**LESSON AGGREGATE FUNCTIONS**

**Introduction**

3 min

SQL Queries don’t just access raw data, they can also perform calculations on the raw data to answer specific data questions.

Calculations performed on multiple rows of a table are called **aggregates**.

In this lesson, we have given you a table named fake\_apps which is made up of fake mobile applications data.

Here is a quick preview of some important aggregates that we will cover in the next five exercises:

* Preview: Docs Loading link description

[COUNT()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/count?page_ref=catalog)

: count the number of rows

* Preview: Docs Loading link description

[SUM()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/sum?page_ref=catalog)

: the sum of the values in a column

* Preview: Docs Loading link description

[MAX()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/max?page_ref=catalog)

/

Preview: Docs Loading link description

[MIN()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/min?page_ref=catalog)

: the largest/smallest value

* Preview: Docs Loading link description

[AVG()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/avg?page_ref=catalog)

: the average of the values in a column

* Preview: Docs Loading link description

[ROUND()](https://www.codecademy.com/resources/docs/sql/commands/round?page_ref=catalog)

: round the values in the column

Let’s get started!

**Instructions**

1. Checkpoint 1 Passed

**1.**

Before getting started, take a look at the data in the fake\_apps table.

In the code editor, type the following:

SELECT \*  
FROM fake\_apps;

Copy to Clipboard

What are the column names?

Hint

The column names are id, name, category, downloads, and price.

**aggregate-functions.sqlite**

SELECT \*

FROM fake\_apps;





\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Count**

3 min

The fastest way to calculate how many rows are in a table is to use the COUNT() function.

Preview: Docs Loading link description

[COUNT()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/count?page_ref=catalog)

 is a function that takes the name of a column as an argument and counts the number of non-empty values in that column.

SELECT COUNT(\*)  
FROM table\_name;

Copy to Clipboard

Here, we want to count every row, so we pass \* as an argument inside the parenthesis.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Let’s count how many apps are in the table.

In the code editor, run:

SELECT COUNT(\*)   
FROM fake\_apps;

Copy to Clipboard

Hint

There are 200 apps.

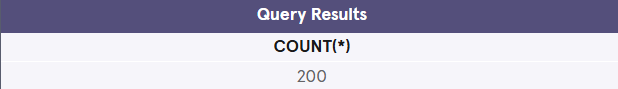
Common errors:

* + Missing parenthesis.
  + Missing ;.

**count.sqlite**

SELECT COUNT(\*)

FROM fake\_apps;



**2.**

Add a WHERE clause in the previous query to count how many *free* apps are in the table.

Hint

Remember the WHERE statement?

The following code should go inside the previous query, before the semicolon:

SELECT COUNT(\*)   
FROM fake\_apps  
WHERE price = 0;

Copy to Clipboard

* WHERE indicates we want to only include rows where the following condition is true.
* price = 0 is the condition.

There are 73 free apps in the table.

**count.sqlite**

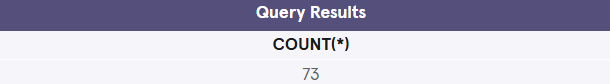
/\*SELECT COUNT(\*)

FROM fake\_apps;\*/

SELECT COUNT(\*)

FROM fake\_apps

WHERE price = 0.0;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Sum**

1 min

SQL makes it easy to add all values in a particular column using SUM().

Preview: Docs Loading link description

[SUM()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/sum?page_ref=catalog)

 is a function that takes the name of a column as an argument and returns the sum of all the values in that column.

What is the total number of downloads for all of the apps combined?

SELECT SUM(downloads)  
FROM fake\_apps;

Copy to Clipboard

This adds all values in the downloads column.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Let’s find out the answer!

In the code editor, type:

SELECT SUM(downloads)  
FROM fake\_apps;

Copy to Clipboard

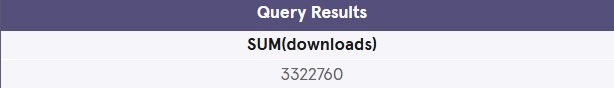
Hint

There are 3,322,760 total downloads.

**sum.sqlite**

SELECT SUM(downloads)

FROM fake\_apps;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Max / Min**

2 min

The MAX() and MIN() functions return the highest and lowest values in a column, respectively.

How many downloads does the most popular app have?

SELECT MAX(downloads)  
FROM fake\_apps;

Copy to Clipboard

The most popular app has 31,090 downloads!

Preview: Docs Loading link description

[MAX()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/max?page_ref=catalog)

 takes the name of a column as an argument and returns the largest value in that column. Here, we returned the largest value in the downloads column.

Preview: Docs Loading link description

[MIN()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/min?page_ref=catalog)

 works the same way but it does the exact opposite; it returns the smallest value.

**Instructions**

1. Checkpoint 1 Passed

**1.**

What is the least number of times an app has been downloaded?

In the code editor, type:

SELECT MIN(downloads)  
FROM fake\_apps;

Copy to Clipboard

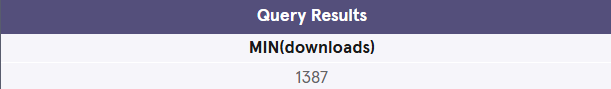
Hint

1,387 downloads.

**max.sqlite**

SELECT MIN(downloads)

FROM fake\_apps;



**2.**

Delete the previous query.

Write a new query that returns the price of the most expensive app.

Hint

SELECT MAX(price)  
FROM fake\_apps;

Copy to Clipboard

$14.99 is the price of the most expensive app.

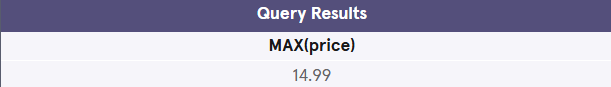
**max.sqlite**

/\*SELECT MIN(downloads)

FROM fake\_apps;\*/

SELECT MAX(price)

FROM fake\_apps;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Average**

1 min

SQL uses the

Preview: Docs Loading link description

[AVG()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/avg?page_ref=catalog)

 function to quickly calculate the average value of a particular column.

The statement below returns the average number of downloads for an app in our database:

SELECT AVG(downloads)  
FROM fake\_apps;

Copy to Clipboard

The AVG() function works by taking a column name as an argument and returns the average value for that column.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Calculate the average number of downloads for all the apps in the table.

In the code editor, type:

SELECT AVG(downloads)  
FROM fake\_apps;

Copy to Clipboard

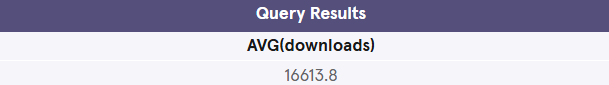
Hint

16,613.8 average downloads.

**avg.sqlite**

SELECT AVG(downloads)

FROM fake\_apps;



**2.**

Remove the previous query.

Write a new query that calculates the average price for all the apps in the table.

Hint

Which column should go inside the parenthesis?

SELECT AVG(\_\_\_\_\_)  
 FROM fake\_apps;

Copy to Clipboard

The average price is $2.02365.

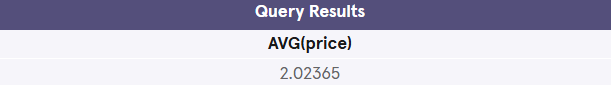
**avg.sqlite**

/\*SELECT AVG(downloads)

FROM fake\_apps;\*/

SELECT AVG(price)

FROM fake\_apps;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Round**

5 min

By default, SQL tries to be as precise as possible without rounding. We can make the result table easier to read using the ROUND() function.

Preview: Docs Loading link description

[ROUND()](https://www.codecademy.com/resources/docs/sql/commands/round?page_ref=catalog)

 function takes two arguments inside the parenthesis:

1. a column name
2. an integer

It rounds the values in the column to the number of decimal places specified by the integer.

SELECT ROUND(price, 0)  
FROM fake\_apps;

Copy to Clipboard

Here, we pass the column price and integer 0 as arguments. SQL rounds the values in the column to 0 decimal places in the output.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Let’s return the name column and a rounded price column.

In the code editor, type:

SELECT name, ROUND(price, 0)  
FROM fake\_apps;

Copy to Clipboard

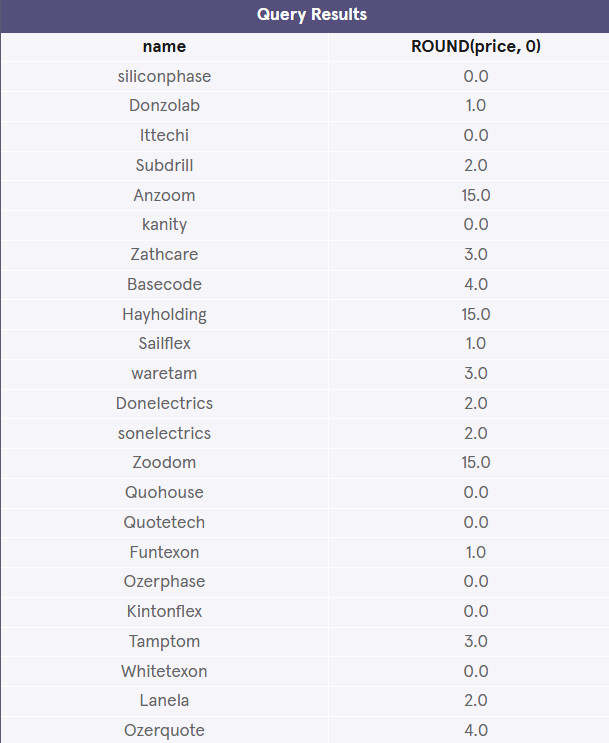
Hint

We are selecting ROUND(price, 0) as the second column in this query.

**round.sqlite**

SELECT name, ROUND(price, 0)

FROM fake\_apps;



**2.**

Remove the previous query.

In the last exercise, we were able to get the average price of an app ($2.02365) using this query:

SELECT AVG(price)  
FROM fake\_apps;

Copy to Clipboard

Now, let’s edit this query so that it rounds this result to 2 decimal places.

This is a tricky one!

Hint

You can treat AVG(price) just like any other value and place it inside the ROUND function like so:

ROUND(AVG(price), 2)

Copy to Clipboard

Here, AVG(price) is the 1st argument and 2 is the 2nd argument because we want to round it to two decimal places:

SELECT ROUND(AVG(price), 2)  
FROM fake\_apps;

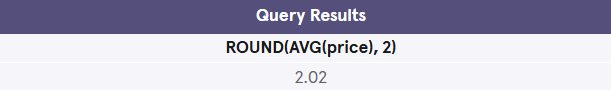
**round.sqlite**

/\*SELECT name, ROUND(price, 0)

FROM fake\_apps;\*/

SELECT ROUND(AVG(price), 2)

FROM fake\_apps;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Group By I**

11 min

Oftentimes, we will want to calculate an aggregate for data with certain characteristics.

For instance, we might want to know the mean IMDb ratings for all movies each year. We could calculate each number by a series of queries with different WHERE statements, like so:

SELECT AVG(imdb\_rating)  
FROM movies  
WHERE year = 1999;  
  
SELECT AVG(imdb\_rating)  
FROM movies  
WHERE year = 2000;  
  
SELECT AVG(imdb\_rating)  
FROM movies  
WHERE year = 2001;

Copy to Clipboard

and so on.

Luckily, there’s a better way!

We can use

Preview: Docs Loading link description

[GROUP BY](https://www.codecademy.com/resources/docs/sql/commands/group-by?page_ref=catalog)

 to do this in a single step:

SELECT year,  
   AVG(imdb\_rating)  
FROM movies  
GROUP BY year  
ORDER BY year;

Copy to Clipboard

GROUP BY is a clause in SQL that is used with

Preview: Docs Loading link description

[aggregate functions](https://www.codecademy.com/resources/docs/sql/aggregate-functions)

. It is used in collaboration with the

Preview: Docs Loading link description

[SELECT](https://www.codecademy.com/resources/docs/sql/commands/select)

 statement to arrange identical data into *groups*.

The GROUP BY statement comes after any

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 statements, but before

Preview: Docs Loading link description

[ORDER BY](https://www.codecademy.com/resources/docs/sql/commands/order-by?page_ref=catalog)

 or

Preview: Docs Loading link description

[LIMIT](https://www.codecademy.com/resources/docs/sql/commands/limit?page_ref=catalog)

.

**Instructions**

1. Checkpoint 1 Passed

**1.**

In the code editor, type:

SELECT price, COUNT(\*)   
FROM fake\_apps  
GROUP BY price;

Copy to Clipboard

Here, our aggregate function is COUNT() and we arranged price into groups.

What do you expect the result to be?

Hint

The result contains the total number of apps for each price.

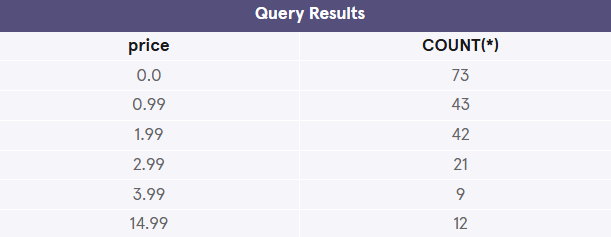
It is organized into two columns, making it very easy to see the number of apps at each price.

**groupby.sqlite**

SELECT price, COUNT(\*)

FROM fake\_apps

GROUP BY price;



**2.**

In the previous query, add a WHERE clause to count the total number of apps that have been downloaded more than 20,000 times, at each price.

Hint

Remember, WHERE statement goes *before* the GROUP BY statement:

SELECT price, COUNT(\*)   
FROM fake\_apps  
WHERE downloads > 20000  
GROUP BY price;

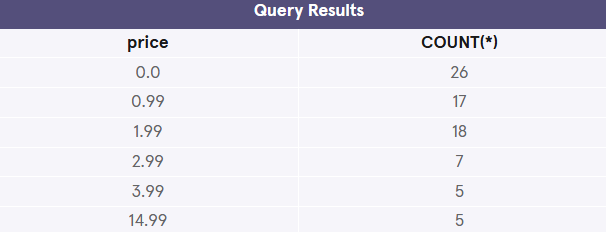
**groupby.sqlite**

SELECT price, COUNT(\*)

FROM fake\_apps

WHERE downloads > 20000

GROUP BY price;



**3.**

Remove the previous query.

Write a new query that calculates the total number of downloads for each category.

Select category and SUM(downloads).

Hint

First, select the two columns we want:

SELECT category, SUM(downloads)  
FROM fake\_apps;

Copy to Clipboard

Next, group the result for each category by adding a GROUP BY:

SELECT category, SUM(downloads)  
FROM fake\_apps  
GROUP BY category;

**groupby.sqlite**

/\*SELECT price, COUNT(\*)

FROM fake\_apps

WHERE downloads > 20000

GROUP BY price;\*/

SELECT category, SUM(downloads)

FROM fake\_apps

GROUP BY category;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Group By II**

5 min

Sometimes, we want to

Preview: Docs Loading link description

[GROUP BY](https://www.codecademy.com/resources/docs/sql/commands/group-by?page_ref=catalog)

 a calculation done on a column.

For instance, we might want to know how many movies have IMDb ratings that round to 1, 2, 3, 4, 5. We could do this using the following syntax:

SELECT ROUND(imdb\_rating),  
   COUNT(name)  
FROM movies  
GROUP BY ROUND(imdb\_rating)  
ORDER BY ROUND(imdb\_rating);

Copy to Clipboard

However, this query may be time-consuming to write and more prone to error.

SQL lets us use column reference(s) in our GROUP BY that will make our lives easier.

* 1 is the first column selected
* 2 is the second column selected
* 3 is the third column selected

and so on.

The following query is equivalent to the one above:

SELECT ROUND(imdb\_rating),  
   COUNT(name)  
FROM movies  
GROUP BY 1  
ORDER BY 1;

Copy to Clipboard

Here, the 1 refers to the first column in our

Preview: Docs Loading link description

[SELECT](https://www.codecademy.com/resources/docs/sql/commands/select)

 statement, ROUND(imdb\_rating).

**Instructions**

1. Checkpoint 1 Passed

**1.**

Suppose we have the query below:

SELECT category,   
   price,  
   AVG(downloads)  
FROM fake\_apps  
GROUP BY category, price;

Copy to Clipboard

Write the exact query, but use column reference numbers instead of column names after GROUP BY.

Hint

These numbers represent the selected columns:

* + 1 refers to category.
  + 2 refers to price.
  + 3 refers to AVG(downloads)

Now, change the GROUP BY with numbers:

SELECT category,   
   price,  
   AVG(downloads)  
FROM fake\_apps  
GROUP BY 1, 2;

Copy to Clipboard

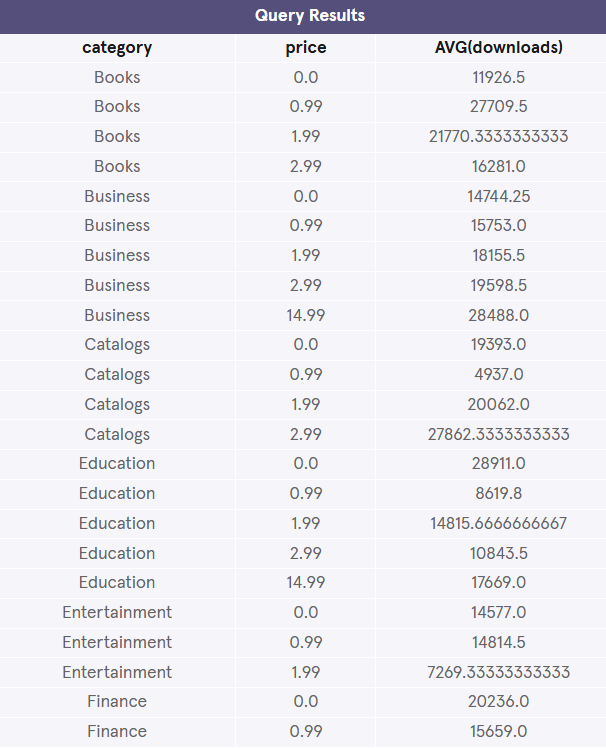
Note: Even if you use column names instead of numbers, it will still be correct because these two queries are exactly the same!

**groupby-ii.sqlite**

SELECT category, price, AVG(downloads)

FROM fake\_apps

GROUP BY 1, 2;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Having**

7 min

In addition to being able to group data using

Preview: Docs Loading link description

[GROUP BY](https://www.codecademy.com/resources/docs/sql/commands/group-by?page_ref=catalog)

, SQL also allows you to filter which groups to include and which to exclude.

For instance, imagine that we want to see how many movies of different genres were produced each year, but we only care about years and genres with at least 10 movies.

We can’t use

Preview: Docs Loading link description

[WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog)

 here because we don’t want to filter the rows; we want to *filter groups*.

This is where

Preview: Docs A conditional statement similar to WHERE but used with aggregate functions (COUNT(), MIN(), MAX(), SUM(), AVG()).

[HAVING](https://www.codecademy.com/resources/docs/sql/commands/having?page_ref=catalog)

 comes in.

HAVING is very similar to WHERE. In fact, all types of WHERE clauses you learned about thus far can be used with HAVING.

We can use the following for the problem:

SELECT year,  
   genre,  
   COUNT(name)  
FROM movies  
GROUP BY 1, 2  
HAVING COUNT(name) > 10;

Copy to Clipboard

* When we want to limit the results of a query based on values of the individual rows, use WHERE.
* When we want to limit the results of a query based on an aggregate property, use HAVING.

HAVING statement always comes after GROUP BY, but before

Preview: Docs Loading link description

[ORDER BY](https://www.codecademy.com/resources/docs/sql/commands/order-by?page_ref=catalog)

 and

Preview: Docs Loading link description

[LIMIT](https://www.codecademy.com/resources/docs/sql/commands/limit?page_ref=catalog)

.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Suppose we have the query below:

SELECT price,   
   ROUND(AVG(downloads)),  
   COUNT(\*)  
FROM fake\_apps  
GROUP BY price;

Copy to Clipboard

It returns the average downloads (rounded) and the number of apps – at each price point.

However, certain price points don’t have very many apps, so their average downloads are less meaningful.

Add a HAVING clause to restrict the query to price points that have more than 10 apps.

Hint

The total number of apps at each price point would be given by COUNT(\*).

SELECT price,   
   ROUND(AVG(downloads)),  
   COUNT(\*)  
FROM fake\_apps  
GROUP BY price  
HAVING COUNT(\*) > 10;

Copy to Clipboard

COUNT(\*) > 10 is the condition.

Because the condition has an aggregate function in it, we have to use HAVING instead of WHERE.

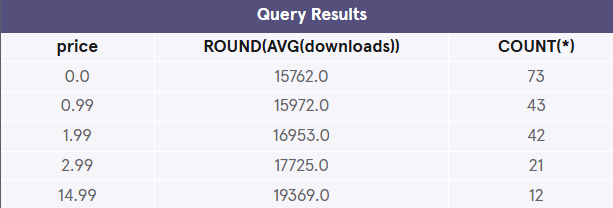
**having.sqlite**

SELECT price, ROUND(AVG(downloads)), COUNT(\*)

FROM fake\_apps

GROUP BY price

HAVING COUNT(\*) > 10;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Review**

<1 min

Congratulations!

You just learned how to use aggregate functions to perform calculations on your data. What can we generalize so far?

* Preview: Docs Loading link description

[COUNT()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/count?page_ref=catalog)

: count the number of rows

* Preview: Docs Loading link description

[SUM()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/sum?page_ref=catalog)

: the sum of the values in a column

* Preview: Docs Loading link description

[MAX()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/max?page_ref=catalog)

/

Preview: Docs Loading link description

[MIN()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/min?page_ref=catalog)

: the largest/smallest value

* Preview: Docs Loading link description

[AVG()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/avg?page_ref=catalog)

: the average of the values in a column

* Preview: Docs Loading link description

[ROUND()](https://www.codecademy.com/resources/docs/sql/commands/round?page_ref=catalog)

: round the values in the column

*Aggregate functions* combine multiple rows together to form a single value of more meaningful information.

* Preview: Docs Loading link description

[GROUP BY](https://www.codecademy.com/resources/docs/sql/commands/group-by?page_ref=catalog)

 is a clause used with aggregate functions to combine data from one or more columns.

* Preview: Docs Loading link description

[HAVING](https://www.codecademy.com/resources/docs/sql/commands/having?page_ref=catalog)

 limit the results of a query based on an aggregate property.

**Instructions**

Feel free to experiment a bit more with the fake\_apps table before moving on!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROJECT – TRENDS IN STARTUPS**

**Trends in Startups**

Howdy! It’s your first day as a [TechCrunch](https://techcrunch.com/) reporter. Your first task is to write an article on the rising trends in the startup world.

To get you started with your research, your boss emailed you a **project.sqlite** file that contains a table called startups. It is a portfolio of some of the biggest names in the industry.

Write queries with aggregate functions to retrieve some interesting insights about these companies.

What are you waiting for? Let’s get started!

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Unstuck**“ to see a **project walkthrough video**.

**Tasks**

14/14 complete

Mark the tasks as complete by checking them off

**Write the following queries:**

**1.**

Getting started, take a look at the startups table:

SELECT \*  
FROM startups;

Copy to Clipboard

How many columns are there?

Hint

You can expand the right panel (or scroll right) to take a look at all the columns.

The startups table has ten columns:

* + name
  + location
  + category
  + employees
  + raised
  + valuation
  + founded
  + stage
  + ceo
  + info

Do you recognize any of these companies?

**2.**

Calculate the total number of companies in the table.

Hint

SELECT COUNT(\*)  
FROM startups;

Copy to Clipboard

There are 70 companies in the table.

**3.**

We want to know the total value of all companies in this table.

Calculate this by getting the SUM() of the valuation column.

Hint

SELECT SUM(valuation)  
FROM startups;

Copy to Clipboard

The sum is $974,455,790,000!

**4.**

What is the highest amount raised by a startup?

Return the maximum amount of money raised.

Hint

SELECT MAX(raised)  
FROM startups;

Copy to Clipboard

**5.**

Edit the query so that it returns the maximum amount of money raised, during ‘Seed’ stage.

Hint

SELECT MAX(raised)  
FROM startups  
WHERE stage = 'Seed';

Copy to Clipboard

**6.**

In what year was the oldest company on the list founded?

Hint

SELECT MIN(founded)  
FROM startups;

Copy to Clipboard

**Let's find out the valuations among different sectors:**

**7.**

Return the average valuation.

Hint

SELECT AVG(valuation)  
FROM startups;

Copy to Clipboard

**8.**

Return the average valuation, in each category.

Hint

SELECT category, AVG(valuation)  
FROM startups  
GROUP BY category;

Copy to Clipboard

**9.**

Return the average valuation, in each category.

Round the averages to two decimal places.

Hint

SELECT category, ROUND(AVG(valuation), 2)  
FROM startups  
GROUP BY category;

Copy to Clipboard

**10.**

Return the average valuation, in each category.

Round the averages to two decimal places.

Lastly, order the list from highest averages to lowest.

Hint

SELECT category, ROUND(AVG(valuation), 2)  
FROM startups  
GROUP BY 1  
ORDER BY 2 DESC;

Copy to Clipboard

Health Care startups seem to have higher valuations than other categories.

**What are the most competitive markets?**

**11.**

First, return the name of each category with the total number of companies that belong to it.

Hint

SELECT category, COUNT(\*)  
FROM startups  
GROUP BY category;

Copy to Clipboard

**12.**

Next, filter the result to only include categories that have more than three companies in them.

What are the most competitive markets?

Hint

Because you are filtering on a condition that has an aggregate function, you need to use HAVING instead of WHERE:

SELECT category, COUNT(\*)  
FROM startups  
GROUP BY category  
HAVING COUNT(\*) > 3;

Copy to Clipboard

If you want to go a step further, sort the result using ORDER BY:

SELECT category, COUNT(\*)  
FROM startups  
GROUP BY category  
HAVING COUNT(\*) > 3  
ORDER BY 2 DESC;

Copy to Clipboard

The most competitive markets are:

* 1. Social
  2. Mobile
  3. Education

**Let's see if there's a difference in startups sizes among different locations:**

**13.**

What is the average size of a startup in each location?

Hint

SELECT location, AVG(employees)  
FROM startups  
GROUP BY location;

Copy to Clipboard

**14.**

What is the average size of a startup in each location, with average sizes above 500?

Hint

SELECT location, AVG(employees)  
FROM startups  
GROUP BY location  
HAVING AVG(employees) > 500;

**project.sqlite**

SELECT \*

FROM startups;

SELECT COUNT(\*)

FROM startups;

SELECT SUM(valuation)

FROM startups;

SELECT MAX(raised)

FROM startups

WHERE stage = 'Seed';

SELECT MIN(founded)

FROM startups;

SELECT AVG(valuation)

FROM startups;

SELECT category, AVG(valuation)

FROM startups

GROUP BY category;

SELECT category, ROUND(AVG(valuation), 2)

FROM startups

GROUP BY category;

SELECT category, ROUND(AVG(valuation), 2)

FROM startups

GROUP BY 1

ORDER BY 2 DESC;

SELECT category, COUNT(\*)

FROM startups

GROUP BY category;

SELECT category, COUNT(\*)

FROM startups

GROUP BY (category)

HAVING COUNT(\*) > 3;

SELECT location, AVG(employees)

FROM startups

GROUP BY (location);

SELECT location, AVG(employees)

FROM startups

GROUP BY (location)

HAVING AVG(employees) > 500;

**Query Results**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **name** | **location** | **category** | **employees** | **raised** | **valuation** | **founded** | **stage** | **ceo** | **info** |
| Pied Piper | Silicon Valley | Cloud Computing | 6 | 5000000 | 50000000 | 2014 | A | Richard Hendricks | A Middle-Out Compression Solution |
| Hooli | Silicon Valley | Enterprise | 9000 | 580000000 | 49500000000 | 1997 |  | Gavin Bensen | Hooli Is About People |
| Raviga Capital | Silicon Valley | Venture Capital | 12 | 300000000 | 3000000000 | 2012 |  | Peter Gregory | Share Only In Success |
| Aviato | Silicon Valley | Travel | 3 | 250000 | 2500000 | 2006 | Acquired | Erlich Bachman | Software Aggregation Program |
| SEE FOOD | Silicon Valley | Mobile | 2 |  | 15000000 | 2016 | Acquired | Jian-Yang | The Shazam of Food |
| Forbid | New York | Mobile | 25 | 1400000 | 5000000 | 2013 | Acquired | Charlie Dattolo | Charge Users $10 for Calling Their Ex |
| Soulstice | New York | Fitness | 300 | 30000000 | 120000000 | 2014 | B |  | What are your goals today? |
| E-Corp | New York | Enterprise | 10000 |  | 66000000000 | 2006 |  | Phillip Price | Together We Can Change the Wolrd |
| Allsafe Cybersecurity | New York | Security | 250 | 123000000 | 1000000000 | 2014 |  | Gideon Goddard |  |
| fsociety | Brooklyn | Games | 5 |  |  | 2015 | Stealth | Elliot Alderson | Fun Society Arcade |
| Summit Ice | Los Angeles | Fashion | 2 | 50000 | 800000 | 2015 | Seed | Nathan Fielder | Raising Awareness of the Holocaust |
| The Movement | Los Angeles | Fitness | 3 | 15000 |  | 2015 | Seed | Nathan Fielder | No Gym. No Weights. Just Results. |
| WUPHF! | Scranton | Social | 6 | 25000 | 10000 | 2010 | Seed | Ryan Howard | Cross-Portal Messaging System |
| Fakeblock | Irvine | Mobile | 2 | 3 |  | 2013 | Seed | George Michael | OC's Most Popular Digital Wood Instrument |
| Magic Heap | Fort Lauderdale | Augmented Reality | 500 | 1800000000 | 8000000000 | 2013 | D | Johnny Malkovich | ¯\\_(ツ)\_/¯ |
| HoloGrail | San Francisco | Technology | 30 | 51000000 | 1800000000 | 2016 | B | Daniel Li | Volumetric Display |
| Voice2Txt | San Francisco | Mobile | 16 | 1800000 | 15000000 | 2012 | Seed | Cole Romano | Transcribes Voicemails to Texts |
| V-Arrr | San Francisco | Virtual Reality | 6 | 750000 | 2500000 | 2015 | Seed | Cole Romano | Pirate-Themed VR App |
| REBU | San Francisco | Logistics | 15000 | 11500000000 | 70000000 | 2009 | G | Kravis Talanick | Ridesharing Platform |
| Rave New World | San Francisco | Social | 8 | 250000 | 6000000 | 2014 |  | Sam Guss | Social Media for the EDM Community |
| WaterCnC | San Francisco | Travel | 4000 | 4400000000 | 25000000000 | 2008 | F | Crian Besky | Water Sharing Platform for Cooks and Chefs |
| RankerBot | San Francisco | Algorithms | 5 | 1500000 | 7600000 | 2013 | Acquired | Delaney | Water Sharing Platform for Cooks and Chefs |
| Xandesk | San Francisco | Customer Service | 100 | 75500000 | 640000000 | 2012 | IPO | Ronald Paris | Keep Frowning |
| Everyotherday.me | San Francisco | Mobile | 35 |  |  | 2012 | Seed | Jeremy Lin | Journal that Captures the Special Moments |
| Frontsby | San Francisco |  |  |  |  | 2013 | Seed | Olivia Plath |  |
| 2Smart2Host | Brooklyn | Data Analytics | 4 | 380000 |  | 2014 | Acquired | Evan Nails | Dynamic Pricing for Vacation Rentals |
| StarsNGripes | Boulder | Social | 3 | 2000000 | 560000000 | 2015 | A | Gregory Hilt | Yelp for US Landmarks and Tourist Attractions |
| SoCorny | Omaha | Technology | 65 | 25000000 | 2000000000 | 2013 | B | Elon Husk | Analytics Platform for Corn Production |
| VeVork | New York | Real Estate | 200 | 750000000 | 20000000000 | 2010 | G | Mada Nammneu | Co-relaxing Space for Vampires |
| Brrr | New York | Gambling | 10 | 5000000 | 322000000 | 2012 | A | Solo | eSports Betting |
| BackPax | New York | Education | 2 | 80000 | 14000000 | 2017 | Seed | Jose Ferreira | MOOCs are Boo Hoo |
| CapitalPunishment | New York | Mobile | 15 | 500000 | 10000000 | 2008 | A | Cole Romano | Stock Market Brawler Game |
| Son of a Fletch | New York | Social | 4 | 300000 | 3000000 | 2014 | Seed | CC Vaught | Social Media App for Archery Enthusiasts |
| Yas Queen | New York | Health Care | 300 | 45000000 | 980000000 | 2015 | A | Jamie Brieman | Accelerating Breast Cancer Research |
| Geocities-Lite | New York | Social | 20 | 1000000 | 50000000 | 2012 | A | Jim Jason | Mobile Friendly Geocities |
| Cluster.ly | New York | Big Data Analytics | 10 | 750000 | 15000000 | 2015 | B | Hillary Limegreen |  |
| Codefellas | New York | Education | 50 | 50000000 | 10000000000 | 2012 | A | Zane Sims | The Not So Easy Way to Learn to Code |
| kryptoPark | Palo Alto | Security | 70 | 75000000 | 10000000000 | 2011 | B | Joe Wang | Password Mgmt for Password Mgmt Services |
| zZz | Palo Alto | SaaS | 20 | 50000000 | 800000000 | 2008 | C | Henry Campbell | Dream Tracking/Recording |
| Pita Pan | Palo Alto | Technology | 40 | 30000000 | 5500000000 | 2015 | B | Schuylar Croom | Drone Food Delivery Service |
| nova.ai | Palo Alto | Enterprise | 340 | 3000000 | 25000000 | 2013 | A | Nick Kinsella |  |
| XNA | Palo Alto | Health Care | 250 | 540000000 | 760000000000 | 2012 | C | Owen Pilsner | Genome Database |
| Cloud Nine | Palo Alto | Cloud Computing | 35 | 36000000 | 140000000 | 2013 | A | Jase Farmer |  |
| Scramptons | Long Island | Security | 5 | 200000 | 8000000000 | 2017 | Seed | Henri Bernard | Home Security |
| Carcheesian Plane | Paris | Education | 30 | 3000000 | 50000000 | 2012 | A | Jim Mussen | A Gouda Place to Learn Math |
| Dungees and Dragos | Atlanta | Games | 3 | 300000 | 1600000 | 2014 | Seed | Stuart Stickler | Tabletop Gaming for the Insufferable |
| VaporWear | Virginia Beach | Social | 15 | 450000 | 14000000 | 2017 | Seed | TJ Harris | Social Network for Vaping Enthusiasts |
| RICE Media | Brooklyn | Video Streaming | 3000 | 730000000 |  | 1994 |  | Shayna Craftsman | 2cool4school |
| Townie666 | Brooklyn |  | 2 | 20000 | 80000 | 2017 | Seed | Sonny Li | \*\_\* |
| Kimberly | Brooklyn |  | 2 | 100000 | 8500000 | 2018 | Seed | Jillian Kuzmin | Synthetic Diamonds |
| Wakeful | Brooklyn | Virtual Reality | 3 | 30000 | 1800000 | 2016 | Seed | Maria Epstein | Meditation VR App |
| MLGB | Columbus | E-commerce | 2 | 50000 | 500000 | 2012 | Seed | Xin Xin | Authentic Streetwear or DDoS'ed |
| PostGurt | Savannah | Social | 6 | 550000 | 200000000 | 2013 | Seed | Lauretta Brieman | Subscription Based Yogurt in the Mail |
| Virtual Tour | Chicago | E-commerce | 12 | 750000 | 120000000 | 2014 | Seed | Ari L'Boy | Real Estate for Fake People |
| Eärendil | Los Angeles | Mobile | 2 | 50000 | 200000 | 2009 | Seed | Kent Bergensen | Flashlight App |
| QT | Los Angeles | Social | 3 | 1500000 | 4500000 | 2018 | Seed | Keiko | Who Can Be More Kawaii (づ｡◕‿‿◕｡)づ |
| LiftHub | Los Angeles | Fitness | 30 | 2350000 | 15000000 | 2015 | A | Mike McManus | Eat Clean. Train Dirty. |
| Napflix | Los Angeles | Mobile | 1 |  |  | 2016 | Stealth | Reed Samson | Bootleg Netflix Shows |
| CyShare | Seattle | Social | 11 | 300000 | 10000000 | 2015 | B | Cathy Zhou | Bicycle Sharing Service |
| FizzyWillow | Seattle | Social | 6 | 2000000 | 120000000 | 2014 | C | Sandra Emmerson | Sparkling Wine Review App |
| Q Green | Seattle | Nanotechnology | 12 | 3600000 | 150000000 | 2013 | A | Quan Li |  |
| Friend, Where Is My Vehicle | Washington DC | SaaS | 8 | 1800000 | 2000000 | 2016 | A | Roland Tavurner | GPS Services |
| Allniter | New Delhi | Education | 250 | 1000000 | 15000000 | 2009 | A | Rohan Taneja | Homework/Essay Help |
| Gnome Reverser | Denver | Mobile | 10 | 1500000 | 10000000 | 2014 | A | Bruce Dzeda | Location Service to Find Your Keys/Wallet |
| ScoopAble | Denver | Social | 14 | 1000000 | 7600000 | 2016 | Seed | Zoe Gfell | Yogurt Finder |
| YoBert | New York | Education | 31 | 800000 | 40000000 | 2016 | Acquired | Ernie Dzeda | Seasme Street Themed Educational App |
| Trufflez | New York | Mobile | 23 | 1000000 | 12000000 | 2014 | Seed |  | AI Suggests Recipes Based on Photos |
| Unfade | New York | Fashion | 4 | 400000 | 5600000 | 2016 | A | Peggy Grimes | Everything Hair |
| OrangeYouLonely | Minneapolis | Social | 20 | 1200000 | 14000000 | 2012 | Seed | Gus Dawson | Dating App for Farmers |
| WellDeserved | San Francisco | E-commerce | 4 |  |  | 2015 | Seed | Kasima Tharnpipitchai | The Premiere Marketplace for Privilege |
| **COUNT(\*)** |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |
| **SUM(valuation)** |  |  |  |  |  |  |  |  |  |
| 974455790000 |  |  |  |  |  |  |  |  |  |
| **MAX(raised)** |  |  |  |  |  |  |  |  |  |
| 1800000 |  |  |  |  |  |  |  |  |  |
| **MIN(founded)** |  |  |  |  |  |  |  |  |  |
| 1994 |  |  |  |  |  |  |  |  |  |
| **AVG(valuation)** |  |  |  |  |  |  |  |  |  |
| 15974685081.9672 |  |  |  |  |  |  |  |  |  |
| **category** | **AVG(valuation)** |  |  |  |  |  |  |  |  |
|  | 4290000.0 |  |  |  |  |  |  |  |  |
| Algorithms | 7600000.0 |  |  |  |  |  |  |  |  |
| Augmented Reality | 8000000000.0 |  |  |  |  |  |  |  |  |
| Big Data Analytics | 15000000.0 |  |  |  |  |  |  |  |  |
| Cloud Computing | 95000000.0 |  |  |  |  |  |  |  |  |
| Customer Service | 640000000.0 |  |  |  |  |  |  |  |  |
| Data Analytics |  |  |  |  |  |  |  |  |  |
| E-commerce | 60250000.0 |  |  |  |  |  |  |  |  |
| Education | 2023800000.0 |  |  |  |  |  |  |  |  |
| Enterprise | 38508333333.3333 |  |  |  |  |  |  |  |  |
| Fashion | 3200000.0 |  |  |  |  |  |  |  |  |
| Fitness | 67500000.0 |  |  |  |  |  |  |  |  |
| Gambling | 322000000.0 |  |  |  |  |  |  |  |  |
| Games | 1600000.0 |  |  |  |  |  |  |  |  |
| Health Care | 380490000000.0 |  |  |  |  |  |  |  |  |
| Logistics | 70000000.0 |  |  |  |  |  |  |  |  |
| Mobile | 9600000.0 |  |  |  |  |  |  |  |  |
| Nanotechnology | 150000000.0 |  |  |  |  |  |  |  |  |
| Real Estate | 20000000000.0 |  |  |  |  |  |  |  |  |
| SaaS | 401000000.0 |  |  |  |  |  |  |  |  |
| Security | 6333333333.33333 |  |  |  |  |  |  |  |  |
| Social | 82425833.3333333 |  |  |  |  |  |  |  |  |
| Technology | 3100000000.0 |  |  |  |  |  |  |  |  |
| Travel | 12501250000.0 |  |  |  |  |  |  |  |  |
| Venture Capital | 3000000000.0 |  |  |  |  |  |  |  |  |
| Video Streaming |  |  |  |  |  |  |  |  |  |
| Virtual Reality | 2150000.0 |  |  |  |  |  |  |  |  |
| **category** | **ROUND(AVG(valuation), 2)** |  |  |  |  |  |  |  |  |
|  | 4290000.0 |  |  |  |  |  |  |  |  |
| Algorithms | 7600000.0 |  |  |  |  |  |  |  |  |
| Augmented Reality | 8000000000.0 |  |  |  |  |  |  |  |  |
| Big Data Analytics | 15000000.0 |  |  |  |  |  |  |  |  |
| Cloud Computing | 95000000.0 |  |  |  |  |  |  |  |  |
| Customer Service | 640000000.0 |  |  |  |  |  |  |  |  |
| Data Analytics |  |  |  |  |  |  |  |  |  |
| E-commerce | 60250000.0 |  |  |  |  |  |  |  |  |
| Education | 2023800000.0 |  |  |  |  |  |  |  |  |
| Enterprise | 38508333333.33 |  |  |  |  |  |  |  |  |
| Fashion | 3200000.0 |  |  |  |  |  |  |  |  |
| Fitness | 67500000.0 |  |  |  |  |  |  |  |  |
| Gambling | 322000000.0 |  |  |  |  |  |  |  |  |
| Games | 1600000.0 |  |  |  |  |  |  |  |  |
| Health Care | 380490000000.0 |  |  |  |  |  |  |  |  |
| Logistics | 70000000.0 |  |  |  |  |  |  |  |  |
| Mobile | 9600000.0 |  |  |  |  |  |  |  |  |
| Nanotechnology | 150000000.0 |  |  |  |  |  |  |  |  |
| Real Estate | 20000000000.0 |  |  |  |  |  |  |  |  |
| SaaS | 401000000.0 |  |  |  |  |  |  |  |  |
| Security | 6333333333.33 |  |  |  |  |  |  |  |  |
| Social | 82425833.33 |  |  |  |  |  |  |  |  |
| Technology | 3100000000.0 |  |  |  |  |  |  |  |  |
| Travel | 12501250000.0 |  |  |  |  |  |  |  |  |
| Venture Capital | 3000000000.0 |  |  |  |  |  |  |  |  |
| Video Streaming |  |  |  |  |  |  |  |  |  |
| Virtual Reality | 2150000.0 |  |  |  |  |  |  |  |  |
| **category** | **ROUND(AVG(valuation), 2)** |  |  |  |  |  |  |  |  |
| Health Care | 380490000000.0 |  |  |  |  |  |  |  |  |
| Enterprise | 38508333333.33 |  |  |  |  |  |  |  |  |
| Real Estate | 20000000000.0 |  |  |  |  |  |  |  |  |
| Travel | 12501250000.0 |  |  |  |  |  |  |  |  |
| Augmented Reality | 8000000000.0 |  |  |  |  |  |  |  |  |
| Security | 6333333333.33 |  |  |  |  |  |  |  |  |
| Technology | 3100000000.0 |  |  |  |  |  |  |  |  |
| Venture Capital | 3000000000.0 |  |  |  |  |  |  |  |  |
| Education | 2023800000.0 |  |  |  |  |  |  |  |  |
| Customer Service | 640000000.0 |  |  |  |  |  |  |  |  |
| SaaS | 401000000.0 |  |  |  |  |  |  |  |  |
| Gambling | 322000000.0 |  |  |  |  |  |  |  |  |
| Nanotechnology | 150000000.0 |  |  |  |  |  |  |  |  |
| Cloud Computing | 95000000.0 |  |  |  |  |  |  |  |  |
| Social | 82425833.33 |  |  |  |  |  |  |  |  |
| Logistics | 70000000.0 |  |  |  |  |  |  |  |  |
| Fitness | 67500000.0 |  |  |  |  |  |  |  |  |
| E-commerce | 60250000.0 |  |  |  |  |  |  |  |  |
| Big Data Analytics | 15000000.0 |  |  |  |  |  |  |  |  |
| Mobile | 9600000.0 |  |  |  |  |  |  |  |  |
| Algorithms | 7600000.0 |  |  |  |  |  |  |  |  |
|  | 4290000.0 |  |  |  |  |  |  |  |  |
| Fashion | 3200000.0 |  |  |  |  |  |  |  |  |
| Virtual Reality | 2150000.0 |  |  |  |  |  |  |  |  |
| Games | 1600000.0 |  |  |  |  |  |  |  |  |
| Video Streaming |  |  |  |  |  |  |  |  |  |
| Data Analytics |  |  |  |  |  |  |  |  |  |
| **category** | **COUNT(\*)** |  |  |  |  |  |  |  |  |
|  | 3 |  |  |  |  |  |  |  |  |
| Algorithms | 1 |  |  |  |  |  |  |  |  |
| Augmented Reality | 1 |  |  |  |  |  |  |  |  |
| Big Data Analytics | 1 |  |  |  |  |  |  |  |  |
| Cloud Computing | 2 |  |  |  |  |  |  |  |  |
| Customer Service | 1 |  |  |  |  |  |  |  |  |
| Data Analytics | 1 |  |  |  |  |  |  |  |  |
| E-commerce | 3 |  |  |  |  |  |  |  |  |
| Education | 5 |  |  |  |  |  |  |  |  |
| Enterprise | 3 |  |  |  |  |  |  |  |  |
| Fashion | 2 |  |  |  |  |  |  |  |  |
| Fitness | 3 |  |  |  |  |  |  |  |  |
| Gambling | 1 |  |  |  |  |  |  |  |  |
| Games | 2 |  |  |  |  |  |  |  |  |
| Health Care | 2 |  |  |  |  |  |  |  |  |
| Logistics | 1 |  |  |  |  |  |  |  |  |
| Mobile | 10 |  |  |  |  |  |  |  |  |
| Nanotechnology | 1 |  |  |  |  |  |  |  |  |
| Real Estate | 1 |  |  |  |  |  |  |  |  |
| SaaS | 2 |  |  |  |  |  |  |  |  |
| Security | 3 |  |  |  |  |  |  |  |  |
| Social | 12 |  |  |  |  |  |  |  |  |
| Technology | 3 |  |  |  |  |  |  |  |  |
| Travel | 2 |  |  |  |  |  |  |  |  |
| Venture Capital | 1 |  |  |  |  |  |  |  |  |
| Video Streaming | 1 |  |  |  |  |  |  |  |  |
| Virtual Reality | 2 |  |  |  |  |  |  |  |  |
| **category** | **COUNT(\*)** |  |  |  |  |  |  |  |  |
| Education | 5 |  |  |  |  |  |  |  |  |
| Mobile | 10 |  |  |  |  |  |  |  |  |
| Social | 12 |  |  |  |  |  |  |  |  |
| **location** | **AVG(employees)** |  |  |  |  |  |  |  |  |
| Atlanta | 3.0 |  |  |  |  |  |  |  |  |
| Boulder | 3.0 |  |  |  |  |  |  |  |  |
| Brooklyn | 502.666666666667 |  |  |  |  |  |  |  |  |
| Chicago | 12.0 |  |  |  |  |  |  |  |  |
| Columbus | 2.0 |  |  |  |  |  |  |  |  |
| Denver | 12.0 |  |  |  |  |  |  |  |  |
| Fort Lauderdale | 500.0 |  |  |  |  |  |  |  |  |
| Irvine | 2.0 |  |  |  |  |  |  |  |  |
| Long Island | 5.0 |  |  |  |  |  |  |  |  |
| Los Angeles | 6.83333333333333 |  |  |  |  |  |  |  |  |
| Minneapolis | 20.0 |  |  |  |  |  |  |  |  |
| New Delhi | 250.0 |  |  |  |  |  |  |  |  |
| New York | 702.75 |  |  |  |  |  |  |  |  |
| Omaha | 65.0 |  |  |  |  |  |  |  |  |
| Palo Alto | 125.833333333333 |  |  |  |  |  |  |  |  |
| Paris | 30.0 |  |  |  |  |  |  |  |  |
| San Francisco | 1920.4 |  |  |  |  |  |  |  |  |
| Savannah | 6.0 |  |  |  |  |  |  |  |  |
| Scranton | 6.0 |  |  |  |  |  |  |  |  |
| Seattle | 9.66666666666667 |  |  |  |  |  |  |  |  |
| Silicon Valley | 1804.6 |  |  |  |  |  |  |  |  |
| Virginia Beach | 15.0 |  |  |  |  |  |  |  |  |
| Washington DC | 8.0 |  |  |  |  |  |  |  |  |
| **location** | **AVG(employees)** |  |  |  |  |  |  |  |  |
| Brooklyn | 502.666666666667 |  |  |  |  |  |  |  |  |
| New York | 702.75 |  |  |  |  |  |  |  |  |
| San Francisco | 1920.4 |  |  |  |  |  |  |  |  |
| Silicon Valley | 1804.6 |  |  |  |  |  |  |  |  |

**Database Schema**

**startups**

|  |  |
| --- | --- |
| **name** | **type** |
| name | TEXT |
| location | TEXT |
| category | TEXT |
| employees | INTEGER |
| raised | INTEGER |
| valuation | INTEGER |
| founded | INTEGER |
| stage | TEXT |
| ceo | TEXT |
| info | TEXT |

|  |
| --- |
| **Rows: 70** |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AGGREGATE FUNCTIONS CHALLENGE**

**Codeflix Introduction**

<1 min

Welcome to Code Challenge: Aggregate Functions!

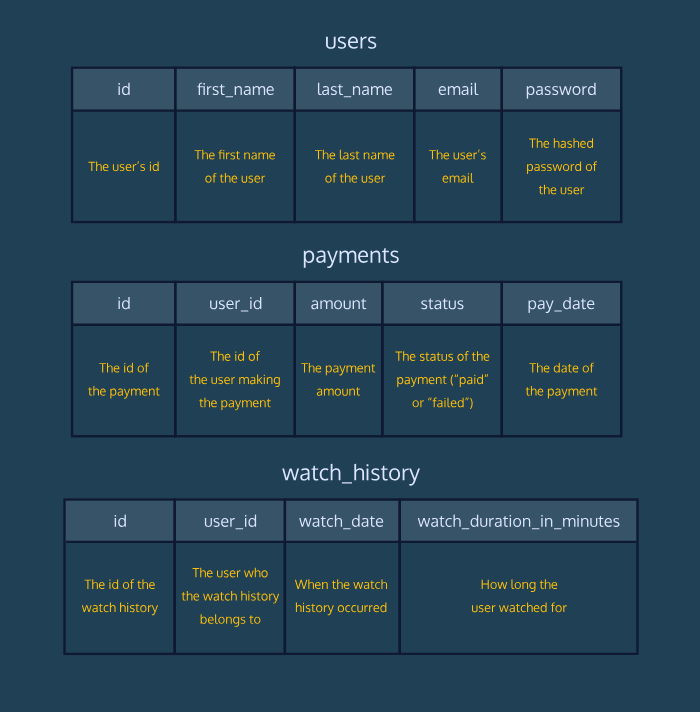
A streaming video company, Codeflix, needs your help analyzing their user data. We’ve imported a portion of their dataset into the SQL workspace for this lesson.

The database contains 3 tables:

* users contains the basic account details for each user
* payments contains payment details for a 3 month period
* watch\_history contains watch events for the users

**Instructions**

When you’re ready, let’s begin!



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 1**

3 min

The users table has the following columns:

* id
* first\_name
* last\_name
* email
* password

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Use COUNT() and a LIKE operator to determine the number of users that have an email ending in ‘.com’.

Hint

As an example,

SELECT \*   
FROM users   
WHERE first\_name LIKE '%ny';

Copy to Clipboard

will give you all the users whose names end in ny.

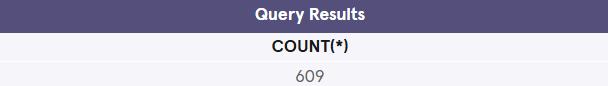
Using COUNT(\*) instead of \* will return the number of rows that matches the criteria.

**test.sqlite**

SELECT COUNT(\*)

FROM users

WHERE email LIKE '%.com';



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 2**

5 min

The users table has the following columns:

* id
* first\_name
* last\_name
* email
* password

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

What are the most popular first names on Codeflix?

Use COUNT(), GROUP BY, and ORDER BY to create a list of first names and occurrences within the users table.

Order the data so that the most popular names are displayed first.

Hint

When you select the count of users, make sure to rename it:

SELECT first\_name, COUNT(\*) AS 'count'  
FROM users  
GROUP BY first\_name  
ORDER BY 2 DESC;

Copy to Clipboard

This way, after you GROUP BY first\_name, you can use ORDER BY to sort the data by the count variable.

Also, because we want to display the most popular names first, DESC is needed!

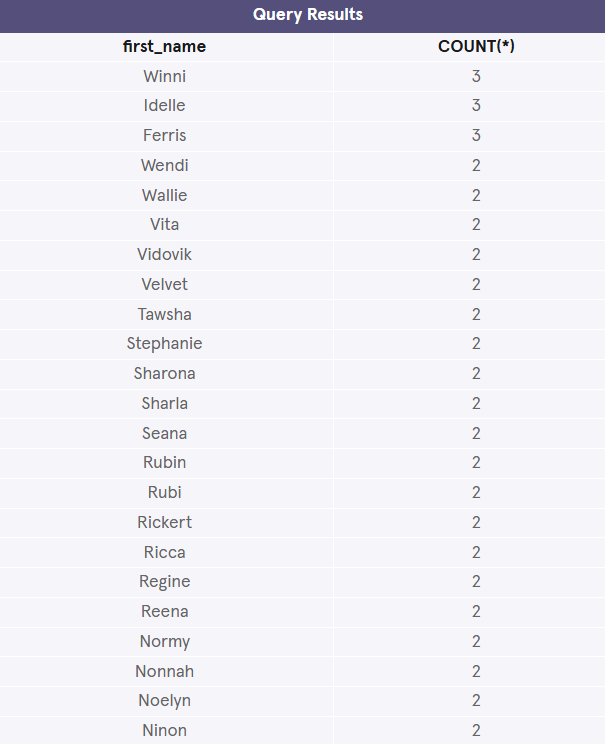
**test.sqlite**

SELECT first\_name, COUNT(\*)

FROM users

GROUP BY first\_name

ORDER BY COUNT(\*) DESC;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 3**

9 min

The watch\_history table has the following columns:

* id
* user\_id
* watch\_date
* watch\_duration\_in\_minutes

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

The UX Research team wants to see a distribution of watch durations. They want the result to contain:

* + duration, which is the watch event duration, rounded to the closest minute
  + count, the number of watch events falling into this “bucket”

Your result should look like:

| **duration** | **count** |
| --- | --- |
| 1.0 | 9 |
| 2.0 | 21 |
| 3.0 | 19 |
| … | … |

Use COUNT(), GROUP BY, and ORDER BY to create this result and order this data by increasing duration.

Hint

Use ROUND() to create a column which is the watch duration rounded to the nearest minute. And return a second column that has the number of watch events falling into each duration:

SELECT ROUND(watch\_duration\_in\_minutes) AS 'duration',  
   COUNT(\*) AS 'count'  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ;

Copy to Clipboard

You can then GROUP BY and ORDER BY this variable.

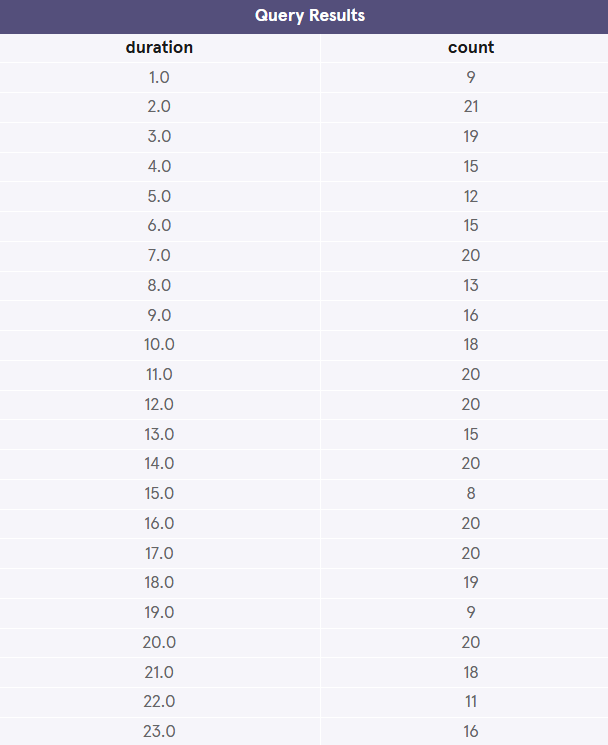
**test.sqlite**

SELECT ROUND(watch\_duration\_in\_minutes) AS 'duration', COUNT(\*) AS 'count'

FROM watch\_history

GROUP BY duration

ORDER BY duration;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 4**

7 min

The payments table has the following columns:

* id
* user\_id
* amount
* status
* pay\_date

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Find all the users that have successfully made a payment to Codeflix and find their total amount paid.

Sort them by their total payments (from high to low).

Use SUM(), WHERE, GROUP BY, and ORDER BY.

Hint

Group the users and calculate their total payments by using SUM().

Filter the successful payments by using WHERE with the condition status = 'paid'.

Sort the total payments in a descending order using ORDER BY and DESC.

SELECT user\_id, SUM(amount)  
FROM payments  
WHERE status = 'paid'  
GROUP BY user\_id  
ORDER BY SUM(amount) DESC;

Copy to Clipboard

You can also rename the SUM(amount) column:

SELECT user\_id, SUM(amount) AS 'total'  
FROM payments  
WHERE status = 'paid'  
GROUP BY user\_id  
ORDER BY total DESC;

Copy to Clipboard

And add column reference numbers:

SELECT user\_id, SUM(amount) AS 'total'  
FROM payments  
WHERE status = 'paid'  
GROUP BY 1  
ORDER BY 2 DESC;

**test.sqlite**

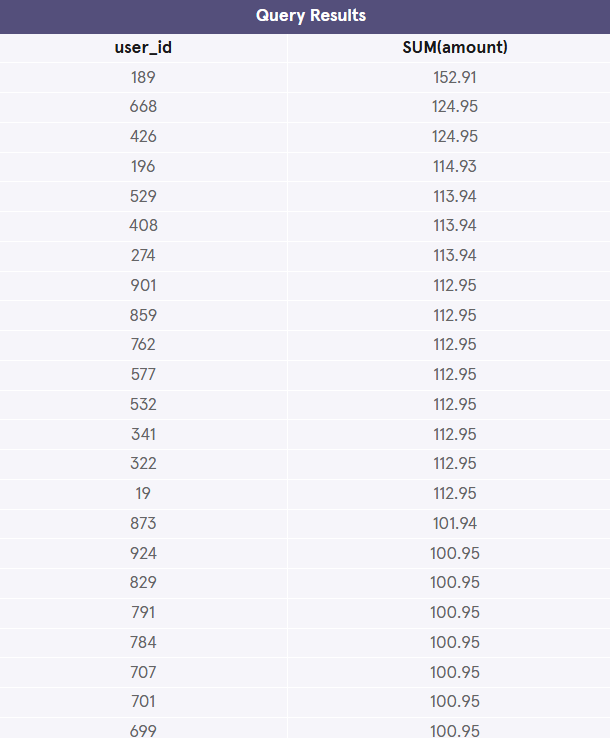
SELECT user\_id, SUM(amount)

FROM payments

WHERE status = 'paid'

GROUP BY user\_id

ORDER BY SUM(amount) DESC;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 5**

6 min

The watch\_history table has the following columns:

* id
* user\_id
* watch\_date
* watch\_duration\_in\_minutes

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Generate a table of user ids and total watch duration for users who watched more than 400 minutes of content.

Use SUM(), GROUP BY, and HAVING to achieve this.

Hint

Use SUM() and GROUP BY to calculate a total amount for each user\_id.

The HAVING allows us to filter the result to users that have a total amount greater than 400:

SELECT user\_id,   
   SUM(watch\_duration\_in\_minutes)  
FROM watch\_history  
GROUP BY user\_id  
HAVING SUM(watch\_duration\_in\_minutes) > 400;

Copy to Clipboard

If you give the total amount an alias, you can use it in HAVING:

SELECT user\_id,   
   SUM(watch\_duration\_in\_minutes) AS 'total\_duration'  
FROM watch\_history  
GROUP BY 1  
HAVING total\_duration > 400;

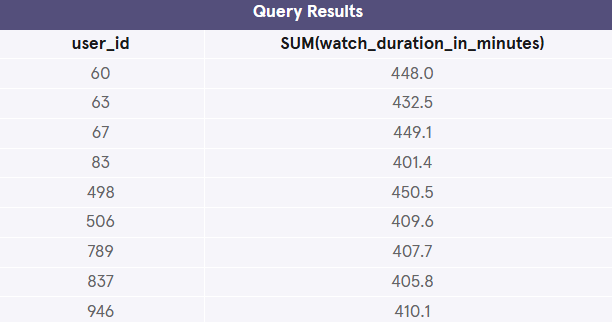
**test.sqlite**

SELECT user\_id, SUM(watch\_duration\_in\_minutes)

from watch\_history

GROUP BY user\_id

HAVING SUM(watch\_duration\_in\_minutes) > 400;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 6**

2 min

The watch\_history table has the following columns:

* id
* user\_id
* watch\_date
* watch\_duration\_in\_minutes

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

To the nearest minute, how many minutes of content were streamed on Codeflix?

Hint

Use the SUM() and ROUND() functions with the watch\_history table.

You can treat SUM(watch\_duration\_in\_minutes) just like any other value and place it inside the ROUND() function like so:

ROUND(SUM(watch\_duration\_in\_minutes), 0)

Copy to Clipboard

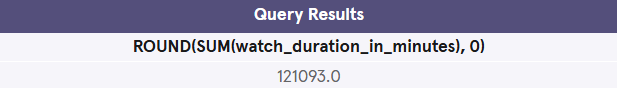
Here, SUM(watch\_duration\_in\_minutes) is the 1st argument and 0 is the 2nd argument because we want to round it to zero decimal places:

SELECT ROUND(SUM(watch\_duration\_in\_minutes), 0)   
FROM watch\_history;

**test.sqlite**

SELECT ROUND(SUM(watch\_duration\_in\_minutes), 0)

FROM watch\_history;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 7**

4 min

The payments table has the following columns:

* id
* user\_id
* amount
* status
* pay\_date

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Which days in this period did Codeflix collect the most money?

Your result should have two columns, pay\_date and total amount, sorted by the latter descending.

This should only include successful payments (status = 'paid').

Use SUM(), GROUP BY, and ORDER BY.

Hint

Use SUM() to calculate the total amount paid each day and rename it to ‘total’ using AS.

Be sure to GROUP BY the day and ORDER BY the total in a decreasing order.

SELECT pay\_date, SUM(amount)  
FROM payments  
WHERE status = 'paid'  
GROUP BY pay\_date  
ORDER BY SUM(amount) DESC;

Copy to Clipboard

You can also rename pay\_date → day and SUM(amount) → total using AS:

SELECT pay\_date AS 'day',  
   SUM(amount) AS 'total'  
FROM payments  
WHERE status = 'paid'  
GROUP BY day  
ORDER BY total DESC;

**test.sqlite**

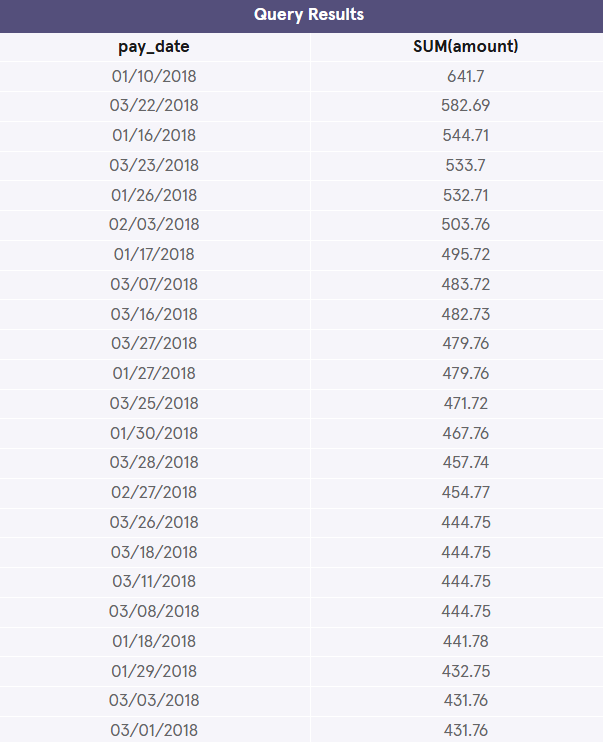
SELECT pay\_date, SUM(amount)

FROM payments

WHERE status = 'paid'

GROUP BY pay\_date

ORDER BY SUM(amount) DESC;



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 8**

2 min

The payments table has the following columns:

* id
* user\_id
* amount
* status
* pay\_date

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

When users successfully pay Codeflix (status = 'paid'), what is the average payment amount?

Hint

Use the AVG() function after selecting the correct rows from payments:

SELECT AVG(\_\_\_\_\_\_)  
FROM payments  
WHERE status = 'paid';

Copy to Clipboard

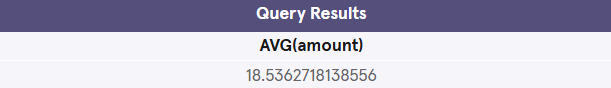
You don’t need GROUP BY, ORDER BY, and HAVING for this query!

**test.sqlite**

SELECT AVG(amount)

FROM payments

WHERE status = 'paid';



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code Challenge 9**

3 min

The watch\_history table has the following columns:

* id
* user\_id
* watch\_date
* watch\_duration\_in\_minutes

Click [here](https://content.codecademy.com/courses/learn-sql-aggregates-code-challenge/user-payments-history.png) for the table diagram.

**Instructions**

1. Checkpoint 1 Passed

**1.**

Of all the events in the watch\_history table, what is the duration of the longest individual watch event? What is the duration of the shortest?

Use one query and rename the results to max and min.

Hint

Use the MAX() function to find the longest duration event and MIN() to find the shortest.

SELECT MAX(\_\_\_\_\_), MIN(\_\_\_\_\_)  
FROM \_\_\_\_\_\_ ;

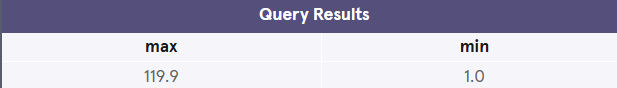
Copy to Clipboard

You can rename the results using AS.

**test.sqlite**

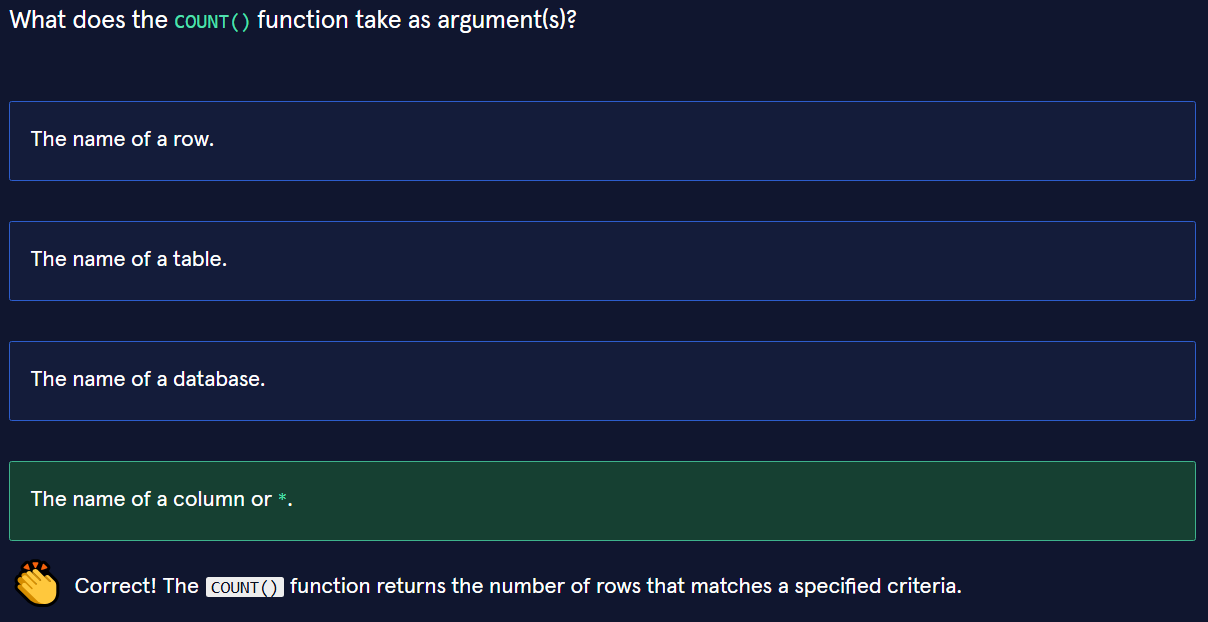
SELECT MAX(watch\_duration\_in\_minutes) AS max, MIN(watch\_duration\_in\_minutes) AS min

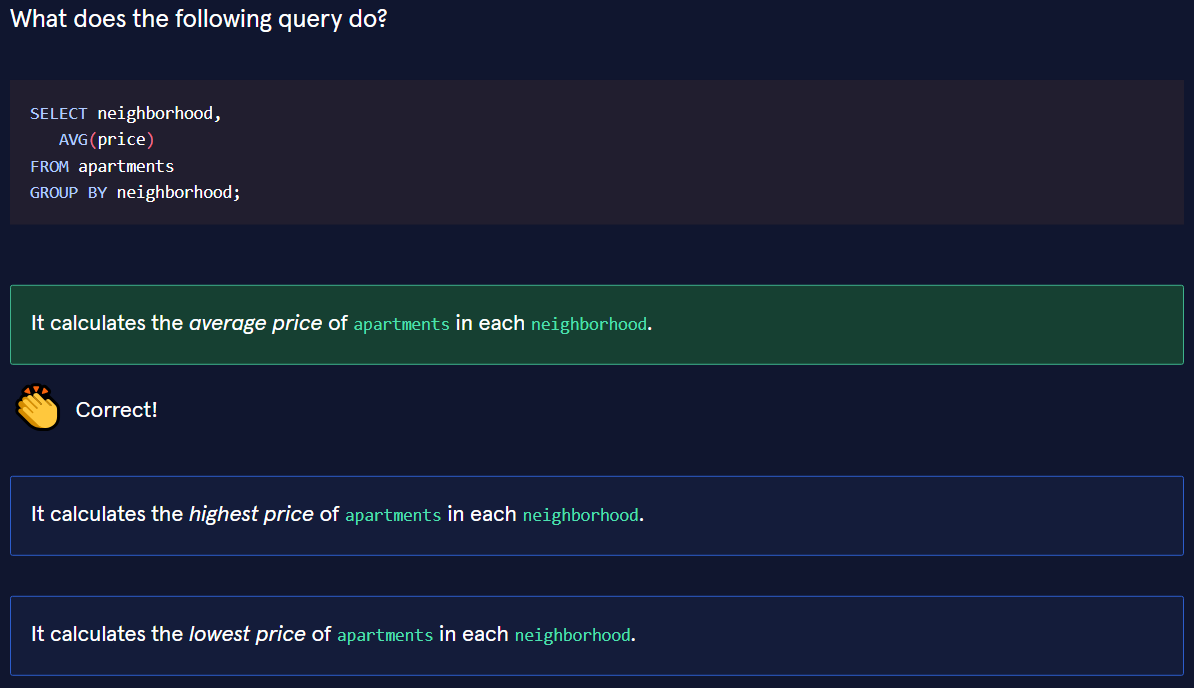
FROM watch\_history;

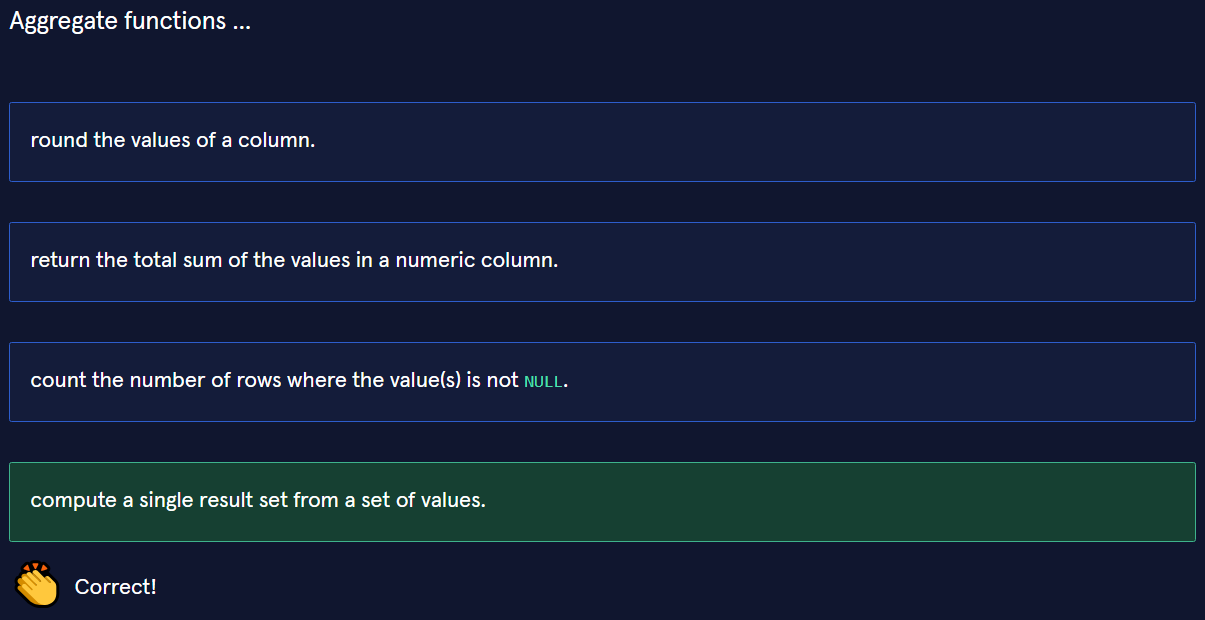


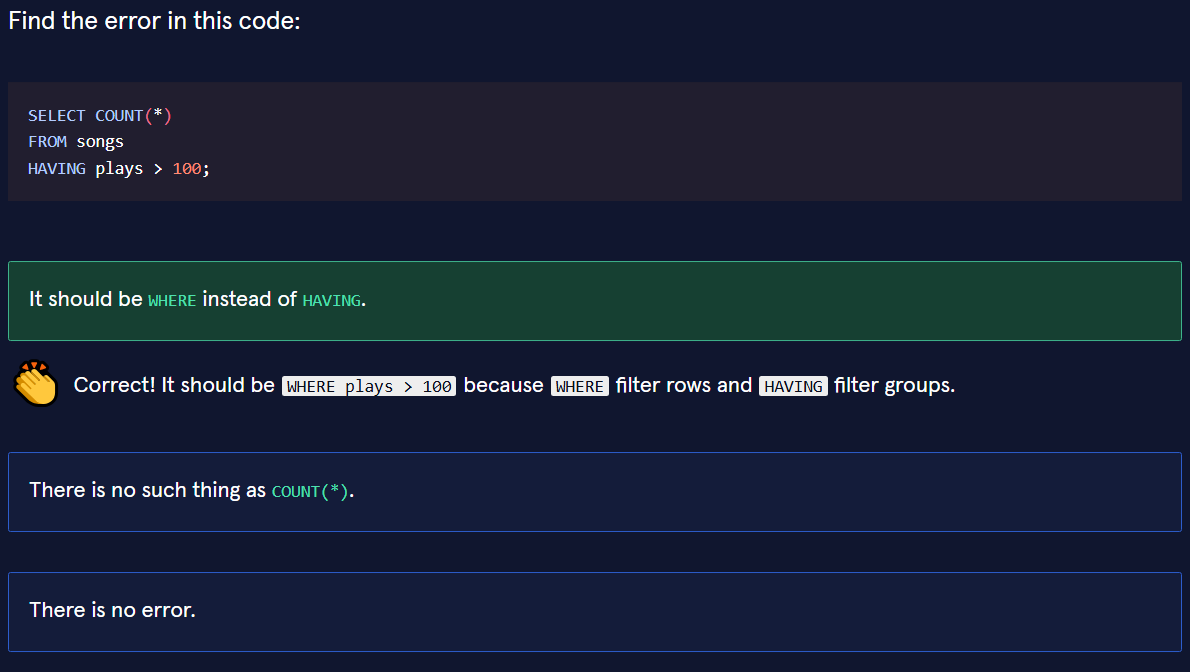
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

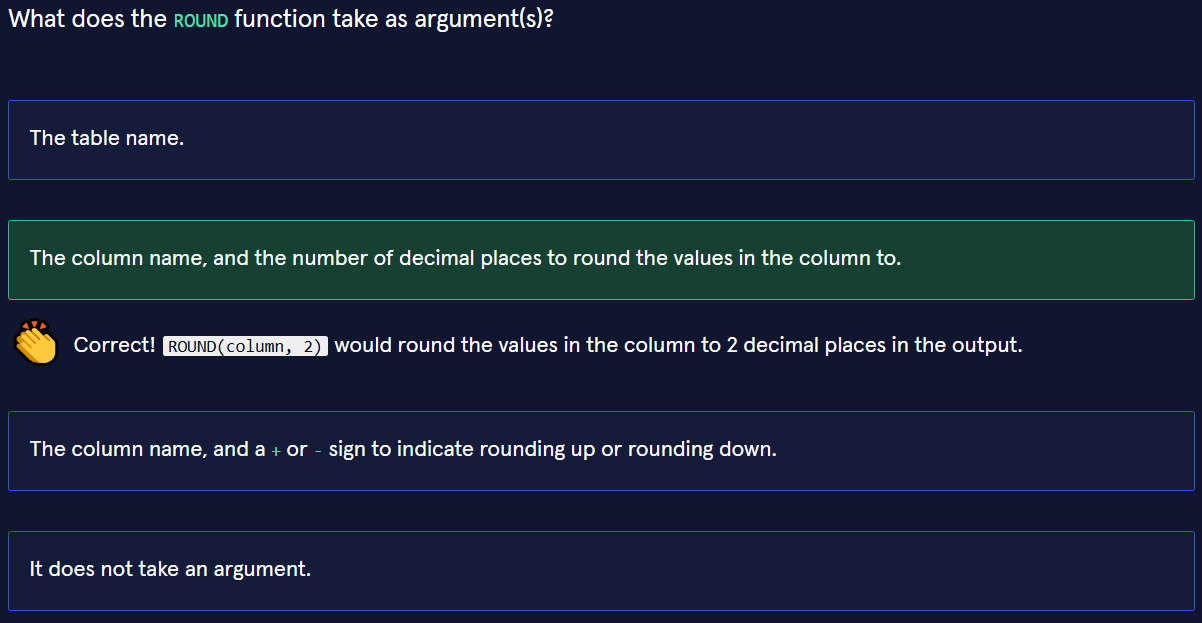
**QUIZ**

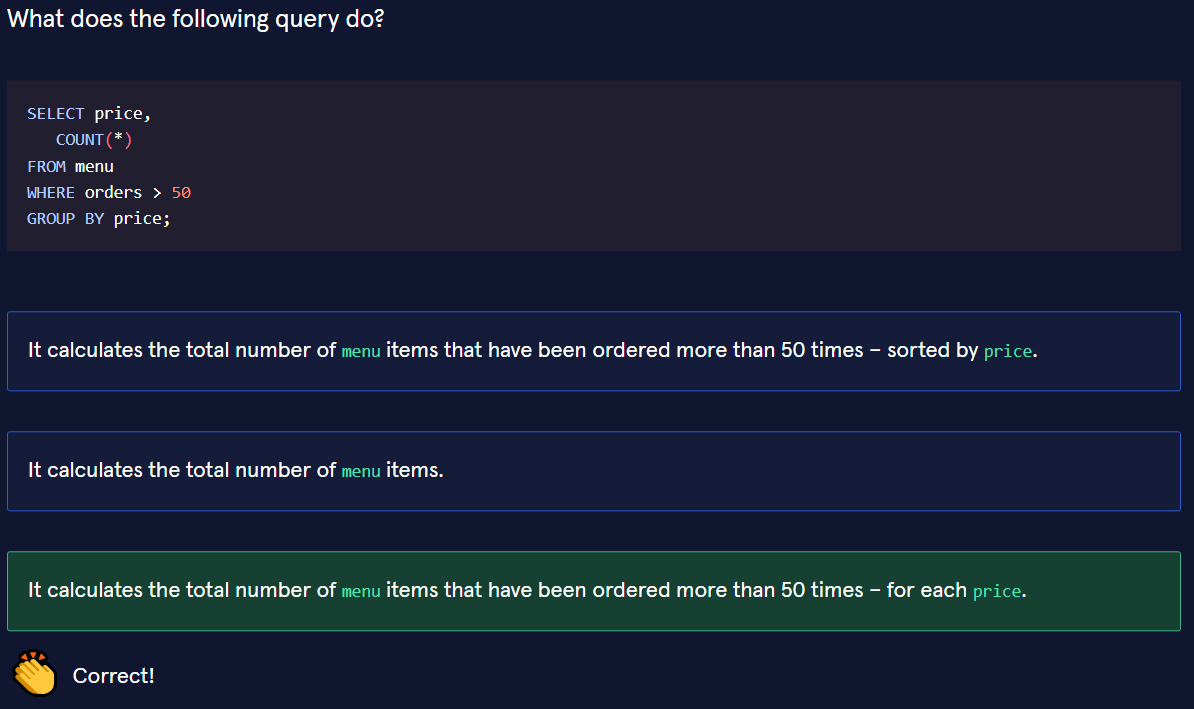
****

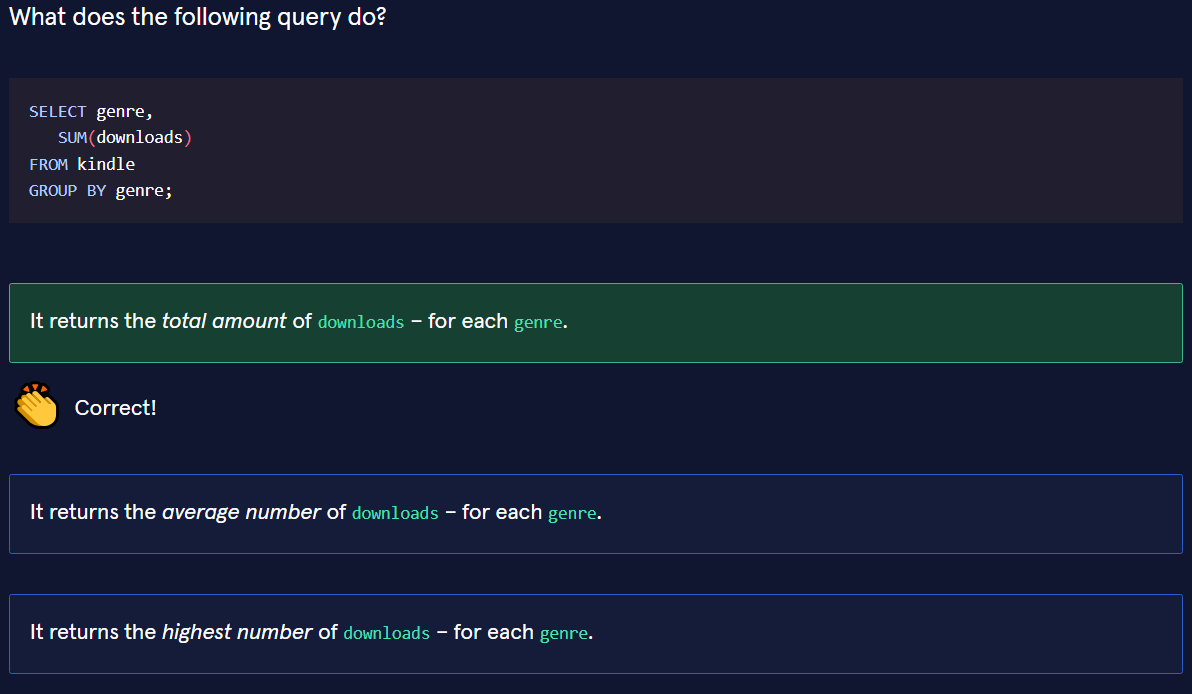
****

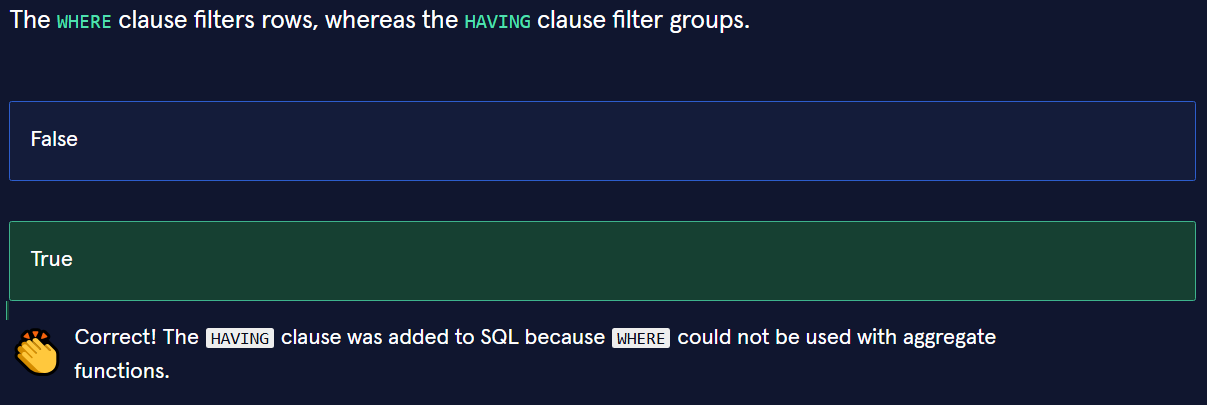
****

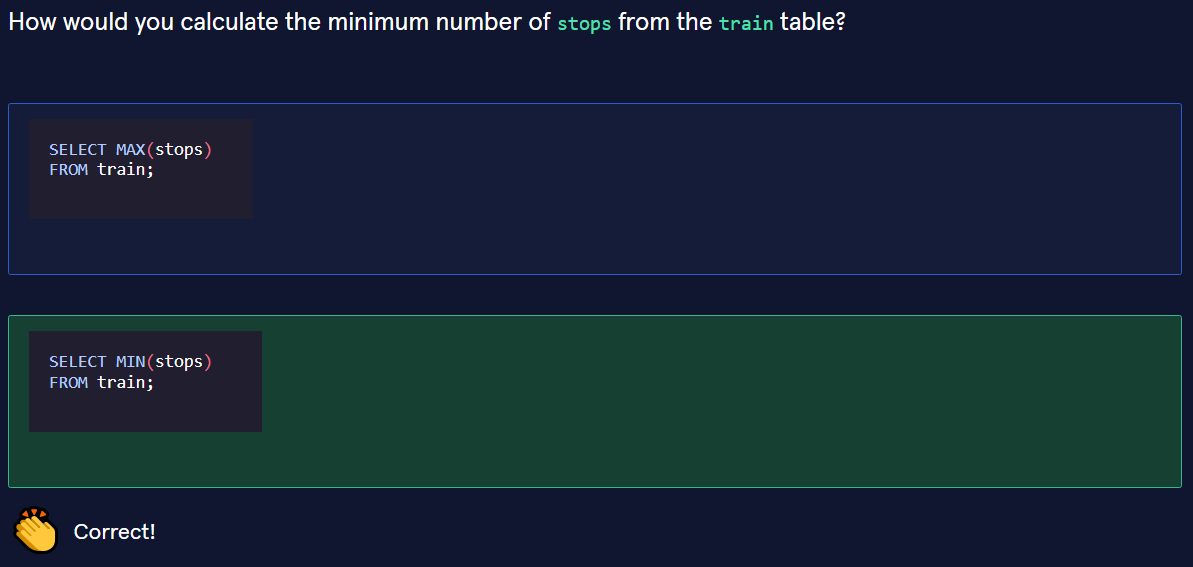
****

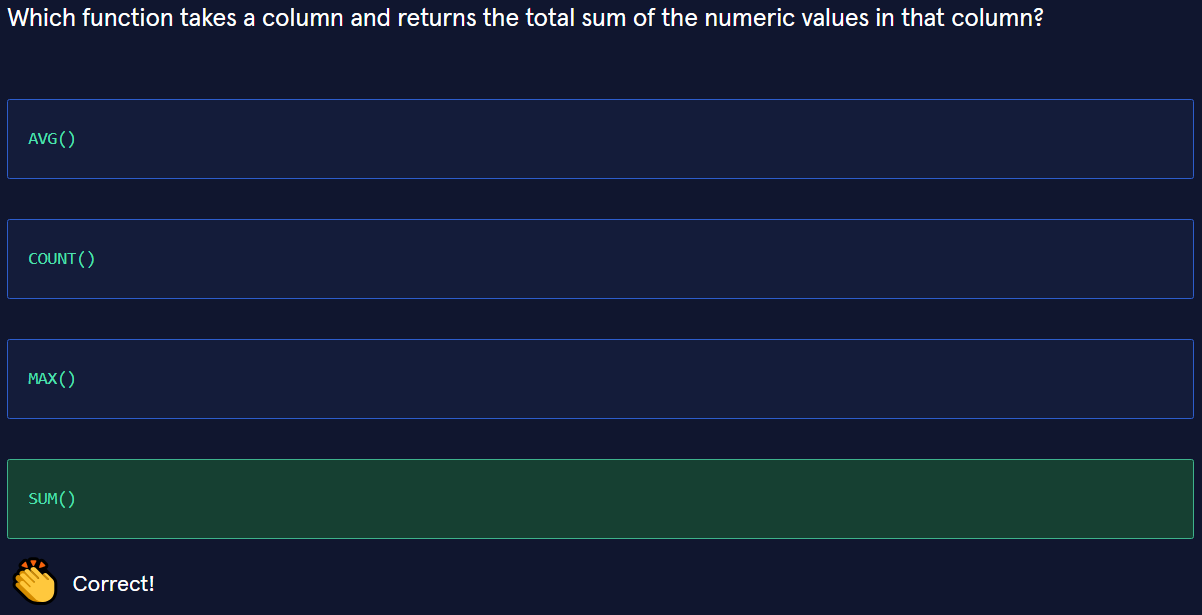
****

****

****

****

****

****