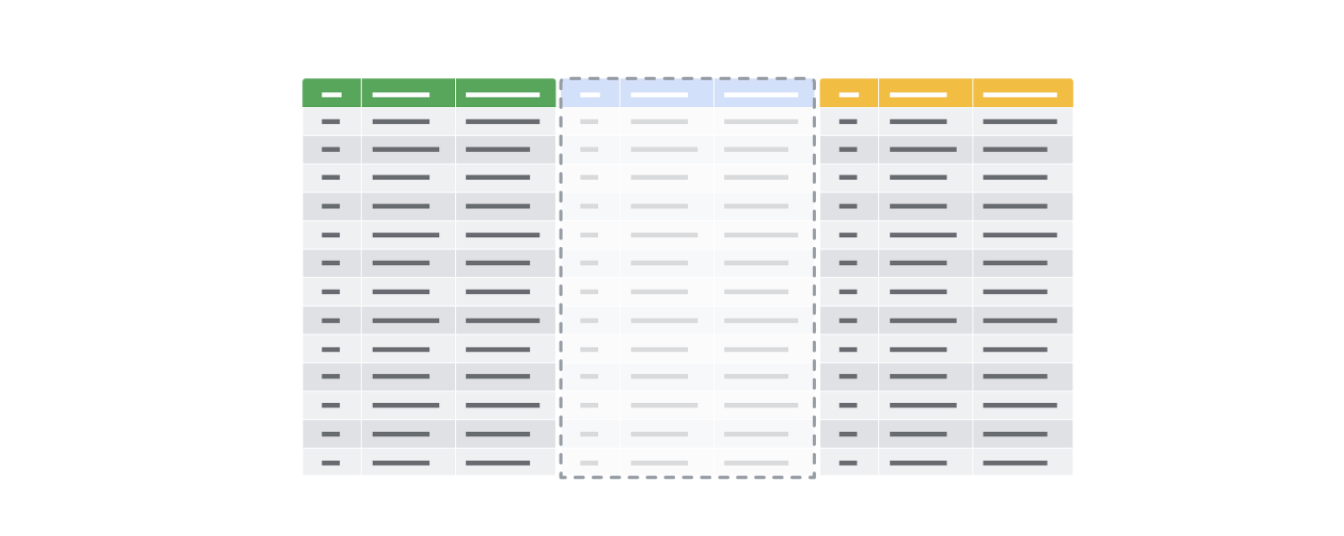
**Working with temporary tables**

**Temporary tables** are exactly what they sound like—temporary tables in a SQL database that aren’t stored permanently. In this reading, you will learn the methods to create temporary tables using SQL commands. You will also learn a few best practices to follow when working with temporary tables.

**A quick refresher on what you have already learned about temporary tables**

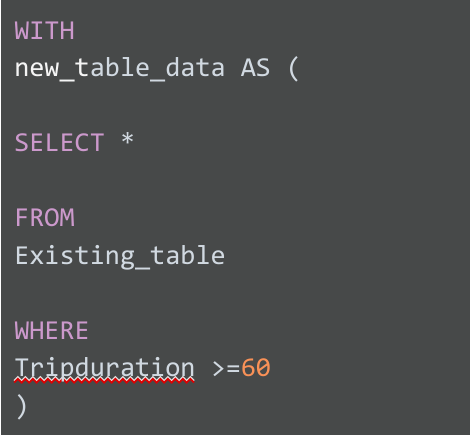
* They are automatically deleted from the database when you end your SQL session.
* They can be used as a holding area for storing values if you are making a series of calculations. This is sometimes referred to as **pre-processing** of the data.
* They can collect the results of multiple, separate queries. This is sometimes referred to as data **staging**. Staging is useful if you need to perform a query on the collected data or merge the collected data.
* They can store a filtered subset of the database. You don’t need to select and filter the data each time you work with it. In addition, using fewer SQL commands helps to keep your data clean.

It is important to point out that each database has its own unique set of commands to create and manage temporary tables. We have been working with BigQuery, so we will focus on the commands that work well in that environment. The rest of this reading will go over the ways to create temporary tables, primarily in BigQuery.



**Temporary table creation in BigQuery**

Temporary tables can be created using different clauses. In BigQuery, the **WITH** clause can be used to create a temporary table. The general syntax for this method is as follows:



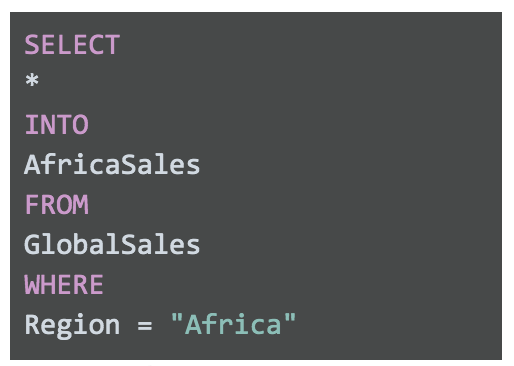
Breaking down this query a bit, notice the following:

* The statement begins with the **WITH** clause followed by the name of the new temporary table you want to create
* The **AS** clause appears after the name of the new table. This clause instructs the database to put all of the data identified in the next part of the statement into the new table.
* The opening parenthesis after the **AS** clause creates the subquery that filters the data from an existing table. The subquery is a regular **SELECT** statement along with a **WHERE** clause to specify the data to be filtered.
* The closing parenthesis ends the subquery created by the **AS** clause.

When the database executes this query, it will first complete the subquery and assign the values that result from that subquery to “new\_table\_data,” which is the temporary table. You can then run multiple queries on this filtered data without having to filter the data every time.

**Temporary table creation in other databases (not supported in BigQuery)**

The following method isn’t supported in BigQuery, but most other versions of SQL databases support it, including SQL Server and mySQL. Using **SELECT** and **INTO**, you can create a temporary table based on conditions defined by a **WHERE** clause to locate the information you need for the temporary table. The general syntax for this method is as follows:

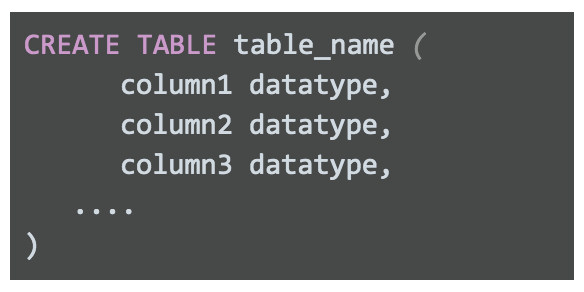
SELECT \* INTO AfricaSales FROM GlobalSales WHERE Region = "Africa"

This **SELECT** statement uses the standard clauses like **FROM** and **WHERE**, but the **INTO** clause tells the database to store the data that is being requested in a new temporary table named, in this case, “AfricaSales.”

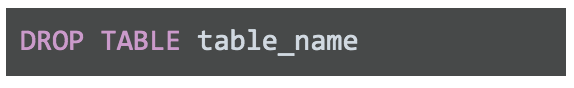
**User-managed temporary table creation**

So far, we have explored ways of creating temporary tables that the database is responsible for managing. But, you can also create temporary tables that you can manage as a user. As an analyst, you might decide to create a temporary table for your analysis that you can manage yourself. You would use the **CREATE TABLE** statement to create this kind of temporary table. After you have finished working with the table, you would then delete or drop it from the database at the end of your session.

**Note:** BigQuery uses **CREATE TEMP TABLE** instead of **CREATE TABLE**, but the general syntax is the same.

CREATE TABLE table\_name ( column1 datatype, column2 datatype, column3 datatype, .... )

After you have completed working with your temporary table, you can remove the table from the database using the **DROP TABLE** clause. The general syntax is as follows:



**Best practices when working with temporary tables**

* **Global vs. local temporary tables:** Global temporary tables are made available to all database users and are deleted when all connections that use them have closed. Local temporary tables are made available only to the user whose query or connection established the temporary table. You will most likely be working with local temporary tables. If you have created a local temporary table and are the only person using it, you can drop the temporary table after you are done using it.
* **Dropping temporary tables after use:** Dropping a temporary table is a little different from deleting a temporary table. Dropping a temporary table not only removes the information contained in the rows of the table, but removes the table variable definitions (columns) themselves. Deleting a temporary table removes the rows of the table but leaves the table definition and columns ready to be used again. Although local temporary tables are dropped after you end your SQL session, it may not happen immediately. If a lot of processing is happening in the database, dropping your temporary tables after using them is a good practice to keep the database running smoothly.