## 1. Summarizing subsets

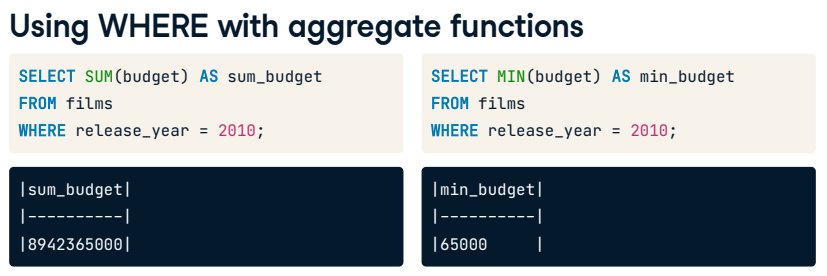
Well done! Now we will combine our filtering skills with our new summarizing skills.

## 2. Using WHERE with aggregate functions



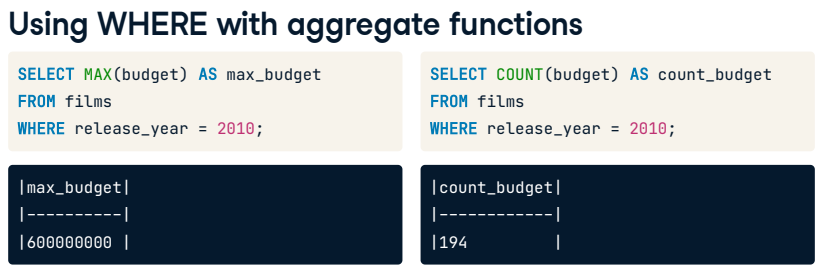
We can combine aggregate functions with the WHERE clause to gain further insights from our data. That's because the WHERE clause executes before the SELECT statement. For example, to get the average budget of movies made in 2010 or later, we would select the average of the budget field from the films table where the release year is greater than or equal to 2010.

## 3. Using WHERE with aggregate functions



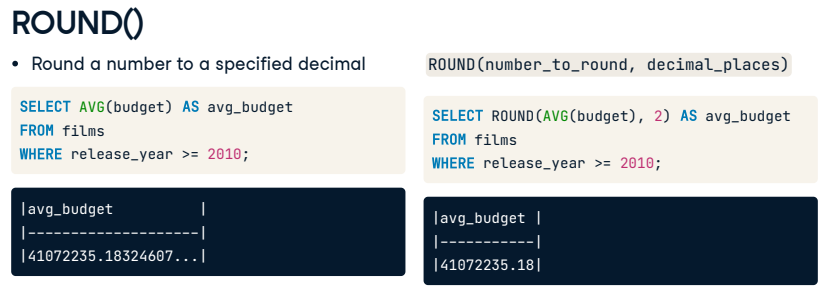
Here are a few more examples using the other functions: we find the total budget of movies made in 2010 using the SUM function, that's over 8.9 billion! Next, we get the smallest budget using the MIN function, which is 65,000.

## 4. Using WHERE with aggregate functions



Here, we query the highest budget using the MAX function. 600 million feels like a lot again for a movie budget, but this is in Indian Rupees for the movie "Kites". Finally, we query the count of the number of budgets using the COUNT function, which gives us the total number of non-missing values in the budget field, meaning there are 194 budgets recorded for the year 2010 in the films table.

## 5. ROUND()



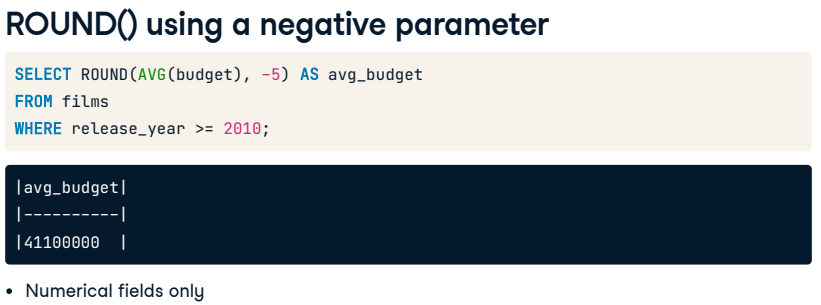
Now that we are doing all sorts of things with our numerical values, we'll likely want to clean up some of the crazy decimals that might appear. In SQL, we can use ROUND() to round our number to a specified decimal. There are two parameters for ROUND(): the number we want to round and the decimal place we want to round to. Here we have re-calculated the same average budget as before, but this time we have included ROUND() and specified we want to round to two decimal places because we are dealing with currency.

## 6. ROUND() to a whole number



The second parameter in our ROUND() function is optional, so we can leave it out if we want to round to a whole number. We would get the same result if we passed zero as the second argument, as it is the default when no number is given.

## 7. ROUND() using a negative parameter



Here is a tricky one: we could also pass a negative number as the second parameter and still get a result. Here, the function is rounding to the left of the decimal point instead of the right. Using negative five as the decimal place parameter will cause the function to round to the hundred thousand or five places to the left. ROUND() can only be used with numerical fields.

## 8. Let's practice!

Let's practice!

## Exercise

# Combining aggregate functions with WHERE

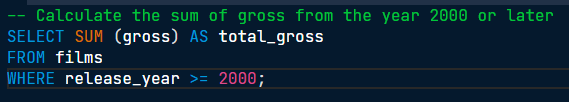
When combining aggregate functions with WHERE, you get a powerful tool that allows you to get more granular with your insights, for example, to get the total budget of movies made from the year 2010 onwards.

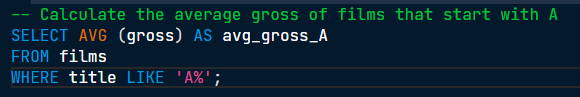
This combination is useful when you only want to summarize a subset of your data. In your film-industry role, as an example, you may like to summarize each certification category to compare how they each perform or if one certification has a higher average budget than another.

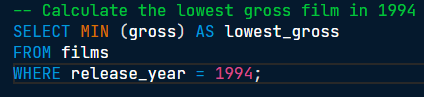
Let's see what insights you can gain about the financials in the dataset.

## Instructions

* Use SUM() to calculate the total gross for all films made in the year 2000 or later, and use the alias total\_gross.
* Calculate the average amount grossed by all films whose titles start with the letter 'A' and alias with avg\_gross\_A.
* Calculate the lowest gross film in 1994 and use the alias lowest\_gross.
  + Calculate the highest gross film between 2000 and 2012, inclusive, and use the alias highest\_gross.









Nice. SQL provides us with several building blocks that we can combine in all kinds of ways, hence the name: Structured Query Language.

## Exercise

# Using ROUND()

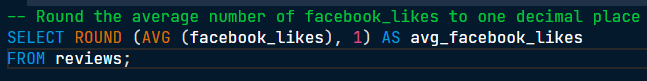
Aggregate functions work great with numerical values; however, these results can sometimes get unwieldy when dealing with long decimal values. Luckily, SQL provides you with the ROUND() function to tame these long decimals.

If asked to give the average budget of your films, ten decimal places is not necessary. Instead, you can round to two decimal places to create results that make more sense for currency.

Now you try!

## Instructions

* Calculate the average facebook\_likes to one decimal place and assign to the alias, avg\_facebook\_likes.



Well done! The average, rounded facebook\_likes is 7802.9. This insight can be used as a benchmark to measure film reviews; any film with over 7802.9 likes can be considered popular.

## Exercise

# ROUND() with a negative parameter

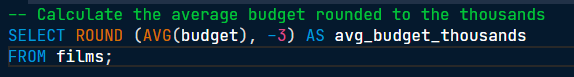
A useful thing you can do with ROUND() is have a negative number as the decimal place parameter. This can come in handy if your manager only needs to know the average number of facebook\_likes to the hundreds since granularity below one hundred likes won't impact decision making.

Social media plays a significant role in determining success. If a movie trailer is posted and barely gets any likes, the movie itself may not be successful. Remember how 2020's "Sonic the Hedgehog" movie got a revamp after the public saw the trailer?

Let's apply this to other parts of the dataset and see what the benchmark is for movie budgets so, in the future, it's clear whether the film is above or below budget.

## Instructions

* Calculate the average budget from the films table, aliased as avg\_budget\_thousands, and round to the nearest thousand.



ROUND() of applause! The ROUND() function is very handy when making financial calculations to get a top-level view or specify to the penny or cent.