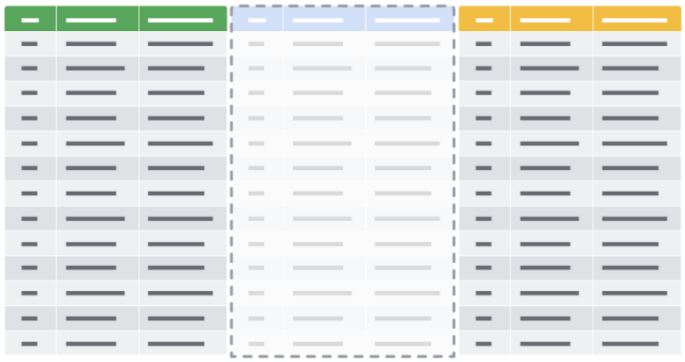
Working with temporary tables

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:

```
WITH
new _table_data AS (

SELECT *

FROM
Existing_table

WHERE
Tripduration >=60
)
```

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 open 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 close bracket 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. Some databases use **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:

```
DROP TABLE table_name
```

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.

For more information

- BigQuery Documentation for Temporary Tables**:** Documentation has the syntax to create temporary tables in BigQuery
- How to use temporary tables via WITH in Google BigQuery**:** Article describes how to use **WITH**
- Introduction to Temporary Tables in SQL Server**:** Article describes how to use **SELECT INTO** and **CREATE TABLE**
- SQL Server Temporary Tables**:** Article describes temporary table creation and removal
- Choosing Between Table Variables and Temporary Tables**:** Article describes the differences between passing variables in SQL statements vs. using temporary tables