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()`: count the number of rows
- `SUM()`: the sum of the values in a column
- `MAX()`/`MIN()`: the largest/smallest value
- `AVG()`: the average of the values in a column
- `ROUND()`: 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?

Hint

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

aggregate-functions.sqlite

```
SELECT *
FROM fake_apps;
```

Query Results				
id	name	category	downloads	price
3	siliconphase	Productivity	17193	0.0
6	Donzolab	Education	4259	0.99
10	Ittechi	Reference	3874	0.0
13	Subdrill	Education	7132	1.99
14	Anzoom	Health & Fitness	6941	14.99
21	kanity	Health & Fitness	2299	0.0
25	Zathcare	Books	16281	2.99
30	Basecode	Sports	21203	3.99
45	Hayholding	Medical	15845	14.99
59	Sailflex	Productivity	21984	0.99
60	waretam	Catalogs	31087	2.99
62	Donelectrics	News	6650	1.99
81	sonelectrics	Entertainment	9297	1.99
89	Zoodom	Food & Drink	29619	14.99
94	Quohouse	Lifestyle	4109	0.0
95	Quotetech	Health & Fitness	24934	0.0
104	Funtexon	Utilities	11229	0.99
108	Ozerphase	Navigation	14881	0.0
109	Kintonflex	Catalogs	28705	0.0
110	Tamptom	Education	5918	2.99

Count

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

`COUNT()` 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;
```

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.

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;
```

Query Results
COUNT(*)
200

Database Schema fake_apps	
name	**type**
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL

Rows: 200

Query Results
COUNT(*)
73

Database Schema fake_apps	
name	**type**
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL

Rows: 200

Sum

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

`SUM()` 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;
```

Hint
There are 3,322,760 total downloads.

sum.sqlite

```
SELECT SUM(downloads)
FROM fake_apps;
```

Query Results	
SUM(downloads)	
3322760	
Database Schema	
fake_apps	
name	**type**
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL
Rows: 200	

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()` 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()` 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;
```

Hint
1,387 downloads.

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;
```

$14.99 is the price of the most expensive app.

max.sqlite

```
/*SELECT MIN(downloads)
FROM fake_apps;*/


SELECT MAX(price)
FROM fake_apps;
```

Query Results

MAX(price)
14.99

Database Schema

fake_apps

name	type
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL

Rows: 200

Average

SQL uses the `AVG()` 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;
```

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.

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;
```

Query Results
AVG(price)
2.02365

Database Schema	
fake_apps	
name	type
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL

Rows: 200

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()` 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;
```

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!

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;
```

Query Results
ROUND(AVG(price), 2)
2.02

Database Schema	
fake_apps	
name	**type**
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL

Rows: 200

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` 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` statements, but before `ORDER BY` or `LIMIT`.

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?

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.

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)`.

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;
```

Query Results	
category	**SUM(downloads)**
Books	160864
Business	178726
Catalogs	186158
Education	184724
Entertainment	95168
Finance	178163
Food & Drink	90950
Games	256083
Health & Fitness	165555
Lifestyle	166832
Medical	77191
Music	59367
Navigation	152114
News	103259
Photo & Video	184848
Productivity	117811
Reference	162032
Social Networking	126549
Sports	176988
Travel	242116
Utilities	96099
Weather	161163

Group By II

Sometimes, we want to `GROUP BY` 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`.

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
```

Query Results

category	price	AVG(downloads)
Books	0.0	11926.5
Books	0.99	27709.5
Books	1.99	21770.3333333333
Books	2.99	16281.0
Business	0.0	14744.25
Business	0.99	15753.0
Business	1.99	18155.5
Business	2.99	19598.5
Business	14.99	28488.0
Catalogs	0.0	19393.0
Catalogs	0.99	4937.0
Catalogs	1.99	20062.0
Catalogs	2.99	27862.3333333333
Education	0.0	28911.0
Education	0.99	8619.8
Education	1.99	14815.6666666667
Education	2.99	10843.5
Education	14.99	17669.0
Entertainment	0.0	14577.0

Having

In addition to being able to group data using `GROUP BY`, 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` here because we don't want to filter the rows; we want to *filter groups*.

This is where `HAVING` 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` and `LIMIT`.

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.

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;
```

Query Results

price	ROUND(AVG(downloads))	COUNT(*)
0.0	15762.0	73
0.99	15972.0	43
1.99	16953.0	42
2.99	17725.0	21
14.99	19369.0	12

Database Schema

fake_apps

name	type
id	INTEGER
name	TEXT
category	TEXT
downloads	INTEGER
price	REAL

Rows: 200

Review

Congratulations!

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

- `COUNT()`: count the number of rows
- `SUM()`: the sum of the values in a column

- `MAX()`/`MIN()`: the largest/smallest value
- `AVG()`: the average of the values in a column
- `ROUND()`: round the values in the column

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

- `GROUP BY` is a clause used with aggregate functions to combine data from one or more columns.
- `HAVING` 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!