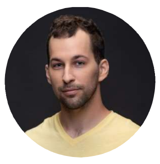
data36

(https://data36.com)

SQL for Data Analysis – Tutorial for Beginners – ep2

2017-06-12



Tomi Mester

This is the second episode of my SQL for Data Analysis (for beginners) series, and today I'll show you every tiny little detail of the **SQL WHERE clause**. It's not by accident that I've dedicated a whole article to this topic; the `WHERE` clause is essential if you want to select the right bit of your data from your data table!

In the first half of this article I'll show you the different operators. In the second half, we will finally import our favorite 7.000.000+ rows dataset (the one with the airplane delays). And at the end of the tutorial, I'll give you a few assignments to test your SQL knowledge and practice a bit!

If you are new here, let's start with these articles first:

1. *How to install Python, SQL, R and Bash (for non-devs) (https://data36.com/data-coding-101-install-python-sql-r-bash/)*
2. *SQL for Data Analysis – Tutorial for Beginners – ep1*

(https://data36.com/sql-for-analysis-tutorial-beginners/)

3. *How to install SQL Workbench for postgreSQL* (https://data36.com/install-sql-workbench-postgresql/)

Make sure your SQL data environment works!

If you have already done the above mentioned articles, you most probably have these in place… but just in case, please double-check if you have these three things:

1. Your fully-functioning data (from this article (https://data36.com/data-coding-101-install-python-sql-r-bash/))
2. SQL Workbench with an established connection to your data server (from this article (https://data36.com/install-sql-workbench-postgresql/))
3. The `zoo` data set (from this article (https://data36.com/sql-for-data-analysis-tutorial-beginners/))

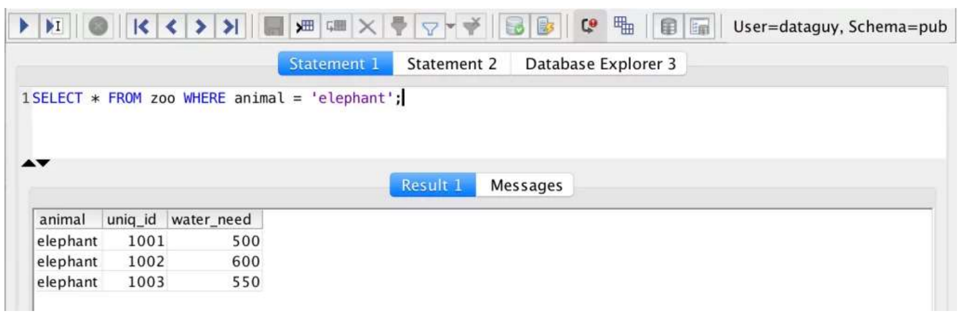
Check-check-check? Okay, you are ready to learn new things!

SQL WHERE Clause

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 you save this article and continue on a computer!

In SQL for Data Analysis episode 1 (https://data36.com/sql-for-data-analysis-tutorial-beginners/) we did this from the Terminal, but this time please re-run this query from *SQL Workbench*:

```
SELECT * FROM zoo WHERE animal =
 'elephant';
```



```
SELECT * FROM zoo WHERE animal = 'elephant';
```

```
animal    uniq_id  water_need
elephant     1001         500
elephant     1002         600
elephant     1003         550
```

This is the most basic example to use the SQL WHERE clause. But you can (and have to) be more sophisticated.

SQL WHERE with comparison operators

Well first of all – you can do more than *"something equals something."* It can be *"greater than,"* *"less than","* *"not equals to,"* etc…

Here's a short list of the comparison operators that work with the SQL `WHERE` clause:

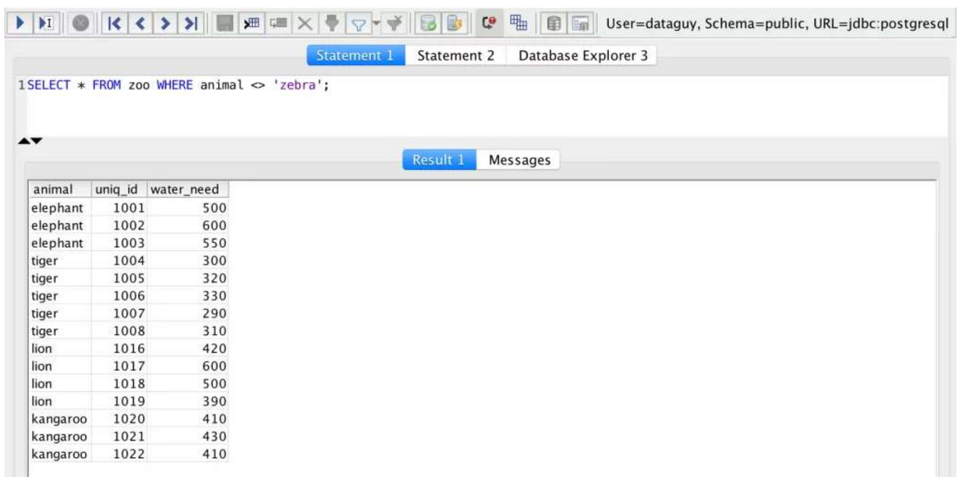
comparison operator	What does it mean?
`=`	equals to
`<>`	not equals to
`!=`	not equals to
`<`	less than
`<=`	less-equal than
`>`	greater than
`>=`	greater-equal than

Let's see some concrete examples:

1. How would you select all the animals, that are not zebras?

Solution:

```
SELECT * FROM zoo WHERE animal <>
 'zebra';
```



User=dataguy, Schema=public, URL=jdbc:postgresql

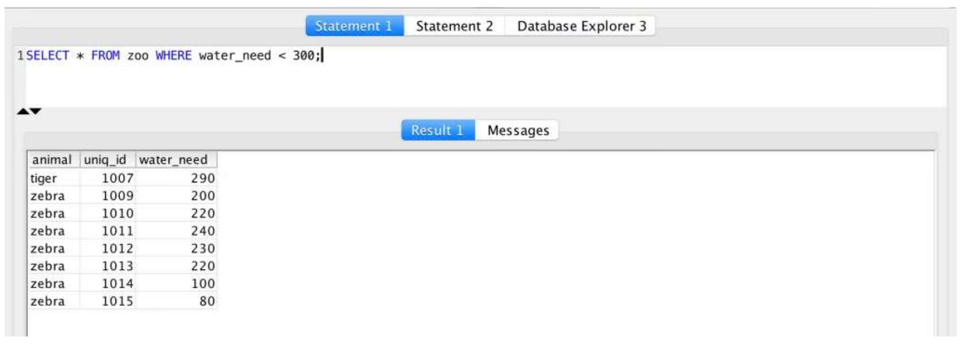
```
SELECT * FROM zoo WHERE animal <> 'zebra';
```

animal	uniq_id	water_need
elephant	1001	500
elephant	1002	600
elephant	1003	550
tiger	1004	300
tiger	1005	320
tiger	1006	330
tiger	1007	290
tiger	1008	310
lion	1016	420
lion	1017	600
lion	1018	500
lion	1019	390
kangaroo	1020	410
kangaroo	1021	430
kangaroo	1022	410

2. How would you select all the animals for whom the `water_need` is less than `300`?

Solution:

```
SELECT * FROM zoo WHERE water_need
 < 300;
```



```
SELECT * FROM zoo WHERE water_need < 300;
```

animal	uniq_id	water_need
tiger	1007	290
zebra	1009	200
zebra	1010	220
zebra	1011	240
zebra	1012	230
zebra	1013	220
zebra	1014	100
zebra	1015	80

Okay, I think you got it!
So let's take this to the next level!

SQL WHERE clause with LIKE operator

Sometimes you want to do your selection based on a pattern. Take a look at this fictional data set:

user_name	user_id
frank	1001
franky	1002

frankenstein	1003
francis	1004
frappuccino	1005

Let’s say, you want to select all the rows where the `user_name` starts with ‘`frank`‘.

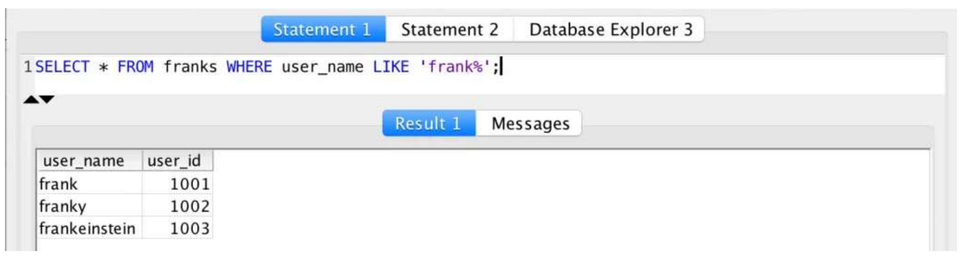
Now it’s time to boost your `WHERE` clause with a `LIKE` operator:

```
SELECT * FROM franks WHERE
 user_name LIKE 'frank%';
```

How does it work?

The `LIKE` logical operator is just like the `>`, `=` or `!=` — so it’s a ***comparsion operator***. But it tells SQL that you want to filter results ***based on a pattern*** — and not on an exact value. Once you typed `LIKE` you specify what the pattern is exactly. Let’s say it’s `'frank%'`! Do you see anything tricky here? Yepp, it’s the `%` character – which in this case means that **any string of characters (of any length) can follow `frank`.**

Hence your result will be:



```
1 SELECT * FROM franks WHERE user_name LIKE 'frank%';
```

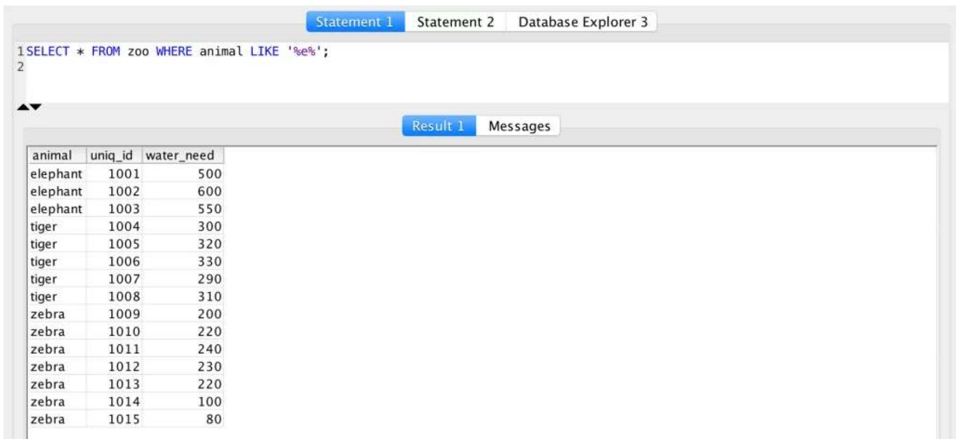
```
user_name     user_id
frank            1001
franky           1002
frankeinstein    1003
```

To hammer this home, let’s go back to our zoo dataset and try to answer these questions:

1. How would you select all an whose name contains at least one e character?

Solution:

```
SELECT * FROM zoo WHERE animal LIKE
 '%e%';
```



```
1 SELECT * FROM zoo WHERE animal LIKE '%e%';
2
```

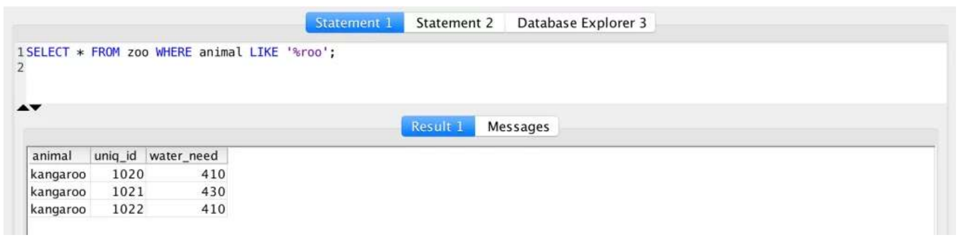
animal	uniq_id	water_need
elephant	1001	500
elephant	1002	600
elephant	1003	550
tiger	1004	300
tiger	1005	320
tiger	1006	330
tiger	1007	290
tiger	1008	310
zebra	1009	200
zebra	1010	220
zebra	1011	240
zebra	1012	230
zebra	1013	220
zebra	1014	100
zebra	1015	80

It needed a % at the beginning and also at the end of the pattern, because the ‘e‘ character could be anywhere in the word, not just at the end or the beginning. (If you don’t get it, try removing the first % and you will understand it immediately!)

2. How would you select all animals whose name ends with ‘roo‘?

Solution:

```
SELECT * FROM zoo WHERE animal LIKE
 '%roo';
```



```
1 SELECT * FROM zoo WHERE animal LIKE '%roo';
2
```

animal	uniq_id	water_need
kangaroo	1020	410
kangaroo	1021	430
kangaroo	1022	410

There is another type of tricky character: _. (Oh, by the way, from now on I’ll just use its official name: ***wildcard***.)

- % means any string of characters of any length.

- _ means any character you
  but only one.

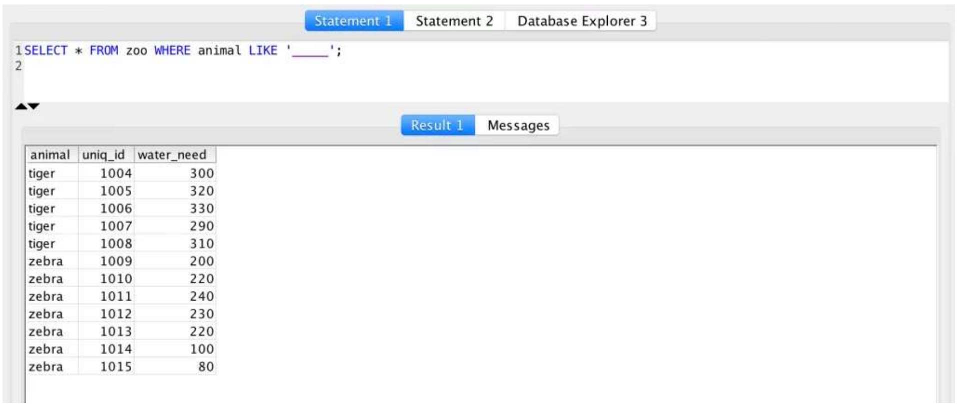
E.g. the word `tiger` can be found using wildcards like `'t%'`. But you can use `'t____'` too.

Knowing this trick, here's another short question:

3. How would you select all animals whose species name is exactly five characters long?

Solution:

```
SELECT * FROM zoo WHERE animal LIKE
 '_____';
```


Statement 1 | Statement 2 | Database Explorer 3

```
SELECT * FROM zoo WHERE animal LIKE '_____';
```

Result 1 | Messages

animal	uniq_id	water_need
tiger	1004	300
tiger	1005	320
tiger	1006	330
tiger	1007	290
tiger	1008	310
zebra	1009	200
zebra	1010	220
zebra	1011	240
zebra	1012	230
zebra	1013	220
zebra	1014	100
zebra	1015	80

This is how the SQL `WHERE` clause works with the `LIKE` operator.

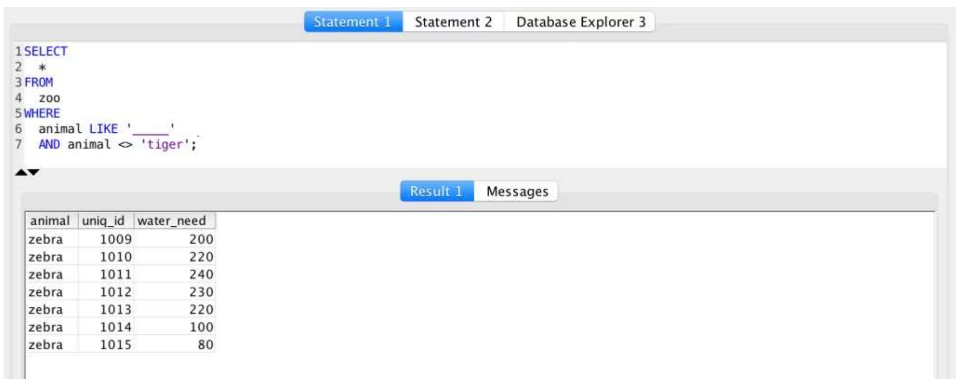
How to combine different conditions in an SQL WHERE clause

You can combine multiple conditions — using ***logical operators***! There are more, but here I'll highlight the two most important: `AND` and `OR`.

Let's say we want to select all the animals whose name is exactly five characters long, except for `tigers`.

We have two conditions and we that with SQL WHERE boosted with the AND logical operator.

```
SELECT
  *
FROM
  zoo
WHERE
  animal LIKE '_____'
  AND
  animal <> 'tiger';
```


Statement 1 | Statement 2 | Database Explorer 3

```
SELECT
 *
FROM
  zoo
WHERE
  animal LIKE '_____'
  AND animal <> 'tiger';
```

Result 1 | Messages

animal	uniq_id	water_need
zebra	1009	200
zebra	1010	220
zebra	1011	240
zebra	1012	230
zebra	1013	220
zebra	1014	100
zebra	1015	80

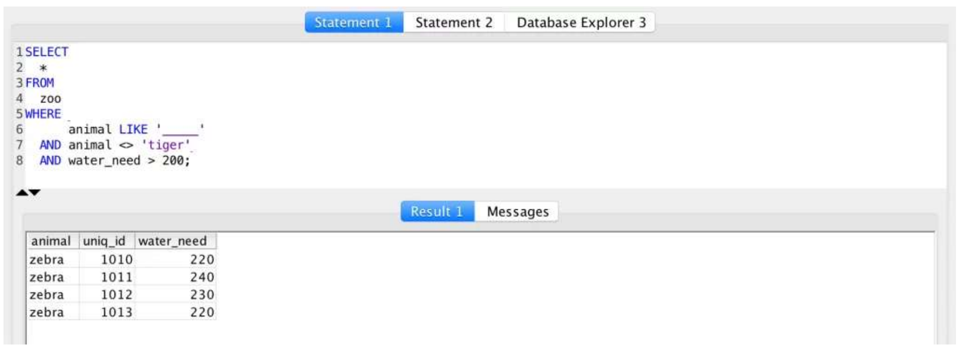
Nice, zebras only!

Now twist it a bit and select only those animals, that:

- have a name exactly five characters long
- are not `tigers`
- have a `water_need` greater than `200`

Solution:

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


Statement 1 | Statement 2 | Database Explorer 3

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

Result 1 | Messages

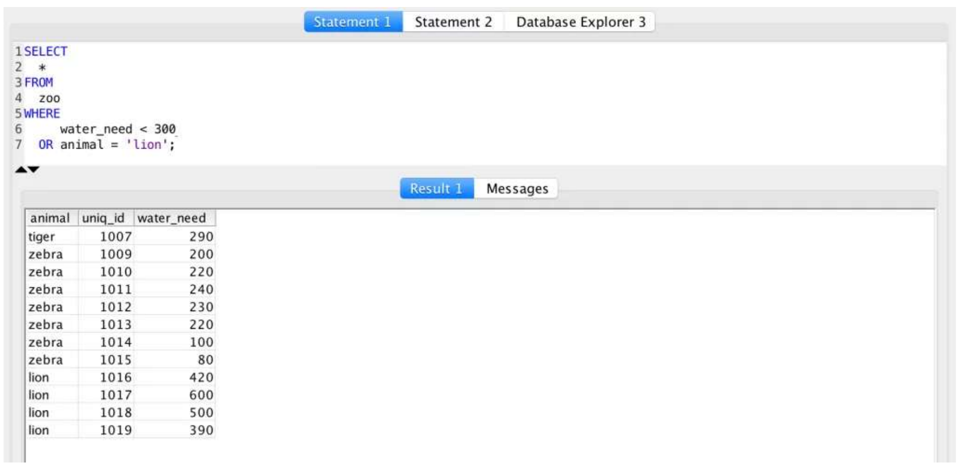
animal	uniq_id	water_need
zebra	1010	220
zebra	1011	240
zebra	1012	230
zebra	1013	220

Yes: you can use multiple `AND` operators.

Finally, let's see an example of the `OR` operator!

```
SELECT
  *
FROM
  zoo
WHERE
  water_need < 300
  OR
  animal = 'lion';
```

It returns all the animals that are lions, plus all the animals that have less than `300` `water_need`.



Statement 1 | Statement 2 | Database Explorer 3

```
SELECT
  *
FROM
  zoo
WHERE
      water_need < 300
  OR animal = 'lion';
```

Result 1 | Messages

animal	uniq_id	water_need
tiger	1007	290
zebra	1009	200
zebra	1010	220
zebra	1011	240
zebra	1012	230
zebra	1013	220
zebra	1014	100
zebra	1015	80
lion	1016	420
lion	1017	600
lion	1018	500
lion	1019	390

Get it?

- `AND` returns every row where all the conditions are true.
- `OR` returns every row where at least one of the conditions is true.

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

[Learn more... (https://data36.com/the-junior-data-scientists-first-month-online-course/)](https://data36.com/the-junior-data-scientists-first-month-online-course/)

The SQL IN operator

Imagine a situation where you want to select all the animals whose unique id is any of these: `1001`, `1008`, `1012`, `1015`, `1018`.

You have already learned a way to do that! Can you find it out?
It’s the SQL WHERE clause with multiple `OR` operators. Something like this:

```
SELECT
  *
FROM
  zoo
WHERE
  uniq_id = 1001
  OR uniq_id = 1008
  OR uniq_id = 1012
  OR uniq_id = 1015
  OR uniq_id = 1018;
```

While it's a valid solution, it's certainly not the most elegant way to get the job done! If you want to reduce the number of `OR`s in your SQL WHERE clause, you should use the `IN` operator instead.

This SQL query returns the exact same result as the previous one:

```
SELECT
  *
FROM
  zoo
WHERE
  uniq_id IN
 (1001,1008,1012,1015,1018);
```

It's a little bit shorter — so it's much nicer. (You know, when it gets to coding, we don't like to repeat things.)

The SQL NOT operator

There is an extra logical operator, which is by the way extremely simple. `NOT` will modify your logical operator to do its opposite instead. Let's get back to a previous example with the 5 characters long animals:

```
SELECT * FROM zoo WHERE animal
LIKE '_____';
```

How to select all the animals th
not 5 characters long?

```
SELECT * FROM zoo WHERE animal
NOT LIKE '_____';
```

Simple as that.

Import a bigger dataset into postgreSQL

Are you bored with our 22-row `zoo` dataset? Me too! It's time to go a bit bigger! Let's import the 7.000.000+ row air-delays dataset we have used before (https://data36.com/data-coding-101-introduction-bash/) in the bash tutorials! Follow these steps to have it imported into your SQL database!

UPDATE: I've realized that this might be a bit complex – not difficult, just complex – so I put all the instructions into a short video too! Click here or read below:

Note: we will work in bash. If you haven't done my bash tutorial series (https://data36.com/learn-data-analytics-bash-scratch/) yet, I highly recommend doing at least the first episode (https://data36.com/data-coding-101-introduction-bash/), but if you don't want to, it's also okay to simply follow my lead step by step below.

1. Open Terminal and login (`ssh`) to your data server!
2. Download the `flight_delays` data!
   ~~`wget http://stat-computing.org/dataexpo/2009/2007.csv.bz2`~~
   ~~*Note: If you have this already, skip forward to 5.*~~
   As of 29 December 2019, I realized that the dataset has been removed from its original place. Now, you can download the dataset using this code instead:
   `wget 46.101.230.157/sql_tutorial/2007.csv.bz2`
3. Set up `dtrx`! That's a command line tool for unzipping stuff!
   (Note: this might have been already set up; if so, skip this step!)
   `sudo apt-get install dtrx`
4. Unzip the .csv file!
   `dtrx 2007.csv.bz2`
   Note: It will take around ~60 seconds to process the whole file, so

don't worry, your Terminal is not freezing, it just needs some time.

5. Format your data!
   ```
   cat 2007.csv |cut -d',' -f1,2,3,4,5,7,10,11,14,15,16,17,18,19 | grep -v ',NA' > sql_ready.csv
   ```
6. Now we have to give permission to our postgreSQL user to create tables and load data into them. This will need multiple steps. Here's a gif first *(note: my username is* `dataguy` *– yours might be something else)*.

   First `sudo` to the user called `postgres`:
   ```
   sudo -u postgres -i
   ```

   Then start postgreSQL:
   ```
   psql
   ```

   The prompt will change to this: `postgres=#!`

   Type:
   ```
   ALTER USER [your_user_name] WITH SUPERUSER;
   ```

   This turns your original user into a super user.
   Exiting from postgreSQL. Type:
   ```
   \q
   ```

Then exit from the user called
`postgres`:

```
exit
```

Finally access your original user's postgreSQL database from the command line:

```
psql -d postgres
```

Okay, this was the hard part…

7. Now all you need to do is create the table by simply copy-pasting these lines into your terminal:

```
CREATE TABLE flight_delays (
  year INTEGER,
  month INTEGER,
  dayofmonth INTEGER,
  dayofweek INTEGER,
  deptime INTEGER,
  arrtime INTEGER,
  flightnum INTEGER,
  tailnum VARCHAR,
  airtime INTEGER,
  arrdelay INTEGER,
  depdelay INTEGER,
  origin VARCHAR,
  dest VARCHAR,
  distance INTEGER);
```

8. And finally, copy the data from the .csv file you have just downloaded!

```
COPY flight_delays FROM
'/home/tomi/sql_ready.csv'
DELIMITER ',' CSV HEADER;
```

Note: make sure that you type your user name where I've typed `tomi` or `dataguy` (which are my user names…)

9. Go back to SQL Workbench make a simple `SELECT` statement... but make sure that you use the `LIMIT` clause, too. Why? Because now you have over 7.000.000 rows of data. PostgreSQL can handle it easily, sure, but your computer might be frozen if you try print all that data on your screen.

   So try something like this first:
   ```
   SELECT * FROM
   flight_delays LIMIT 10;
   ```

Test yourself #1

Here's assignment number one:

How many flights did the plane with the tail-number `N253WN` take on 23 April, 2007?

(Hint: as we haven't learned the `COUNT` function yet, it's enough if you print all the flights with the given conditions and count the lines for yourself.)

.

.

.

Done? Here's my solution:

```
SELECT
  *
FROM
  flight_delays
WHERE month = 4
  AND year = 2007
  AND dayofmonth = 23
  AND tailnum = 'N253WN';
```

The only trick is that the day, the month and the year are in different columns, so you have to use multiple `AND` operators to filter simultaneously for all the conditions.

(Note: since the `month`, `year` and `dayofmonth` fields are integers, you don't need apostrophes. But you need it for the `tailnum` field, because it contains letters too. I'll get back to this topic later.)

Test yourself #2

Select all the rows with these conditions:

- the flight was in April 2007
- it was an even weekday (2nd, 4th or 6th day of the week)
- the flight distance was less than 50 miles
- either the departure or the arrival delay was less than 0 (means the plane took off or landed earlier than planned)

(Hint: you will have to use `OR` *a*

operators together!)

.

.

.

And my solution is:

```
SELECT
  *
FROM
  flight_delays
WHERE month = 4
  AND dayofweek IN (2,4,6)
  AND distance < 50
  AND (arrdelay < 0
  OR depdelay < 0);
```

Can you see the trick here? In SQL WHERE clauses `AND` is stronger by default than `OR` – thus if you don't use the parentheses, the execution order of the different filters would be like this:

```
WHERE (month = 4
  AND dayofweek IN (2,4,6)
  AND distance < 50
  AND arrdelay < 0)
  OR (depdelay < 0);
```

And that would have returned wrong results! Take-away is: always use parentheses when adding multiple conditions in your `WHERE` clause — so SQL can priortize the right filters.

Conclusion

Knowing, understanding and u[illegible] SQL WHERE clauses with the right operators is crucial. In this article you have learned the most important parts of it.

Now – having a good base knowledge – it's time to continue with the next important SQL topic! In my next article (https://data36.com/sql-functions-beginners-tutorial-ep3/), I'll introduce the basic SQL functions (`MAX`, `MIN`, `SUM`, `COUNT`) and I'll show you some more important clauses (`ORDER BY`, `GROUP BY`, `DISTINCT`).

You can continue by clicking here (https://data36.com/sql-functions-beginners-tutorial-ep3/).

- 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/) (It's free!)
- Also check out my 6-week online course: The Junior Data Scientist's First Month video course. (https://data36.com/jds/)

Cheers,
Tomi Mester

Sources:

- http://stat-computing.org/dataexpo/

(http://stat-computing.org/dataexpo/) — shout out for the awesome dataset!