



# SQL for Data Analysis – Tutorial for Beginners – ep4

*Written by Tomi Mester on July 3, 2017*

*Last updated on December 10, 2020*

You have already learned a lot about the basics of SQL for data analysis. I figured now would be a nice time to have an episode focusing only on SQL best practices. So I wrote one! In this article you will learn:

- How to format your SQL query to make it more reusable
- When to use capital and lowercase characters
- How to use aliases
- How to add comments
- And more...

If you are new here, it's better to start with these articles (but for this particular article it's not required):

1. Set up your own data server (including your SQL server) to practice: [How to set up Python, SQL, R and Bash \(for non-devs\)](#)
2. Install SQL Workbench to manage your SQL queries better: [How to install SQL Workbench for postgresSQL](#)
3. Read the first three episodes of the SQL for Data Analysis series: [ep1](#), [ep2](#) and [ep3](#)

4. Make sure that you have the flight\_delays data set imported – and if you don't, check out [this video](#).

All set? Let's see some SQL best practices!

## SQL Cheat Sheet

Do you want to learn faster? Join the Data36 Inner Circle and download the SQL Cheat Sheet. Just enter your email address:

☐ I accept Data36's [Privacy Policy](#). (No spam. Only useful data science related content. When you subscribe, I'll keep you updated with a couple emails per week. You'll get articles, courses, cheatsheets, tutorials and lots of cool stuff that I only share with the Data36 "inner circle.")

Get Access Now!

## List of SQL best practices (content):

1. [Watch out for keyword order!](#)
2. [Make your SQL queries nicer with formatting \(spaces and new lines\)](#)
3. [Upper case vs lower case characters](#)
4. [Use aliases! \(SQL AS\)](#)
5. [Add comments!](#)
6. [Always use the actual name of the column](#)
7. [Avoid SELECT \\*](#)

## SQL best practice #1: The order of your keywords

The order of your SQL keywords counts in your query. This is not even a best practice, this is a must. Looking only at the SQL keywords we have learned so far, this is the proper order:

1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. ORDER BY
6. LIMIT

If you don't use it in this order, your query won't work. Simple as that.

It's good to know that this is not the same order in which the computer processes the query. The logical order is this:

1. FROM
2. WHERE
3. GROUP BY
4. SELECT
5. ORDER BY
6. LIMIT

SQL first checks which data table we will work with. Then it checks the filters. After that it groups the data. Finally it gets the data – and if necessary, sorts it and prints only the first X rows.

It's great to keep this logic in mind, but the writing order is more important for now – so learn that one... But later on, when you'll write more complex SQL queries, it'll be really useful to know the execution order (that the computer uses), as well, because that will help you avoid a few logical errors in your code.

## SQL best practice #2: Indentations and new lines

You have seen me write queries in different styles.

E.g.

A)

```
SELECT * FROM flight_delays WHERE origin = 'PHL';
```

B)

```
SELECT *  
FROM flight_delays  
WHERE origin = 'PHL';
```

C)

```
SELECT  
  *  
FROM  
  flight_delays  
WHERE  
  origin = 'PHL';
```

These three are the same query. Spaces and line breaks do not affect the outcome of an SQL query. The only thing that matters is the semicolon ( ; ) that clearly indicates the end of the SQL query. Apart from that you can use as many spaces and line breaks between the different keywords as you want.

So which style is the best? There is nothing like a set-in-stone rule to answer this question. It depends on many things (and mostly on personal preference), but just in general: people tend to use version C) the most. Why? Because that's the easiest to read. It doesn't make a real difference in a short query like this one, but it does make a huge difference in a very complex one (even 100 or more lines). Being organized with your SQL queries is key to avoiding unnecessary headaches while you are debugging your code.

*Note: I have to admit that I get lazy as well sometimes and when I write an ad-hoc one-time short SQL query, I type everything in one line (Style A). But I break the line immediately if I see that I need to make it more complex.*

You can experiment with your own SQL style, but I suggest following these 3 simple recommendations:

1. Use line breaks at least before the main SQL keywords (e.g. `SELECT` , `FROM` , `WHERE` , etc.). Actually, I prefer to use line breaks before column names, table names, and each `WHERE` condition too...
2. Use indentation for column names, `WHERE` conditions (especially with `AND` , `OR` ) and similars!
3. Stay consistent!

## Example:

```
SELECT
    animal,
    water_need
FROM
    zoo
WHERE
    animal LIKE '_____'
    AND animal <> 'tiger'
    AND water_need > 200;
```

## SQL best practice #3: Upper case vs lower case

I've already mentioned in [episode 1](#) that SQL is not case sensitive for the [reserved SQL keywords](#) (eg. `SELECT` , `AND` , `OR` , `GROUP BY` , etc). But it **can be** case sensitive for column names, table names and field values – this depends on your settings.

*Note: for the postgresSQL database we use in these tutorial articles, only **field values** are case sensitive.*

Regardless of case sensitivity, it's quite common to keep these best practices:

1. All reserved SQL keywords (eg. `SELECT` , `AND` , `OR` , `GROUP BY` , etc.) should be written in capitals.
2. All column names and table names should be written in lowercase (except special situations of course, when the name of the column contains uppercase characters initially).
3. Everything that comes between apostrophes ( `'` ) — so mostly field values — should be written exactly as they are in the tables. (E.g. `'tiger'` from the zoo table lowercase but `'PHL'` from the `flight_delays` table uppercase.)

## Example:

```
SELECT
  animal,
  water_need
FROM
  zoo
WHERE
  animal LIKE '_____'
  AND animal <> 'tiger'
  AND water_need > 200;
```

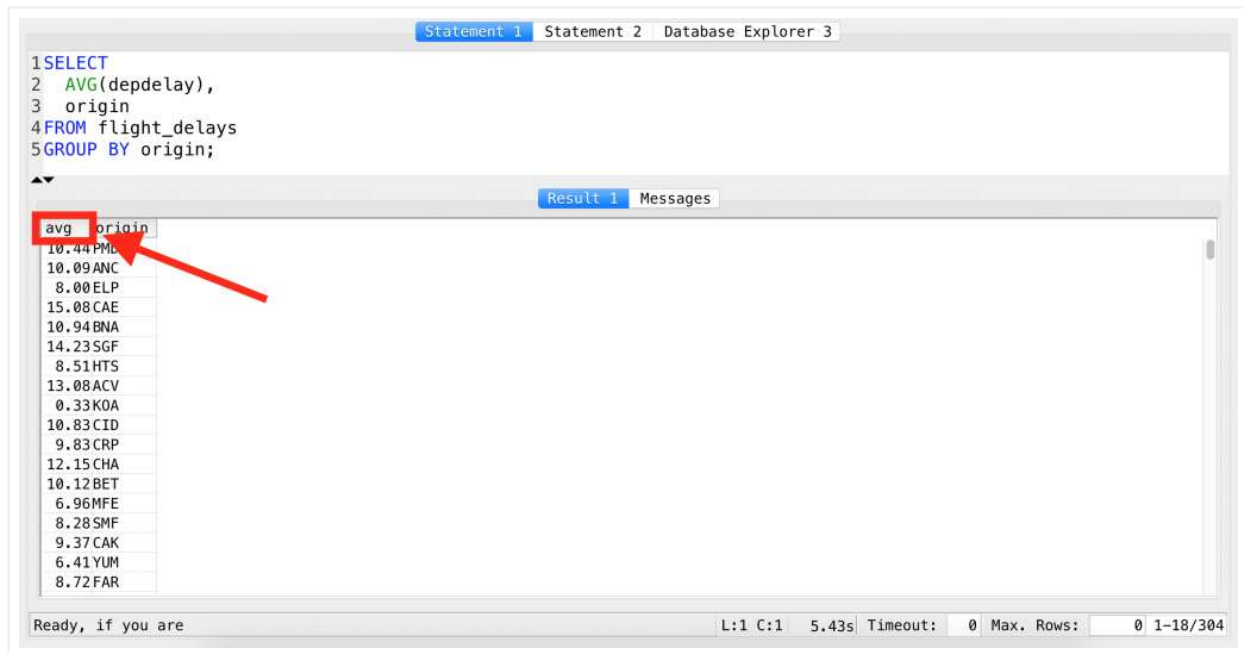
*Note: if you use SQL tools like [SQL Workbench](#), the different keywords will also be colored, which helps a lot in reading and processing.*

## SQL best practice #4: Aliases (SQL AS )

Using the [SQL functions](#) will give new names to your newly generated columns by default. For example, when we have calculated averages for different segments in the previous episode, the new column's name – that contained the actual averages – was `avg` . Just to recall:

```
SELECT
  AVG(depdelay),
  origin
```

```
FROM flight_delays  
GROUP BY origin;
```



avg	origin
10.44	PML
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

Again: the name `avg` is an automatically generated default name by SQL. But we can change it to anything, using *aliases*.

E.g. if in the above results we want to see `average_depdelay` instead of `avg`, we can achieve it like this:

```
SELECT  
AVG(depdelay) AS average_depdelay,  
origin  
FROM flight_delays  
GROUP BY origin;
```

```

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

```

average_depdelay	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

Ready, if you are L:1 C:1 8.57s Timeout: 0 Max. Rows: 0 1-17/304

We have simply added AS average\_depdelay after the function. So the SQL keyword itself looks like this:

AS any\_new\_name\_you\_want\_to\_see .

Now go bananas with the previous query:

```

SELECT
  AVG(depdelay) AS average_depdelay,
  origin AS o
FROM flight_delays AS fd
GROUP BY o;

```

```

1SELECT
2  AVG(depdelay) AS average_depdelay,
3  origin AS o
4FROM flight_delays AS fd
5GROUP BY o;

```

average_depdelay	o
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

Ready, if you are L:1 C:1 4.04s Timeout: 0 Max. Rows: 0 1-17/304



The SQL `AS` (and “*aliasing*”) works with multiple columns and even with table names (which, by the way, is going to be very handy in the [next episode](#), where I’ll introduce the [SQL JOIN](#) clause).

Also: did you notice, that I referred to the `origin` column using its new name `o` in the `GROUP BY` clause too? (By the way, it would have worked with “`origin`” as well.)

It’s good to know two things:

1. Using `AS` assigns the new header name temporarily. The change is effective only in the specific SQL query.
2. Above I gave you some silly examples... The whole point of using the SQL `AS` trick is to simplify and shorten the names — and make the readability and usability of your code better. So be pragmatic and smart when you use aliases!

## SQL best practice #5: Comments

Commenting your code is highly recommended in any programming language. For instance, if you are working in a team, it helps the team to understand your code. But you should comment for your own best interest too! If you write a query today, I guarantee that in 1 year you won’t have a clue what the heck that `COUNT(origin)` function was good for... So the next SQL best practice is about helping yourself remember stuff!

I like to look at commenting as sending messages to myself into the future. And I have to tell you, it’s always a pleasure to receive these kind messages from my past self... And of course, on the other hand, many times I’m also very angry that past me wasn’t more detailed.

Anyway. Commenting is important, no question about that. But how can you do it in SQL?

Very simple. Type this: -- .

After the double-dash, nothing in that line will be executed.

Try this:

```
SELECT -- hello this is a comment
      AVG(depdelay),
      origin
-- this is a comment too
FROM flight_delays
GROUP BY origin;
```



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

```
1 SELECT -- hello this is a comment
2       AVG(depdelay),
3       origin
4 -- this is a comment too
5 FROM flight_delays
6 GROUP BY origin;
```

The results pane shows the output of the query, which is a table with two columns: avg and origin. The data is as follows:

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

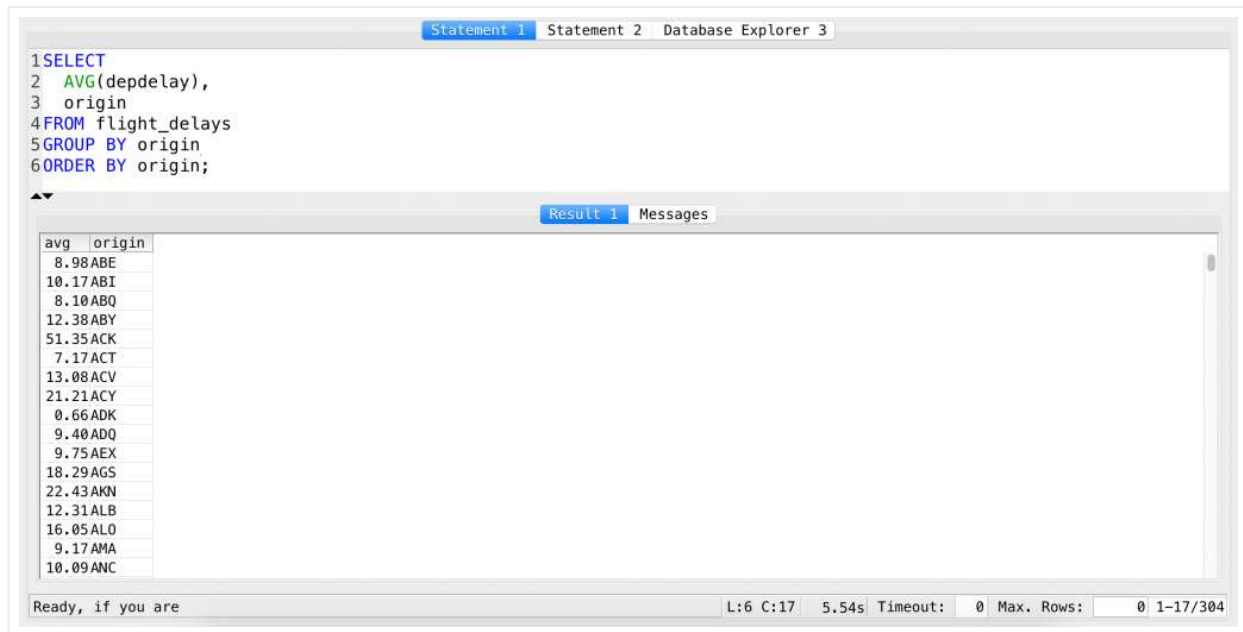
If you run it, you will see that the two comments won't affect your results.

## SQL best practice #6: ORDER BY column name

There is a little “trick” you will meet sooner or later. So I want you to hear it from me first — to avoid any trouble with it in the future! When you use `ORDER BY` and `GROUP BY`, you can of course use column *names* (as we have done so far) but you can also use column *numbers* (that's the new thing).

E.g. using column *names*:

```
SELECT
  AVG(depdelay),
  origin
FROM flight_delays
GROUP BY origin
ORDER BY origin;
```



The screenshot shows a SQL IDE window with three tabs: 'Statement 1', 'Statement 2', and 'Database Explorer 3'. The 'Statement 1' tab is active and contains the following SQL query:

```
1SELECT
2  AVG(depdelay),
3  origin
4FROM flight_delays
5GROUP BY origin
6ORDER BY origin;
```

Below the query editor, there is a 'Result 1' tab and a 'Messages' tab. The 'Result 1' tab is active and displays the results of the query in a table with two columns: 'avg' and 'origin'. The table contains 20 rows of data, sorted by 'origin'.

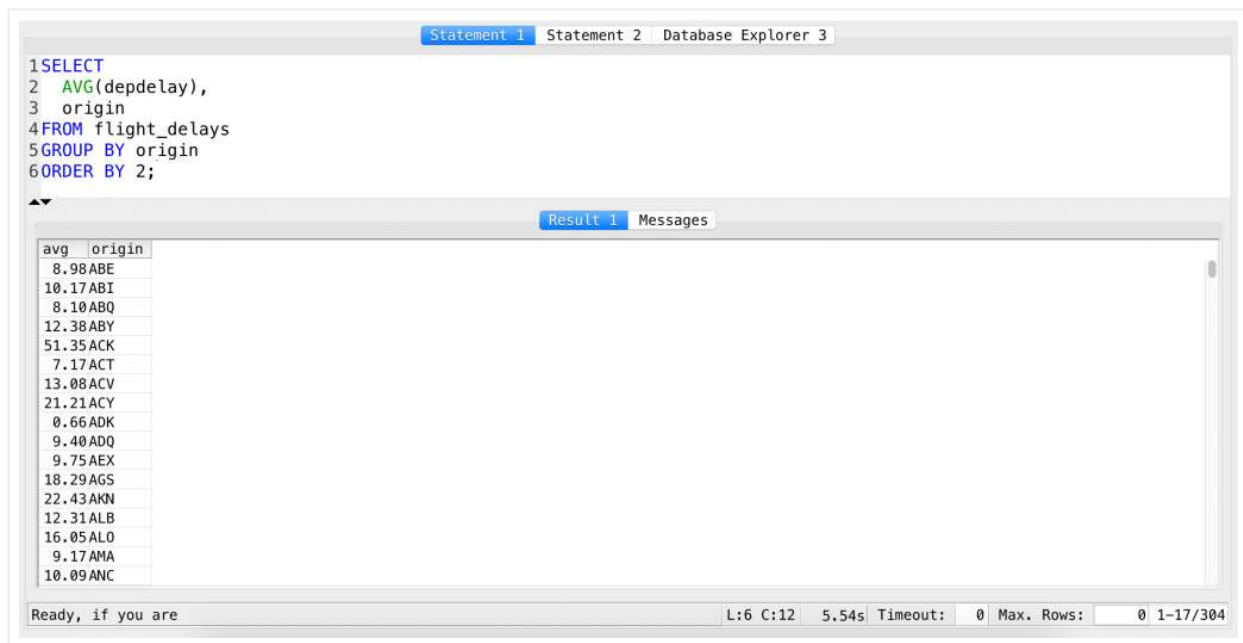
avg	origin
8.98	ABE
10.17	ABI
8.10	ABQ
12.38	ABY
51.35	ACK
7.17	ACT
13.08	ACV
21.21	ACY
0.66	ADK
9.40	ADQ
9.75	AEX
18.29	AGS
22.43	AKN
12.31	ALB
16.05	ALO
9.17	AMA
10.09	ANC

At the bottom of the IDE window, there is a status bar with the text 'Ready, if you are' and a progress indicator showing 'L:6 C:17 5.54s Timeout: 0 Max. Rows: 0 1-17/304'.

And the same query using column *numbers*:

```
SELECT
  AVG(depdelay),
  origin
FROM flight_delays
GROUP BY origin
ORDER BY 2;
```

The 2 after the ORDER BY clause refers to origin – that’s the second column in our query. (If it were 1 , it would be ordered by the first column, AVG(depdelay) .)



Try it yourself... and then **never use it again!**

Wait, what? Why?

Imagine a very common scenario: you tweak your query and you add an extra column (eg. `SUM(depdelays)`) as first in your `SELECT` statement... If you use column names, everything will just work fine. If you use column numbers instead, your query will order by the wrong column (because you should have changed the column number in the `ORDER BY` clause too, but you might have forgotten).

So do yourself a big favour and don't add unnecessary points of failure to your query – your job will be complicated enough without those too!

So the sixth SQL best practice is: Always use the actual name of the columns when you refer to them (either in `ORDER BY` or `GROUP BY`) and never use the number.

## SQL best practice #7: Avoid `SELECT *`

Remember my first [SQL article](#)? I said that this is the base query you'll have to learn, use and expand most of the time:

```
SELECT * FROM data_table_name;
```

Now you know enough about SQL to learn the terrible truth: we don't use `*` in real data projects' real queries – at all.

There are tons of [good reasons](#) for that, but the top two are efficiency and readability. If you need one column from a table, why would you print all of them? It would mean you have to move more data from your SQL server to your computer – slowing down the process unnecessarily. And at the same time, if you add `*` and not column names in your query, you won't have any clue what columns you have in your data table when you want to change something...

Use column names, not `*`.

## Conclusion

Those were my 6+1 SQL best practices for today. I hope they will help you be more efficient and practical with your SQL tasks! For further reading on SQL styles and formatting I recommend checking out [this nice website](#) by Simon Holywell.

And stay with me, because [in the next episode](#) I'll show you one of the most well-known and appreciated features of SQL: `JOIN`.

## SQL for Aspiring Data Scientists (7-day online course)

I've created an online course that will take you from zero to intermediate level with SQL in 7 days. Go ahead and check it out here:



More info...

- 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](#). (It's free!)
- Also check out my 6-week online course: [The Junior Data Scientist's First Month video course](#).

Cheers,

***Tomi Mester***

July 3, 2017 In Coding In Data Science and Analytics

#analytics #best practice #data coding #data science #learn data science #learn to code #sql #SQL best practice #tomi mester

← PREVIOUS POST

NEXT POST →

0 Comments

---

## Leave a Reply

COMMENT

NAME \*

EMAIL \*

WEBSITE

*Post Comment*