**AGGREGATE FUNCTIONS**

**Introduction**

We’ve learned how to write queries to retrieve information from the database. Now, we are going to learn how to perform calculations using SQL.

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:

* [COUNT()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/count?page_ref=catalog): count the number of rows
* [SUM()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/sum?page_ref=catalog): the sum of the values in a column
* [MAX()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/max?page_ref=catalog)/[MIN()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/min?page_ref=catalog): the largest/smallest value
* [AVG()](https://www.codecademy.com/resources/docs/sql/aggregate-functions/avg?page_ref=catalog): the average of the values in a column
* [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.**

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;

What are the column names?

Checkpoint 2 Passed

Hint

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

**aggregate-functions.sqlite**

SELECT \*

FROM fake\_apps;



**Count**

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

[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;

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

**Instructions**

**1.**

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

In the code editor, run:

SELECT COUNT(\*)   
FROM fake\_apps;

Checkpoint 2 Passed

Hint

There are 200 apps.

Common errors:

* Missing parenthesis.
* Missing ;.

**2.**

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

Checkpoint 3 Passed

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;

* 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**

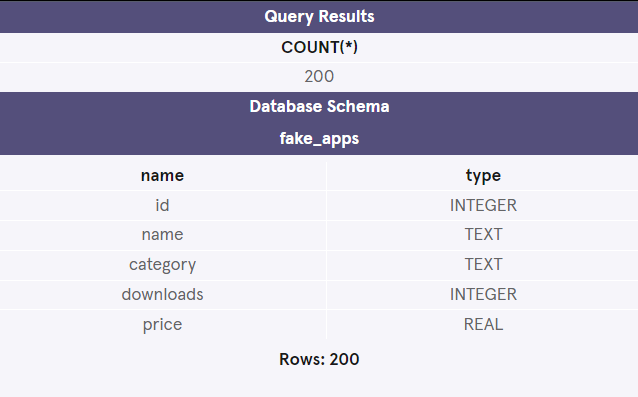
/\*SELECT COUNT(\*)

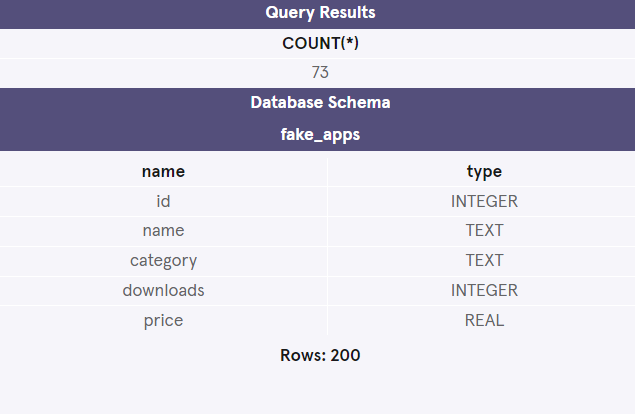
FROM fake\_apps\*/

SELECT COUNT(\*)

FROM fake\_apps

WHERE price = 0.0;

****

****

**Sum**

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

[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;

This adds all values in the downloads column.

**Instructions**

**1.**

Let’s find out the answer!

In the code editor, type:

SELECT SUM(downloads)  
FROM fake\_apps;

Checkpoint 2 Passed

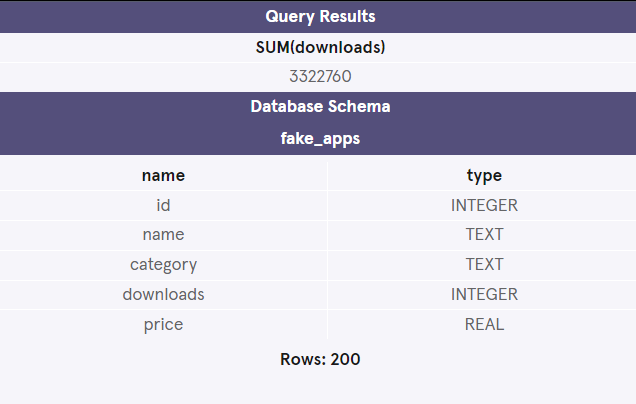
Hint

There are 3,322,760 total downloads.

**sum.sqlite**

SELECT SUM(downloads)

FROM fake\_apps;

****

**Max / 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;

The most popular app has 31,090 downloads!

[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.

[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.**

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

In the code editor, type:

SELECT MIN(downloads)  
FROM fake\_apps;

Checkpoint 2 Passed

Hint

1,387 downloads.

**2.**

Delete the previous query.

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

Checkpoint 3 Passed

Hint

SELECT MAX(price)  
FROM fake\_apps;

$14.99 is the price of the most expensive app.

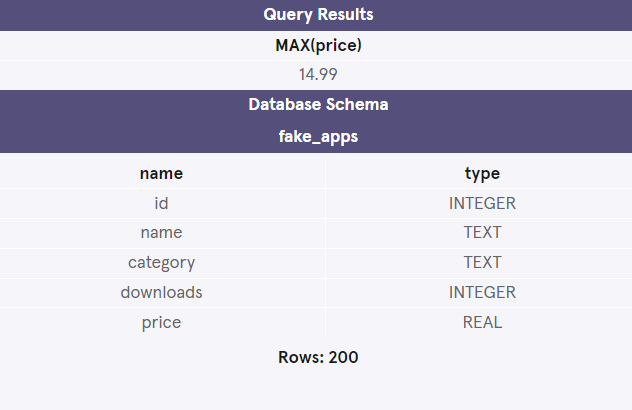
**max.sqlite**

/\*SELECT MIN(downloads)

FROM fake\_apps;\*/

SELECT MAX(price)

FROM fake\_apps;

****

**Average**

SQL uses the [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;

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

**Instructions**

**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;

Checkpoint 2 Passed

Hint

16,613.8 average downloads.

**2.**

Remove the previous query.

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

Checkpoint 3 Passed

Hint

Which column should go inside the parenthesis?

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

The average price is $2.02365.

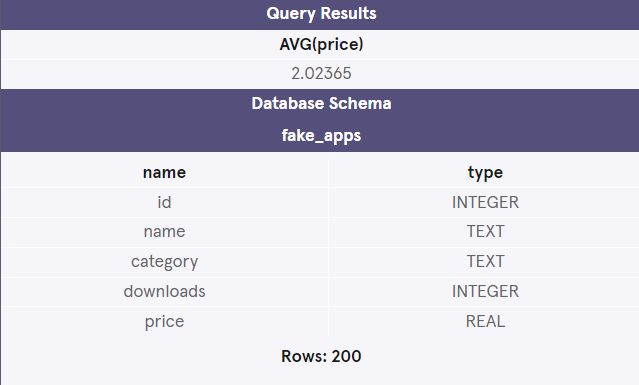
**avg.sqlite**

/\*SELECT AVG(downloads)

FROM fake\_apps;\*/

SELECT AVG(price)

FROM fake\_apps;



**Round**

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.

[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;

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.**

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

In the code editor, type:

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

Checkpoint 2 Passed

Hint

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

**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;

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

This is a tricky one!

Checkpoint 3 Passed

Hint

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

ROUND(AVG(price), 2)

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;

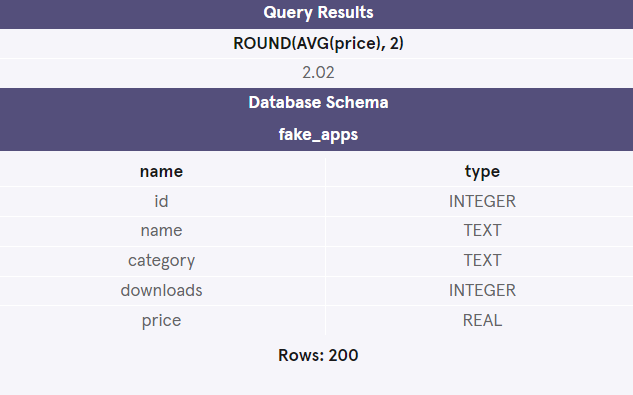
**round.sqlite**

/\*SELECT name, ROUND(price, 0)

FROM fake\_apps;\*/

SELECT ROUND(AVG(price), 2)

FROM fake\_apps;

****

**Group By I**

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;

and so on.

Luckily, there’s a better way!

We can use [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;

GROUP BY is a clause in SQL that is used with aggregate functions. It is used in collaboration with the SELECT statement to arrange identical data into *groups*.

The GROUP BY statement comes after any [WHERE](https://www.codecademy.com/resources/docs/sql/commands/where?page_ref=catalog) statements, but before [ORDER BY](https://www.codecademy.com/resources/docs/sql/commands/order-by?page_ref=catalog) or [LIMIT](https://www.codecademy.com/resources/docs/sql/commands/limit?page_ref=catalog).

**Instructions**

**1.**

In the code editor, type:

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

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

What do you expect the result to be?

Checkpoint 2 Passed

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.

**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.

Checkpoint 3 Passed

Hint

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

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).

Checkpoint 4 Passed

Hint

First, select the two columns we want:

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

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

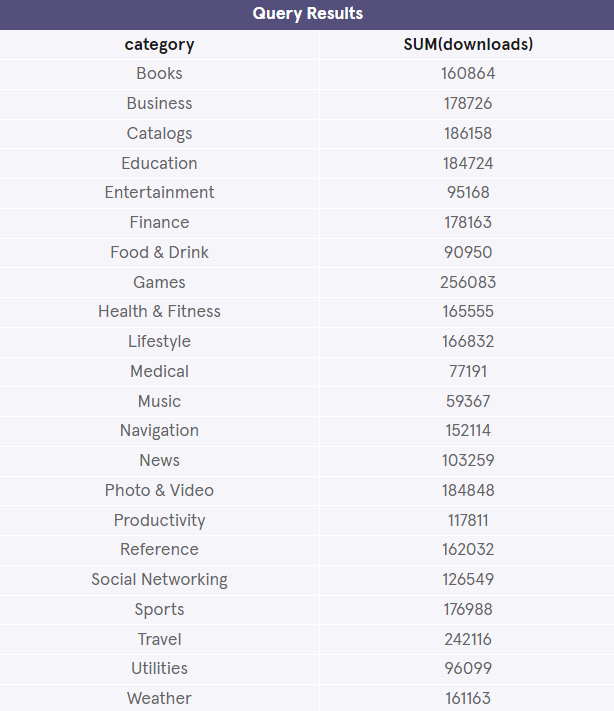
WHERE downloads > 20000

GROUP BY price;\*/

SELECT category, SUM(downloads)

FROM fake\_apps

GROUP BY category;

****

**Group By II**

Sometimes, we want to [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);

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;

Here, the 1 refers to the first column in our SELECT statement, ROUND(imdb\_rating).

**Instructions**

**1.**

Suppose we have the query below:

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

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

Checkpoint 2 Passed

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;

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**

In addition to being able to group data using [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 [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 [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;

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

**Instructions**

**1.**

Suppose we have the query below:

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

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.

Checkpoint 2 Passed

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;

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,

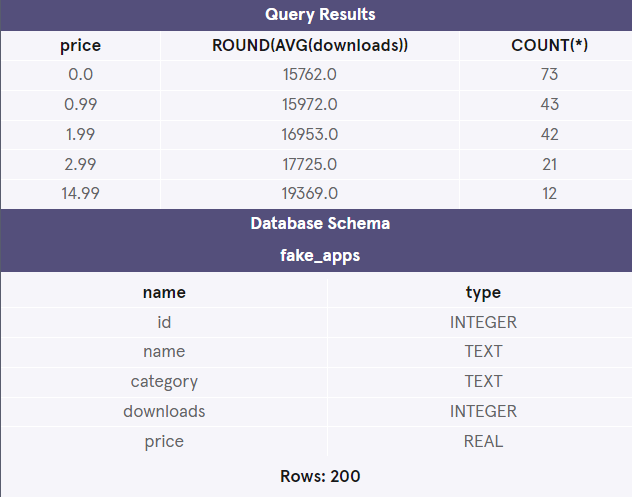
   ROUND(AVG(downloads)),

   COUNT(\*)

FROM fake\_apps

GROUP BY 1

HAVING COUNT(price) > 10;

****