## Course 5 - Analyze Data to Answer Questions

Organizing data makes the data easier to use in your analysis. In this part of the course, you'll learn the importance of organizing your data through sorting and filtering. You'll explore these processes in both spreadsheets and SQL as you continue to prepare your data for analysis.

**Learning Objectives**

- Describe what is involved in the data analysis process with reference to goals and key tasks
- Discuss the importance of organizing data before analysis with references to sorts and filters
- Describe sorting as it relates to data in a spreadsheet or database with reference to functionality and benefits
- Demonstrate an understanding of the steps involved in sorting and filtering data through the use of SQL queries


## Week 1 : data analysis basic

The analysis process :

● Basically, analysis is the process used to make sense of the data collected.

● The goal of analysis is to identify trends and relationships within the data so that you can accurately answer the question you're asking.

The 4 phases of analysis:

- Organize data -list the gift using sort and filter
- Format and adjust data -convert one type to another
- Get input from others - what others bought
- Transform data

Transforming data means identifying relationships and patterns between the data, and making calculations based on the data you have.

## Keeping data organized with sorting and filters

You have learned about four phases of analysis:

- Organize data
- Format and adjust data
- Get input from others
- Transform data

The organization of datasets is really important for data analysts. Most of the datasets you will use will be organized as tables. Tables are helpful because they let you manipulate your data and categorize it. Having distinct categories and classifications lets you focus on, and differentiate between, your data quickly and easily.

Data analysts also need to **format and adjust** data when performing an analysis. **Sorting** and **filtering** are two ways you can keep things organized when you format and adjust data to work with it. For example, a filter can help you find errors or outliers so you can fix or flag them before your analysis. **Outliers** are data points that are very different from similarly collected data and

might not be reliable values. The benefit of filtering the data is that after you fix errors or identify outliers, you can remove the filter and return the data to its original organization.

In this reading, you will learn the difference between sorting and filtering. You will also be introduced to how a particular form of sorting is done in a pivot table.

## Sorting versus filtering

**Sorting** is when you arrange data into a meaningful order to make it easier to understand, analyze, and visualize. It ranks your data based on a specific metric you choose. You can sort data in spreadsheets, SQL databases (when your dataset is too large for spreadsheets), and tables in documents.

For example, if you need to rank things or create chronological lists, you can sort by ascending or descending order. If you are interested in figuring out a group's favorite movies, you might sort by movie title to figure it out. Sorting will arrange the data in a meaningful way and give you immediate insights. Sorting also helps you to group similar data together by a classification. For movies, you could sort by genre -- like action, drama, sci-fi, or romance.

**Filtering** is used when you are only interested in seeing data that meets a specific criteria, and hiding the rest. Filtering is really useful when you have lots of data. You can save time by zeroing in on the data that is really important or the data that has bugs or errors. Most spreadsheets and SQL databases allow you to filter your data in a variety of ways. Filtering gives you the ability to find what you are looking for without too much effort.

For example, if you are only interested in finding out who watched movies in October, you could use a filter on the dates so only the records for movies watched in October are displayed. Then, you could check out the names of the people to figure out who watched movies in October.

To recap, the easiest way to remember the difference between sorting and filtering is that you can use sort to quickly order the data, and filter to display *only* the data that meets the criteria that you have chosen. Use filtering when you need to reduce the amount of data that is displayed.

It is important to point out that, after you filter data, you can **sort the filtered data**, too. If you revisit the example of finding out who watched movies in October, after you have filtered for the movies seen in October, you can then sort the names of the people who watched those movies in alphabetical order.

## Sorting in a pivot table

Items in the row and column areas of a pivot table are sorted in ascending order by any custom list first. For example, if your list contains days of the week, the pivot table allows weekday and month names to sort like this: Monday, Tuesday, Wednesday, etc. rather than alphabetically like this: Friday, Monday, Saturday, etc.

If the items aren't in a custom list, they will be sorted in ascending order by default. But, if you sort in descending order, you are setting up a rule that controls how the field is sorted even after new data fields are added.

**Database organization enables analysts to make decisions about which data is relevant to pull for a specific analysis. Database references let them access objects from other databases.**

# Week 2 : Getting started with data formatting

## Converting data in spreadsheets

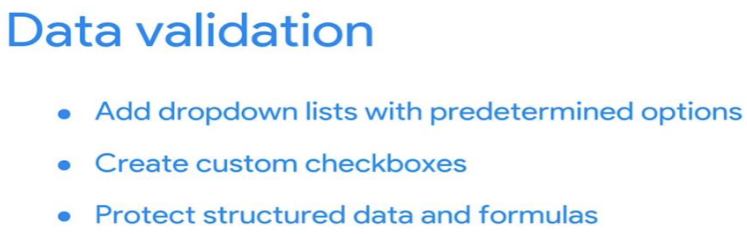
In this reading, you will learn about converting data from one format to another. One of the ways to help ensure that you have an accurate analysis of your data is by putting all of it in the correct format. This is true even if you have already cleaned and processed your data. As a part of getting your data ready for analysis, you will need to convert and format your data early on in the process.

**As a data analyst, there are lots of scenarios when you might need to convert data in a spreadsheet:**

- **String to date**
- **String to number**
- **Combining columns**
- **Number to percentage**

**Data Validation :**

Basically, it allows you to control what can and can't be entered in your worksheet.
Usually, data validation is used to add drop-down lists to cells with predetermined options for users to choose from.




Data tab => Data validation
Data validation.
Data validation can help your team track progress, protect your tables from breaking when working in big teams, and help you customize tables to your needs.

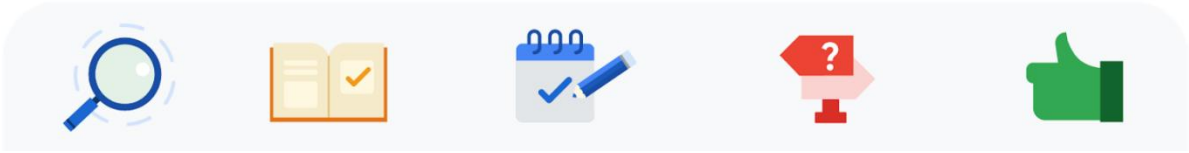
**Conditional formatting**

Format=>Conditional Formatting

Conditional formatting A spreadsheet tool that changes how cells appear when values meet specific conditions.

# Transforming data in SQL

Data analysts usually need to convert data from one format to another to complete an analysis. But what if you are using SQL rather than a spreadsheet? Just like spreadsheets, SQL uses standard rules to convert one type of data to another. If you are wondering why data transformation is an important skill to have as a data analyst, think of it like being a driver who is able to change a flat tire. Being able to convert data to the right format speeds you along in your analysis. You don't have to wait for someone else to convert the data for you.



In this reading, you will go over the conversions that can be done using the **CAST** function. There are also more specialized functions like **COERCION** to work with big numbers, and **UNIX_DATE** to work with dates. **UNIX_DATE** returns the number of days that have passed since January 1, 1970 and is used to compare and work with dates across multiple time zones. You will likely use **CAST** most often.

## Common conversions

The following table summarizes some of the more common conversions made with the **CAST** function. Refer to Conversion Rules in Standard SQL for a full list of functions and associated rules.

| Starting with | CAST function can convert to: |
|---|---|
| Numeric (number) | - Integer - Numeric (number) - Big number - Floating integer - String |
| String | - Boolean - Integer - Numeric (number) - Big number - Floating integer - Stri<br>- Date - Date time - Time - Timestamp |
| Date | - String - Date - Date time - Timestamp |

## The CAST function (syntax and examples)

**CAST** is an American National Standards Institute (ANSI) function used in lots of programming languages, including BigQuery. This section provides the BigQuery syntax and examples of converting the data types in the first column of the previous table. The syntax for the CAST function is as follows:

```
CAST(expression AS typename)
```

CAST (expression AS typename)

Where **expression** is the data to be converted and **typename** is the data type to be returned.

## Converting a number to a string

The following **CAST** statement returns a string from a numeric identified by the variable MyCount in the table called MyTable.

```
SELECT CAST(MyCount AS STRING) FROM MyTable
```

SELECT CAST (MyCount AS STRING) FROM MyTable

In the above SQL statement, the following occurs:

- **SELECT** indicates that you will be selecting data from a table
- **CAST** indicates that you will be converting the data you select to a different data type
- **AS** comes before and identifies the data type which you are casting to
- **STRING** indicates that you are converting the data to a string
- **FROM** indicates which table you are selecting the data from

## Converting a string to a number

The following **CAST** statement returns an integer from a string identified by the variable MyVarcharCol in the table called MyTable. (An integer is any whole number.)

```
SELECT CAST(MyVarcharCol AS INT) FROM MyTable
```

SELECT CAST(MyVarcharCol AS INT) FROM MyTable

In the above SQL statement, the following occurs:

- **SELECT** indicates that you will be selecting data from a table
- **CAST** indicates that you will be converting the data you select to a different data type
- **AS** comes before and identifies the data type which you are casting to
- **INT** indicates that you are converting the data to an integer
- **FROM** indicates which table you are selecting the data from

## Converting a date to a string

The following **CAST** statement returns a string from a date identified by the variable MyDate in the table called MyTable.

```
SELECT CAST(MyDate AS STRING) FROM MyTable
```

In the above SQL statement, the following occurs:

- **SELECT** indicates that you will be selecting data from a table

- **CAST** indicates that you will be converting the data you select to a different data type
- **AS** comes before and identifies the data type which you are casting to
- **STRING** indicates that you are converting the data to a string
- **FROM** indicates which table you are selecting the data from

## Converting a date to a datetime

Datetime values have the format of YYYY-MM-DD hh: mm: ss format, so date and time are retained together. The following **CAST** statement returns a datetime value from a date.

```
SELECT CAST (MyDate AS DATETIME) FROM MyTable
```

In the above SQL statement, the following occurs:

- **SELECT** indicates that you will be selecting data from a table
- **CAST** indicates that you will be converting the data you select to a different data type
- **AS** comes before and identifies the data type which you are casting to
- **DATETIME** indicates that you are converting the data to a datetime value
- **FROM** indicates which table you are selecting the data from

### The SAFE_CAST function

Using the **CAST** function in a query that fails returns an error in BigQuery. To avoid errors in the event of a failed query, use the **SAFE_CAST** function instead. The **SAFE_CAST** function returns a value of Null instead of an error when a query fails.

The syntax for **SAFE_CAST** is the same as for **CAST**. Simply substitute the function directly in your queries. The following **SAFE_CAST** statement returns a string from a date.

```
SELECT SAFE_CAST(MyDate AS STRING) FROM MyTable
```

# Manipulating strings in SQL

Knowing how to convert and manipulate your data for an accurate analysis is an important part of a data analyst's job. In this reading, you will learn about different SQL functions and their usage, especially regarding string combinations.

A **string** is a set of characters that helps to declare the texts in programming languages such as SQL. SQL string functions are used to obtain various information about the characters, or in this case, manipulate them. One such function, **CONCAT**, is commonly used. Review the table below to learn more about the CONCAT function and its variations.

| Function | Usage | Example |
|---|---|---|
| CONCAT | A function that adds strings together to create new text strings that can be used as unique keys | CONCAT ('Google', '.com'); |
| CONCAT_WS | A function that adds two or more strings together with a separator | CONCAT_WS (' . ', 'www', 'google', 'com')<br><br>*The separator (being the period) gets input before and after Google when you run the SQL function |
| CONCAT with + | Adds two or more strings together using the + operator | 'Google' + '.com' |

## Week 3 : VLOOKUP core concepts

**VLOOKUP core concepts**

Functions can be used to quickly find information and perform calculations using specific values. In this reading, you will learn about the importance of one such function, **VLOOKUP**, or Vertical Lookup, which searches for a certain value in a spreadsheet column and returns a corresponding piece of information from the row in which the searched value is found.

When do you need to use VLOOKUP?

Two common reasons to use VLOOKUP are:

- Populating data in a spreadsheet
- Merging data from one spreadsheet with data in another

**VLOOKUP syntax**

A VLOOKUP function is available in both Microsoft Excel and Google Sheets. You will be introduced to the general syntax in Google Sheets. (You can refer to the resources at the end of this reading for more information about VLOOKUP in Microsoft Excel.)

```
VLOOKUP(10003, A2:B26, 2, FALSE)
```

Here is the syntax.

```
VLOOKUP(search_key, range, index, [is_sorted])
```

**search_key**

- The value to search for.
- For example, 42, "Cats", or I24.

**range**

- The range to consider for the search.
- The first column in the range is searched to locate data matching the value specified by search_key.

**index**

- The column index of the value to be returned, where the first column in range is numbered 1.
- If index is not between 1 and the number of columns in range, #VALUE! is returned.

**is_sorted**

- Indicates whether the column to be searched (the first column of the specified range) is sorted. TRUE by default.
- It's recommended to set is_sorted to FALSE. If set to FALSE, an exact match is returned. If there are multiple matching values, the content of the cell corresponding to the first value found is returned, and #N/A is returned if no such value is found.
- If is_sorted is TRUE or omitted, the nearest match (less than or equal to the search key) is returned. If all values in the search column are greater than the search key, #N/A is returned.

What if you get #N/A?

As you have just read, #N/A indicates that a matching value can't be returned as a result of the VLOOKUP. The error doesn't mean that anything is actually wrong with the data, but people might have questions if they see the error in a report. You can use the **IFNA** function to replace the #N/A error with something more descriptive, like "Does not exist."

```
IFNA(#N/A, "Does not exist")
```

Here is the syntax.

```
IFNA(value, value_if_na)
```

**value**

- This is a required value.
- The function checks if the cell value matches the value; such as #N/A.

**value_if_na**

- This is a required value.
- The function returns this value if the cell value matches the value in the first argument; it returns this value when the cell value is #N/A.

**Helpful VLOOKUP reminders**

- TRUE means an approximate match, FALSE means an exact match on the search key. If the data used for the search key is sorted, TRUE can be used.
- You want the column that matches the search key in a VLOOKUP formula to be on the left side of the data. VLOOKUP only looks at data to the right after a match is found. In other words, the index for VLOOKUP indicates columns to the right only. This may require you to move columns around before you use VLOOKUP.
- After you have populated data with the VLOOKUP formula, you may copy and paste the data as values only to remove the formulas so you can manipulate the data again.

**Using JOINs effectively**

In this reading, you will review how JOINs 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. JOINs 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.

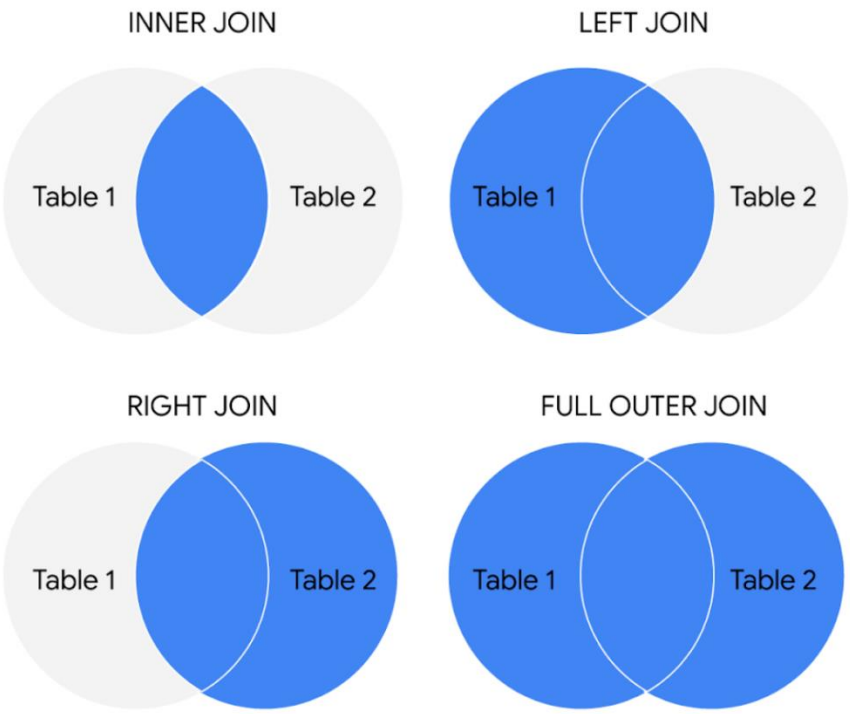
The general JOIN syntax

```
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 JOINs

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




The circles represent left and right tables, and where they are joined is highlighted in blue

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. |

# SQL functions and subqueries: A functional friendship

In this reading, you will learn about SQL functions and how they are sometimes used with subqueries. **SQL functions** are tools built into SQL to make it possible to perform calculations. A **subquery** (also called an inner or nested query) is a query within another query.

## How do SQL functions, function?

SQL functions are what help make data aggregation possible. (As a reminder, data aggregation is the process of gathering data from multiple sources in order to combine it into a single, summarized collection.) So, how do SQL functions work? Going back to W3Schools, let's review some of these functions to get a better understanding of how to run these queries:

- **SQL HAVING:** This is an overview of the HAVING clause, including what it is and a tutorial on how and when it works.
- **SQL CASE:** Explore the usage of the CASE statement and examples of how it works.
- **SQL IF:** This is a tutorial of the IF function and offers examples that you can practice with.

- **SQL COUNT:** The COUNT function is just as important as all the rest, and this tutorial offers multiple examples to review.

## Subqueries - the cherry on top

Think of a query as a cake. A cake can have multiple layers contained within it and even layers within those layers. Each of these layers are our subqueries, and when you put all of the layers together, you get a cake (query). Usually, you will find subqueries nested in the SELECT, FROM, and/or WHERE clauses. There is no general syntax for subqueries, but the syntax for a basic subquery is as follows:

```
SELECT account_table.*
  FROM (
        SELECT *
          FROM transaction.sf_model_feature_2014_01
         WHERE day_of_week = 'Friday'
       ) account_table
 WHERE account_table.availability = 'YES'
```

You will find that, within the first SELECT clause is another SELECT clause. The second SELECT clause marks the start of the subquery in this statement. There are many different ways in which you can make use of subqueries, and resources referenced will provide additional guidance as you learn. But first, let's recap the subquery rules.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses
- A subquery can have only one column specified in the SELECT clause. But if you want a subquery to compare multiple columns, those columns must be selected in the main query.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator which allows you to specify multiple values in a WHERE clause.
- A subquery can't be nested in a SET command. The SET command is used with UPDATE to specify which columns (and values) are to be updated in a table.

# 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.

| | A | B | C |
|---|---|---|---|
| 1 | Expense | Price | Date |
| 2 | Fuel | $48.00 | 12/14/2020 |
| 3 | Food | $12.34 | 12/14/2020 |
| 4 | Taxi | $21.57 | 12/14/2020 |
| 5 | Coffee | $2.50 | 12/15/2020 |
| 6 | Fuel | $36.00 | 12/15/2020 |
| 7 | Taxi | $15.88 | 12/15/2020 |
| 8 | Coffee | $4.15 | 12/15/2020 |
| 9 | Food | $6.75 | 12/15/2020 |

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

```
A11 | fx | =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:

```
A12 | fx | =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:

```
A13  fx  =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:

```
A14  fx  =COUNTIFS(A1:A9, "Coffee", C1:C9, "12/15/2020")
```

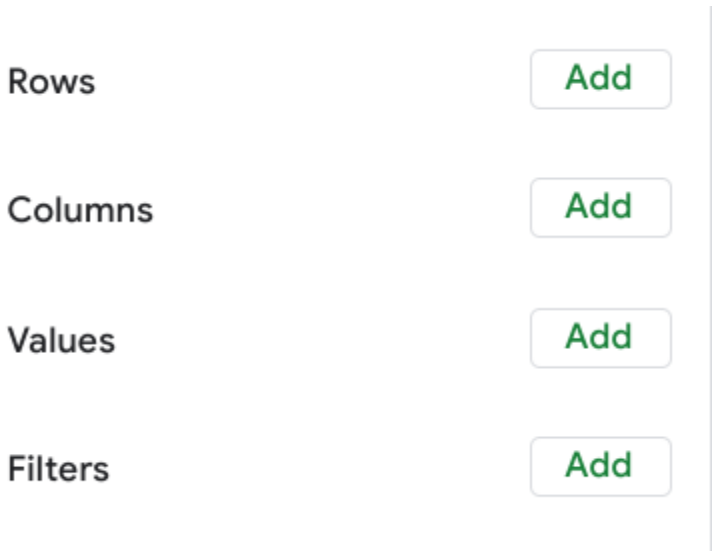
This formula follows the basic syntax to create conditions for "Coffee" and the specific date. Now we can find every instance where both of these conditions are true.

## Elements of a pivot table

Previously, you learned that a pivot table is a tool used to sort, reorganize, group, count, total, or average data in spreadsheets. In this reading, you will learn more about the parts of a pivot table and how data analysts use them to summarize data and answer questions about their data.

**Pivot tables** make it possible to view data in multiple ways in order to identify insights and trends. They can help you quickly make sense of larger data sets by comparing metrics, performing calculations, and generating reports. They're also useful for answering specific questions about your data.

A pivot table has four basic parts: rows, columns, values, and filters.



The **rows** of a pivot table organize and group data you select horizontally. For example, in the Working with pivot tables video, the Release Date values were used to create rows that grouped the data by year.

| *Release Date - Year* |
|---|
| 2012 |
| 2013 |
| 2014 |
| 2015 |
| 2016 |

The **columns** organize and display values from your data vertically. Similar to rows, columns can be pulled directly from the data set or created using **values**. **Values** are used to calculate and count data. This is where you input the variables you want to measure. This is also how you create calculated fields in your pivot table. As a refresher, a **calculated field** is a new field within a pivot table that carries out certain calculations based on the values of other fields

In the previous movie data example, the Values editor created columns for the pivot table, including the SUM of Box Office Revenue, the AVERAGE of Box Office Revenue, and the COUNT of Box Office Revenue columns.

| SUM of Box Office Revenue ($) | AVERAGE of Box Office Revenue ($) | COUNT of Box Office Revenue ($) |
|---|---|---|
| $18,078,040,000.00 | $170,547,547.17 | 106 |
| $13,672,800,000.00 | $160,856,470.59 | 85 |
| $20,013,420,000.00 | $168,180,000.00 | 119 |
| $13,521,310,000.00 | $109,042,822.58 | 124 |
| $11,921,900,000.00 | $161,106,756.76 | 74 |
| **$77,207,470,000.00** | **$151,983,208.66** | **508** |

Finally, the **filters** section of a pivot table enables you to apply filters based on specific criteria — just like filters in regular spreadsheets! For example, a filter was added to the movie data pivot table so that it only included movies that generated less than $10 million in revenue.

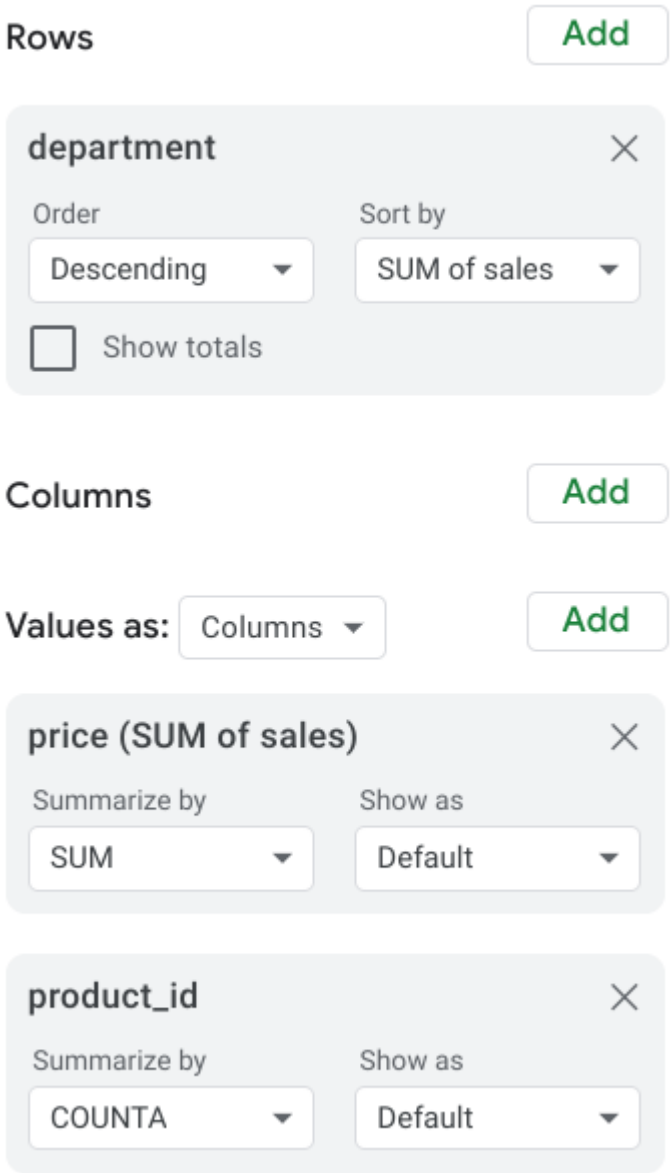
| *Release Date - Year* | SUM < $10M | AVERAGE < $10 M | COUNT < $10 M |
|---|---|---|---|
| 2012 | $18,078,040,000.00 | $170,547,547.17 | 106 |
| 2013 | $13,672,800,000.00 | $160,856,470.59 | 85 |
| 2014 | $20,013,420,000.00 | $168,180,000.00 | 119 |
| 2015 | $13,521,310,000.00 | $109,042,822.58 | 124 |
| 2016 | $11,921,900,000.00 | $161,106,756.76 | 74 |
| **Grand Total** | **$77,207,470,000.00** | **$151,983,208.66** | **508** |

Being able to use all four parts of the pivot table editor will allow you to compare different metrics from your data and execute calculations, which will help you gain valuable insights.

Using pivot tables for analysis

Pivot tables can be a useful tool for answering specific questions about a dataset so you can quickly share answers with stakeholders. For example, a data analyst working at a department store was asked to determine the total sales for each department and the number of products they each sold. They were also interested in knowing exactly which department generated the most revenue.

Instead of making changes to the original spreadsheet data, they used a pivot table to answer these questions and easily compare the sales revenue and number of products sold by each department.




They used the department as the rows for this pivot table to group and organize the rest of the sales data. Then, they input two Values as columns: the SUM of sales and a count of the

products sold. They also sorted the data by the SUM of sales column in order to determine which department generated the most revenue.

| *department* | SUM of sales | COUNTA of product_id |
|---|---|---|
| Toys | $3,045.95 | 49 |
| Beauty | $2,958.37 | 55 |
| Movies | $2,880.55 | 57 |
| Tools | $2,869.96 | 48 |
| Games | $2,785.80 | 49 |
| Industrial | $2,728.90 | 51 |
| Jewelry | $2,669.25 | 52 |
| Health | $2,613.56 | 48 |
| Automotive | $2,589.56 | 47 |
| Garden | $2,586.92 | 45 |
| Sports | $2,460.12 | 46 |
| Grocery | $2,459.22 | 44 |
| Outdoors | $2,399.48 | 47 |
| Electronics | $2,353.65 | 41 |
| Books | $2,313.01 | 42 |
| Baby | $2,272.77 | 46 |
| Home | $2,222.99 | 38 |
| Computers | $2,206.20 | 44 |
| Kids | $2,117.98 | 41 |
| Shoes | $2,108.33 | 39 |
| Clothing | $1,858.47 | 38 |
| Music | $1,809.36 | 33 |

Now they know that the Toys department generated the most revenue!

Pivot tables are an effective tool for data analysts working with spreadsheets because they highlight key insights from the spreadsheet data without having to make changes to the spreadsheet. Coming up, you will create your own pivot table to analyze data and identify trends that will be highly valuable to stakeholders.

**Using pivot tables in analysis**

In this reading, you will learn how to create and use pivot tables for data analysis. You will also get some resources about pivot tables that you can save for your own reference when you start creating pivot tables yourself. **Pivot tables** are a spreadsheet tool that let you view data in multiple ways to find insights and trends.

Pivot tables allow you to make sense of large data sets by giving you tools to easily compare metrics, quickly perform calculations, and generate readable reports. You can create a pivot table to help you answer specific questions about your data. For example, if you were analyzing sales data, you could use pivot tables to answer questions like, "Which month had the most sales?" and "What products generated the most revenue this year?" When you need answers to questions about your data, pivot tables can help you cut through the clutter and focus on only the data you need.

Create your pivot table

Before you can analyze data with pivot tables, you will need to create a pivot table with your data. The following includes the steps for creating a pivot table in Google Sheets, but most spreadsheet programs will have similar tools.

First, you will open the **Insert** menu from the toolbar; there will be an option for **Pivot table**.

This pop-up menu will appear:

There is an option to select New sheet or Existing sheet and a Create button

Generally, you will want to create a new sheet for your pivot table to keep your raw data and your analysis separate. You can also store all of your calculations in one place for easy reference. Once you have created your pivot table, there will be a pivot table editor that you can access to the right of your data.

This is where you will be able to customize your pivot table, including what variables you want to include for your analysis.

Using your pivot table for analysis

You can perform a wide range of analysis tasks with your pivot tables to quickly draw meaningful insights from your data, including performing calculations, sorting, and filtering your data. Below is a list of online resources that will help you learn about performing basic calculations in pivot tables as well as resources for learning about sorting and filtering data in your pivot tables.

**Perform calculations**

| Microsoft Excel | Google Sheets |
| --- | --- |
| **Calculate values in a pivot table:** Microsoft Support's introduction to calculations in Excel pivot tables. This is a useful starting point if you are learning how to perform calculations with pivot tables specifically in Excel. | **Create and use pivot tables:** This guide is fo using pivot tables in Google Sheets and it pr instructions for creating calculated fields. Thi how-to guide you can save and reference as reminder on how to add calculated fields. |

| Microsoft Excel | Google Sheets |
| --- | --- |
| **Pivot table calculated field example:** This resource includes a detailed example of a pivot table being used for calculations. This step-by-step process demonstrates how calculated fields work, and provides you with some idea of how they can be used for analysis. | **All about calculated field in pivot tables:** T comprehensive guide to calculated fields for Sheets. If you are working with Sheets and a interested in learning more about pivot table great resource. |
| **Pivot table calculated fields: step-by-step tutorial:** This tutorial for creating your own calculated fields in pivot tables is a really useful resource to save and bookmark for when you start to apply calculated fields to your own spreadsheets. | **Pivot tables in Google Sheets:** This beginne covers the basics of pivot tables and calculat Google Sheets and uses examples and how-t help demonstrate these concepts. |

**Sort your data**

| Microsoft Excel | Google Sheets |
| --- | --- |
| **Sort data in a pivot table or PivotChart:** This is a Microsoft Support how-to guide to sorting data in pivot tables. This is a useful reference if you are working with Excel and are interested in checking out how filtering will appear in Excel specifically. | **Customize a pivot table:** This guide from Go Support focuses on sorting pivot tables in Go Sheets. This is a useful, quick reference if you working on sorting data in Sheets and need a step guide. |
| **Pivot tables- Sorting data:** This tutorial for sorting data in pivot tables includes an example with real data that demonstrates how sorting in Excel pivot tables works. This example is a great way to experience the entire process from start to finish. | **How to sort pivot table columns:** This detai uses real data to demonstrate how the sorting for Google Sheet pivot tables will work. This is resource if you need a slightly more detailed screenshots of the actual Sheets environment |
| **How to sort a pivot table by value:** This source uses an example to explain sorting by value in pivot tables. It includes a video, which is a useful guide if you need a demonstration of the process. | **Pivot table ascending and descending orde** minute beginner's guide is a great way to bru sorting in pivot tables if you are interested in refresher. |

**Filter your data**

| Microsoft Excel | Google Sheets |
| --- | --- |
| **Filter data in a pivot table:** This resource from the Microsoft Support page provides an explanation of filtering data in pivot tables in Excel. If you are working in Excel spreadsheets, this is a great resource to have bookmarked for quick reference. | **Customize a pivot table:** This is the Google Su page on filtering pivot table data. This is a usef if you are working with pivot tables in Google S need a quick resource to review the process. |
| **How to filter Excel pivot table data:** This how-to guide for filtering data in pivot tables demonstrates the filtering process in an Excel spreadsheet with data | **Filter multiple values in pivot table:** This gui details about how to filter for multiple values i Sheet pivot tables. This resource expands some |

| Microsoft Excel | Google Sheets |
| --- | --- |
| and includes tips and reminders for when you start using these tools on your own. | functionality that you have already learned and up to create more complex filters in Google Sh |

**Format your data**

| Microsoft Excel | Google Sheets |
| --- | --- |
| **Design the layout and format of a PivotTable**: This Microsoft Support article describes how to change the format of the PivotTable by applying a predefined style, banded rows, and conditional formatting. | **Create and edit pivot tables**: This article provides information about h a pivot table to change its style, and data. |

Pivot tables are a powerful tool that you can use to quickly perform calculations and gain meaningful insights into your data directly from the spreadsheet file you are working in! By using pivot table tools to calculate, sort, and filter your data, you can immediately make high-level observations about your data that you can share with stakeholders in reports.

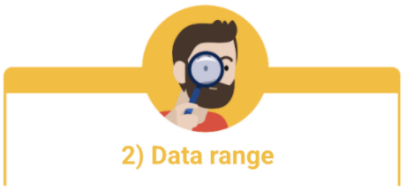
But, like most tools we have covered in this course, the best way to learn is to practice. This was just a small taste of what you can do with pivot tables, but the more you work with pivot tables, the more you will discover.

**Types of data validation**

This reading describes the purpose, examples, and limitations of six types of data validation. The first five are validation types associated with the data (type, range, constraint, consistency, and structure) and the sixth type focuses on the validation of application code used to accept data from user input.

As a junior data analyst, you might not perform all of these validations. But you could ask if and how the data was validated before you begin working with a dataset. Data validation helps to ensure the integrity of data. It also gives you confidence that the data you are using is clean. The following list outlines six types of data validation and the purpose of each, and includes examples and limitations.


- **Purpose**: Check that the data matches the data type defined for a field.
- **Example**: Data values for school grades 1-12 must be a numeric data type.
- **Limitations**: The data value 13 would pass the data type validation but would be an unacceptable value. For this case, data range validation is also needed.



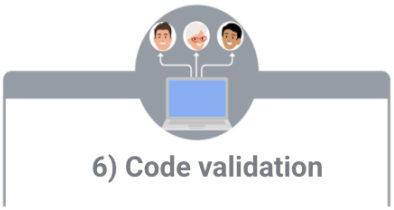

- **Purpose**: Check that the data falls within an acceptable range of values defined for the field.
- **Example**: Data values for school grades should be values between 1 and 12.
- **Limitations**: The data value 11.5 would be in the data range and would also pass as a numeric data type. But, it would be unacceptable because there aren't half grades. For this case, data constraint validation is also needed.


- **Purpose**: Check that the data meets certain conditions or criteria for a field. This includes the type of data entered as well as other attributes of the field, such as number of characters.
- **Example**: Content constraint: Data values for school grades 1-12 must be whole numbers.
- **Limitations**: The data value 13 is a whole number and would pass the content constraint validation. But, it would be unacceptable since 13 isn't a recognized school grade. For this case, data range validation is also needed.


- **Purpose**: Check that the data makes sense in the context of other related data.
- **Example**: Data values for product shipping dates can't be earlier than product production dates.
- **Limitations**: Data might be consistent but still incorrect or inaccurate. A shipping date could be later than a production date and still be wrong.


- **Purpose**: Check that the data follows or conforms to a set structure.

- **Example**: Web pages must follow a prescribed structure to be displayed properly.
- **Limitations**: A data structure might be correct with the data still incorrect or inaccurate. Content on a web page could be displayed properly and still contain the wrong information.




- **Purpose:** Check that the application code systematically performs any of the previously mentioned validations during user data input.
- **Example:** Common problems discovered during code validation include: more than one data type allowed, data range checking not done, or ending of text strings not well defined.
- **Limitations:** Code validation might not validate all possible variations with data input.

# 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.