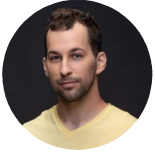


SQL for Data Analysis - Tutorial for Beginners - ep3

2017-06-27



Tomi Mester


Today I'll show you the most essential SQL functions that you will use for finding the maximums or the minimums (MAX, MIN) in a data set and to calculate aggregates (SUM, AVG, COUNT). Then I'll show you some intermediate SQL clauses (ORDER BY, GROUP BY, DISTINCT) that you have to know to efficiently use SQL for data analysis!

And this is going to be super exciting, as we will still use our 7M+ line data set!

Note: to get the most out of this article, you should not just read it, but actually do the coding part with me! So if you are on the phone, I suggest saving this article and continuing on a desktop!

Before we start...

...I recommend going through these articles first – if you haven't done so yet:

1. Set up your  data36 server to practice: How to set up Python, SQL, R and Bash (for non-devs) (<https://data36.com/data-coding-101-install-python-sql-r-bash/>).
2. Install SQL Workbench to manage your SQL stuff better: How to install SQL Workbench for postgresQL (<https://data36.com/install-sql-workbench-postgresql/>).
3. Read the first two episodes of the SQL for Data Analysis series: ep 1 (<https://data36.com/sql-for-data-analysis-tutorial-beginners/>) and ep 2 (<https://data36.com/sql-where-clause-tutorial-beginners-ep2/>).
4. Make sure that you have the flight delays data set imported – and if you don't, check out this video (<https://youtu.be/DGtJAJUIxXw>).

How to Become a Data Scientist (free 50-minute video course by Tomi Mester)

Just subscribe to the Data36 Newsletter here (it's free)!

☐ I accept Data36's Privacy Policy.

(<https://data36.com/privacy-policy/>). (No

spam. Only useful data science related

content. When you subscribe, I'll keep you

updated with new content every week.

data360

You'll get articles, courses, cheatsheets, (https://data36.com), tutorials and many cool stuff.)

Get Access Now!

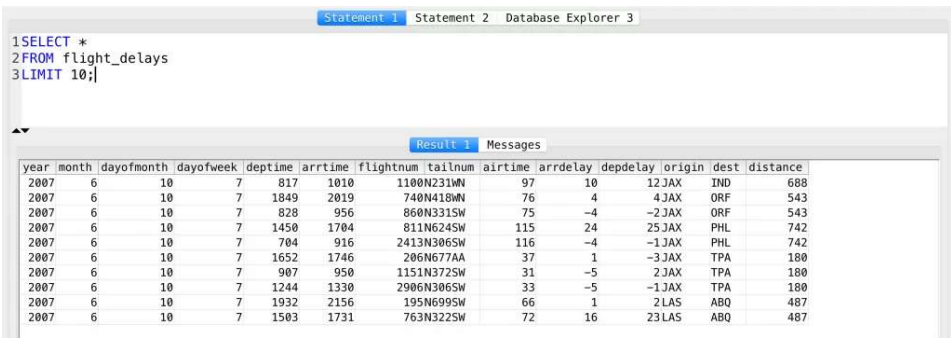
SQL functions to aggregate data

Okay, let’s open SQL Workbench and connect to your data server!

Can you recall our base query?
It was:

```
1 SELECT *
2 FROM flight_delays
3 LIMIT 10;
```

And it returned the first 10 lines of this huge data set.



The screenshot shows a SQL Workbench window with a query editor and a results pane. The query editor contains the following SQL statement:

```
1SELECT *
2FROM flight_delays
3LIMIT 10;
```

The results pane displays the first 10 rows of the `flight_delays` table. The columns are: `year`, `month`, `dayofmonth`, `dayofweek`, `deptime`, `arrtime`, `flightnum`, `tailnum`, `airtime`, `arrdelay`, `depdelay`, `origin`, `dest`, and `distance`.

year	month	dayofmonth	dayofweek	deptime	arrtime	flightnum	tailnum	airtime	arrdelay	depdelay	origin	dest	distance
2007	6	10	7	817	1010	1100N231WN		97	10	12	JAX	IND	688
2007	6	10	7	1849	2019	740N418WN		76	4	4	JAX	ORF	543
2007	6	10	7	828	956	860N331SW		75	-4	-2	JAX	ORF	543
2007	6	10	7	1450	1704	811N624SW		115	24	25	JAX	PHL	742
2007	6	10	7	704	916	2413N306SW		116	-4	-1	JAX	PHL	742
2007	6	10	7	1652	1746	206N677AA		37	1	-3	JAX	TPA	180
2007	6	10	7	907	950	1151N372SW		31	-5	2	JAX	TPA	180
2007	6	10	7	1244	1330	2906N306SW		33	-5	-1	JAX	TPA	180
2007	6	10	7	1932	2156	195N699SW		66	1	2	LAS	ABQ	487
2007	6	10	7	1503	1731	763N322SW		72	16	23	LAS	ABQ	487

We are going to modify this query to get 5 interesting numbers:

- exactly how many flights are in our table (*count*)
- the *sum* of the airtimes (*note: practically speaking, airtime is the flight time*) of these flights
- the *average* arrival delays and the average departure delays
- the *maximum* distance of any of these flights

- the *mini*  data 36 of any of these flights (<https://data36.com>).

All of these are going to be very easy, I promise... **But make sure you are doing the coding part with me,** because 100% understanding of this basic stuff will be important in the long term.

SQL COUNT function to count lines

The easiest aggregation is to count lines in your table, and this is what the COUNT function is for. The only thing you have to change – compared to the above base query – is what you SELECT from your table. Remember? It can be everything (*), or it can be specific columns (arrdelay, depdelay, etc) – and now let's expand this list with *functions*. Copy this query into SQL Workbench and run it:

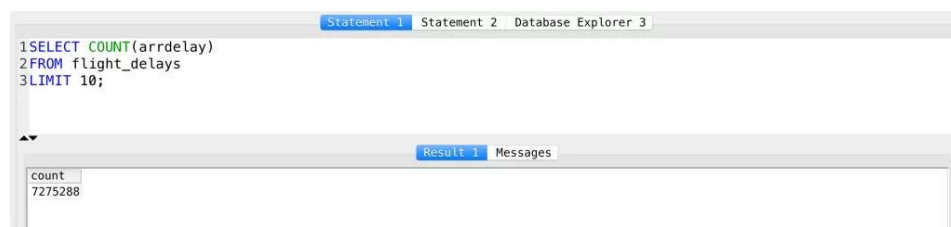
```
1 SELECT COUNT(*)
2 FROM flight_delays
3 LIMIT 10;
```



The result is: 7275288.

The function itself is COUNT, and it says to count the lines using every column (*)... You can change the * to any columns' name (eg. arrdelay) – and you will get the very same number. Try this:

```
1 | SELECT COUNT(*) data36
2 | FROM flight_delays
3 | LIMIT 10;(https://data36.com),
```



Right? Same result: 7275288.

Note 1: This is only true when you don't have NULL values (empty cells) in your table (we don't have NULL values in the flight_delays data set at all). I'll get back to the importance of NULL later.

Note 2: in fact you won't need the LIMIT clause in this SQL query, as you will have only one line of data on your screen. But I figured that sometimes it might be better to keep it there, so even if you mistype anything, your SQL Workbench won't freeze by accidentally trying to return 7M+ lines of data.

SQL SUM function to get summary

Now we want to get the airtime for all flights – added up. In other words: find the sum of column airtime. The SUM function works with the same logic as COUNT does. The only exception, that in this case you **have to** specify the column (in this case airtime). Try this:

```
1 | SELECT SUM(airtime)
2 | FROM flight_delays;
```

The total airtime is a massive 748015545 minutes.



SQL AVG function to get the mean

Our next challenge is to find the mean of the arrival delays and the mean of the departure delays... The function for that is AVG (refers to “average”) – functioning with the exact same logic as SUM and COUNT.

```
1 SELECT AVG(depdelay)
2 FROM flight_delays;
```



Result: 11.36

```
1 SELECT AVG(arrdelay)
2 FROM flight_delays;
```

Result: 10.19

Cool!

SQL MAX and MIN functions to get maximums and minimums

And finally let’s find the maximum and the minimum values of a given column. Finding the maximum and minimum distances for these flights

sounds interesting... MIN and MAX operate just like SUM, AVG and COUNT did:

```
1 SELECT MIN(distance)
2 FROM flight_delays;
```

Result: 11 miles. Well, maybe take the bike next time...



```
1 SELECT MAX(distance)
2 FROM flight_delays;
```


Result: 4962



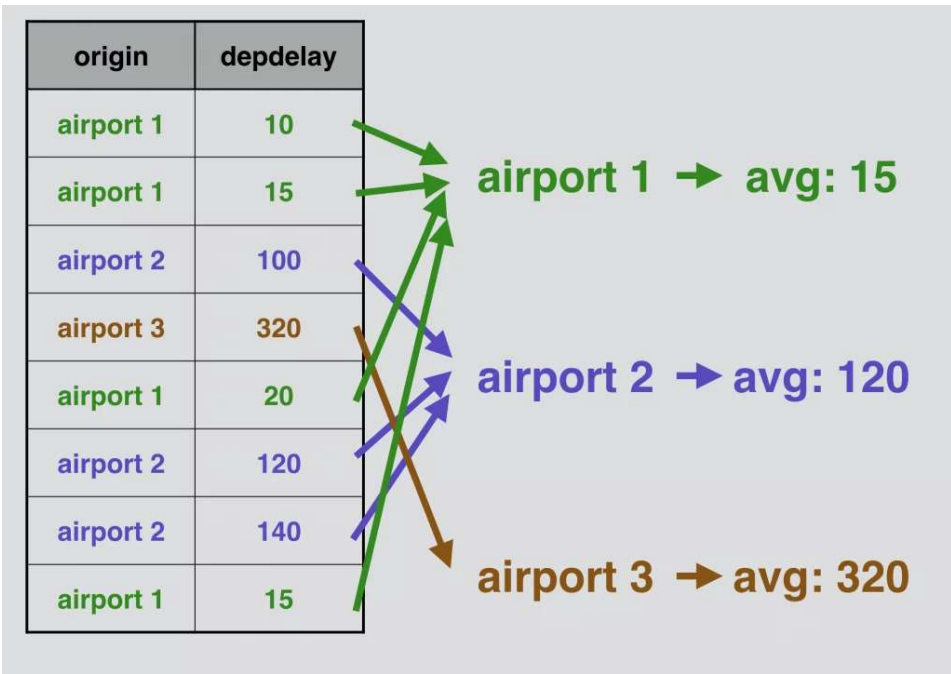
Okay! That was it – these are the basic SQL functions you have to know... It's time to tweak them a little bit.

Basic segmentation analysis with SQL – aka. the GROUP BY clause

As a data analyst or data scientist you will probably do segmentations all the time. For instance, it's interesting to know the average departure delay of all flights (we have just learned that it's 11.36). But when it comes to business decisions, this number is not actionable at all. However, if we turn this information into a more useful

format – let  data360 < it down by
airport – it will instantly become
(<https://data36.com>),
something we can act on!

Here’s a simplified chart showing how SQL performs automatic segmentation based on column values:



The process has three important steps:

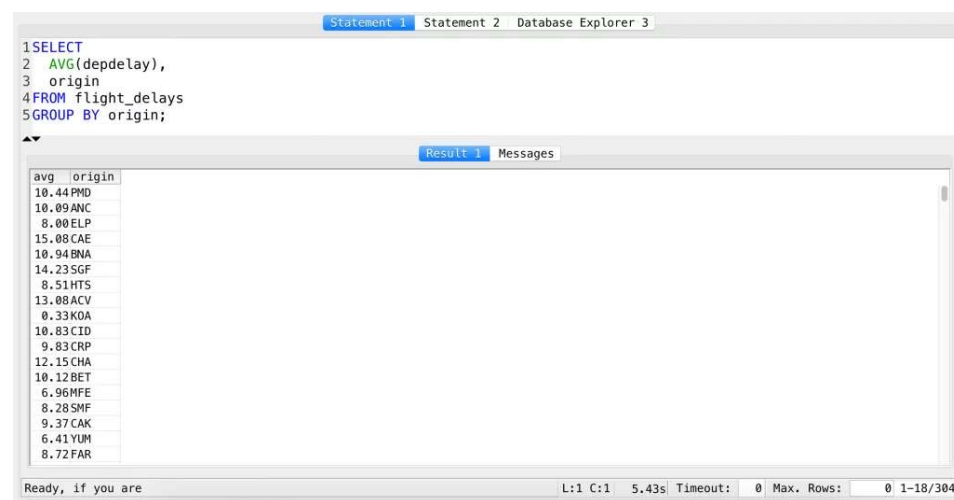
STEP 1 – Specify which columns you want to work with as an input. In our case we want to use the list of the airports (origin column) and the departure delays (depdelay column).

STEP 2 – Specify which column(s) you want to create your segmentation from. For us it’s the origin. SQL automatically looks for every unique value in this column (in the above example – *airport 1*, *airport 2* and *airport 3*), then creates groups from them and sorts each line from your data table into the right group.

STEP 3 – Finally it calculates the averages using the SQL AVG function for each group and returns the results

The only new thing here is the “grouping” at STEP 2. We have an SQL clause for that. It’s called GROUP BY. Let’s see it in action:

```
1 SELECT
2     AVG(depdelay),
3     origin
4 FROM flight_delays
5 GROUP BY origin;
```



avg	origin
10.44	PMD
10.09	ANC
8.00	ELP
15.08	CAE
10.94	BNA
14.23	SGF
8.51	HTS
13.08	ACV
0.33	KOA
10.83	CID
9.83	CRP
12.15	CHA
10.12	BET
6.96	MFE
8.28	SMF
9.37	CAK
6.41	YUM
8.72	FAR

If you scroll through the results, you will see that there are some airports with an average departure delay of more than 30 or even 40 minutes. From a business perspective it’s important to understand what’s going on at those airports. On the other hand it’s also worth taking a closer look at how the good airports (depdelay close to 0) are managing to reach this ideal phase. (Yeah, it’s over-simplified, but just for example...)

But what just happened SQL-wise? We have selected two columns – *origin* and *depdelay*. *origin* has been used to create the segments (**GROUP BY origin**). *depdelay* has been used to

calculate the arrival
delays in these segments
(<https://data36.com>),
(AVG(depdelay)).

Note: As you can see, the logic of SQL is not as linear as it was in bash (<https://data36.com/learn-data-analytics-bash-scratch/>). If you write an SQL query, the first line of it could highly rely on the last line. When you'll write really long and complex queries, this might cause some unexpected errors and thus of course a little headache too... But that's why I find it very, very important to give yourself enough time to practice the basic things and make sure that you fully understand the relationships between the different clauses, functions and other stuff in SQL.

The Junior Data Scientist's First Month

A 100% practical online course.
A 6-week simulation of being a junior data scientist at a true-to-life startup.

“Solving real problems, getting real experience – just like in a real data science job.”

[Learn more...](#)

(<https://data36.com/the-junior-data-scientists-first-month-online-course/>)

Here's a little assignment to practice on and to double-check that you understand everything so far! The task is:

•

Here's my solution:


```
Statement 1 Statement 2 Database Explorer 3
1 SELECT
2   month,
3   SUM(airtime)
4 FROM flight_delays
5 GROUP BY month;
```

Result 1 Messages

month	sum
6	62788140
8	65387750
12	62336691
1	62149018
2	55718006
10	63275589
3	63990803
11	61500900
4	61957903
5	63724389
9	60110929
7	65155347

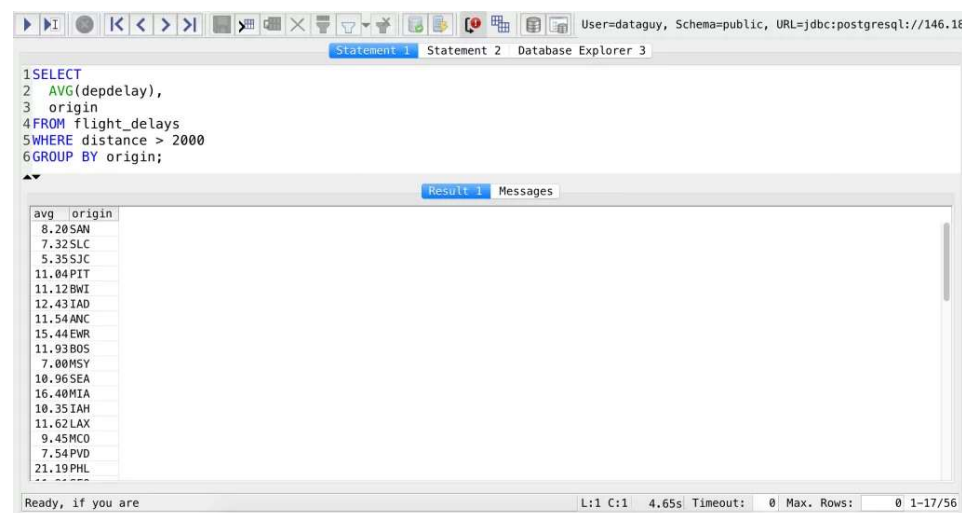
Test yourself #2

Calculate the average departure delay by airport again, but this time use only those flights that flew more than 2000 miles (you will find this info in the distance column).

-  data36
- (<https://data36.com>).
-

Here's the query:

```
1 SELECT
2     AVG(depdelay),
3     origin
4 FROM flight_delays
5 WHERE distance > 2000
6 GROUP BY origin;
```



The screenshot shows a SQL client window with the following query and results:

```
1 SELECT
2     AVG(depdelay),
3     origin
4 FROM flight_delays
5 WHERE distance > 2000
6 GROUP BY origin;
```


avg	origin
8.20	SAN
7.32	SLC
5.35	SJC
11.04	PIT
11.12	BWI
12.43	IAD
11.54	ANC
15.44	EWB
11.93	BOS
7.00	MSY
10.96	SEA
16.40	MIA
10.35	JAH
11.62	LAX
9.45	MCO
7.54	PVD
21.19	PHL

The takeaway from this assignment is something that you might have already realized: you can use the SQL WHERE clause to filter even those columns that are not part of your SELECT statement.

SQL ORDER BY to sort the data based on the value of one (or more) specific column(s)

Let's say we want to see which airport was the busiest in 2007. You can get the number of departures by airport really easily using the COUNT function with GROUP BY clause:

```
1 SELECT
2     COUNT(*)
3     origin (https://data36.com),
4 FROM flight_delays
5 GROUP BY origin;
```



The screenshot shows a SQL query editor with the following query:

```
1SELECT
2  COUNT(*),
3  origin
4FROM flight_delays
5GROUP BY origin;
```

The results are displayed in a table with two columns: 'count' and 'origin'. The data is as follows:

count	origin
380	PHD
19303	ANC
21529	ELP
10224	CAE
59935	BNA
10580	SGF
271	HTS
3734	ACV
15858	KOA
10550	CID
6605	CRP
4355	CHA
1010	BET
4642	MFE
57423	SMF
9079	CAK
3436	YUM

But this list is not sorted by default...
To have that too, you need to add only one more SQL clause: `ORDER BY`.
When you use it, you always have to specify which column you want to order the results by... It's pretty straightforward.

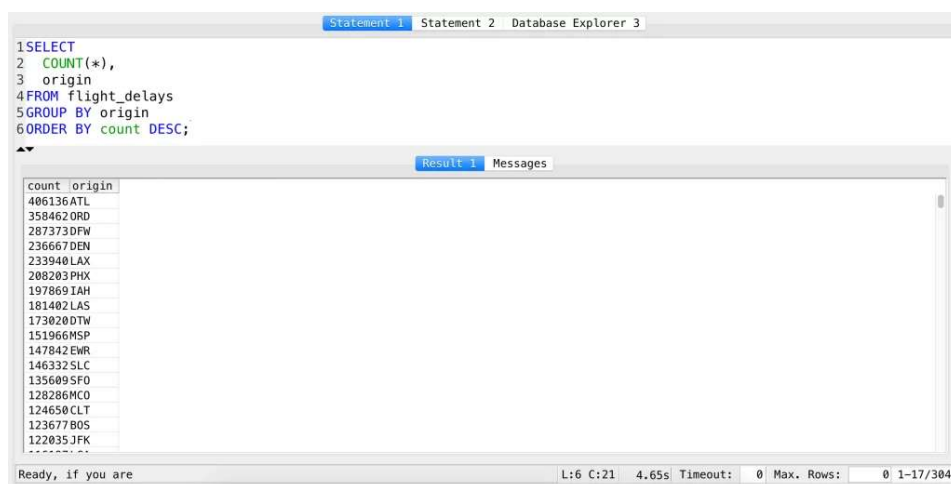
```
1 SELECT
2     COUNT(*),
3     origin
4 FROM flight_delays
5 GROUP BY origin
6 ORDER BY count;
```

Note: the column you will get after the `COUNT` function will be a new column... And it has to have a name – so SQL automatically names it “count” (check the latest screenshot above). When you refer to this column in your `ORDER BY` clause, you have to use this new name. I'll get back to this in my next article in detail. If you find it weird, let's try the same query but with `ORDER BY origin` – and you will understand it instantly.



Hm, almost there. But the problem is that the least busy airport is on the top – in other words, we got a list in ascending order. That’s the default for ORDER BY (in our PostgreSQL database at least). But you can change this to descending order by simply adding the DESC keyword after it!

```
1 SELECT
2   COUNT(*),
3   origin
4 FROM flight_delays
5 GROUP BY origin
6 ORDER BY count DESC;
```



Great!

DISTINCT to get unique values only

This is the last new thing for today. And this will be a very simple one. If you are curious how many different airports are in your table:

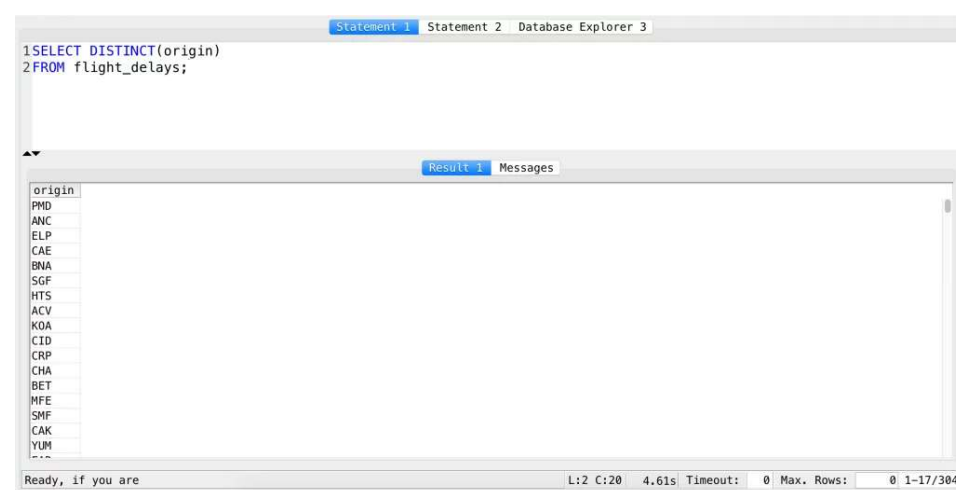
a) you can find it out by using the GROUP BY clause. (Can you figure out how? <https://data36.com>, :-))

b) you can find it out even more easily by using DISTINCT

DISTINCT basically removes all duplicates. Try this:

```
1 SELECT DISTINCT(origin)
2 FROM flight_delays;
```

Now you have only unique airports!



By the way, the GROUP BY version would look like this:

```
1 SELECT origin
2 FROM flight_delays
3 GROUP BY origin;
```

Though result-wise it's pretty much the same, the preferred way to do this is to use the DISTINCT – when making more complex queries, it will help you to keep your query simpler!

Test yourself #3

Today you have learned a ton of small but useful stuff. I'll give you one more assignment that will summarize pretty much everything – even the previous two articles ([ep_1](https://data36.com/sql-for-data-) <https://data36.com/sql-for-data->

analysis-tutorial-36(rs/). and ep 2
(<https://data36.com/sql-where-clause-tutorial-beginners-ep2/>)). This is going to be a difficult one, but you can do it! If it doesn't work at first, try to break it down into smaller tasks, then build and test your query till you get the result.

The task is:

List the:

- **top 5 planes (identified by the tailnum)**
- **by the number of landings**
- **at PHX or SEA airport**
- **on Sundays**

(eg. if the plane with the tailnumber 'N387SW' landed 3 times in PHX and 2 times in SEA in 2007 on any Sunday, then it has a total of 5. And we need the top 5 planes with the higher total.)

Ready? Set! Go!

.
. .

Done? Here's my solution:

```
1 SELECT
2     COUNT(*),
3     tailnum
4 FROM flight_delays
5 WHERE dayofweek = 7
6     AND dest IN ('PHX', 'SEA')
7 GROUP BY tailnum
8 ORDER BY count DESC
9 LIMIT 5;
```



And with comments:

- `SELECT` –» select...
- `COUNT(*)`, –» this function counts the number of lines in a given group; to do that it needs the `GROUP BY` clause later
- `tailnum` –» this will help to specify the groups (referred in the `GROUP BY` function later)
- `FROM flight_delays` –» the name of the table
- `WHERE dayofweek = 7` –» a filter for Sundays only
- `AND dest IN ('PHX', 'SEA')` –» filter for PHX and SEA destinations only
- `GROUP BY tailnum` –» this is the clause that helps us to put the lines into different groups
- `ORDER BY count DESC` –» and let's order by the number of lines in a given group
- `LIMIT 5;` –» list only the top 5 elements.

Conclusion

And that's it! You have learned a lot again today – SQL functions (MIN, MAX, AVG, COUNT, SUM) and new

important SQL  data36 [DISTINCT,
ORDER BY, GROUP BY].
(<https://data36.com>).

If you managed to get the last exercise done by yourself, I can tell you that you have a really good basic knowledge of SQL! Congrats! If not, don't worry, just make sure that you re-read these first 3 chapters ([ep 1 \(https://data36.com/sql-for-data-analysis-tutorial-beginners/\)](https://data36.com/sql-for-data-analysis-tutorial-beginners/)), [ep 2 \(https://data36.com/sql-where-clause-tutorial-beginners-ep2/\)](https://data36.com/sql-where-clause-tutorial-beginners-ep2/), [ep 3 \(https://data36.com/sql-best-practices-data-analysts/\)](https://data36.com/sql-best-practices-data-analysts/)), before you continue with [episode 4 \(https://data36.com/sql-best-practices-data-analysts/\)](https://data36.com/sql-best-practices-data-analysts/)!

- If you want to learn more about how to become a data scientist, take my 50-minute video course: [How to Become a Data Scientist. \(https://data36.com/how-to-become-a-data-scientist/\)](https://data36.com/how-to-become-a-data-scientist/). (It's free!)
- Also check out my 6-week online course: [The Junior Data Scientist's First Month video course. \(https://data36.com/jds/\)](https://data36.com/jds/).

Cheers,
Tomi Mester

