Week 7: Trying Out New SQL Functions

Natalia DuBon

I. Abstract

This week's assignment calls for us to test out new query functions that we may have not used or explored before. After researching the links provided as well as doing my own research, I have gathered the following queries that allow me to truly check out the scope of the functions available in SQL and seeing which ones I can apply for future or past projects.

II. Query Explorations

For the first query, I wanted to explore a new date/time function to which I've never used before or had knowledge of. More specifically, I wanted to use a function that could have potentially helped me further in previous projects. Considering last week's project in which I used machine learning to create a linear regression model, I would have wished to do a daily time interval rather than a weekly one. Therefore I explored the following query which gave me such result (**Query A**):

Query A

```
SELECT DAYOFYEAR(cout) AS Day,

SUM(CASE

WHEN title LIKE '%data science%' Then 1

ELSE 0 END) AS 'Data Science'

FROM spl_2016.inraw
```

```
WHERE

YEAR(cout) = 2019

GROUP BY DAYOFYEAR(cout)

ORDER BY DAYOFYEAR(cout) ;
```

CSV:

Week 7 Query A - Week7_QueryA.pdf

The output is a table that is now divided by day of the year than weekly as I had did last week. This would have been helpful because I would have had even more data points and could have in turn (potentially) created a better model, or at least a new model worth exploring more.

Last week I was able to do a statistical analysis of my dataset using R, but this week I'd like to find a way to apply some of these methods in SQL to the best of my ability. For this query, I am using the average function, the max function, the min function, and finally the standard deviation function for the above data set. This is similar to the boxplot I had created last week in R, in that it provided information on the mean, the distance between the maximum and minimum, and any outliers (Query B).

Query B

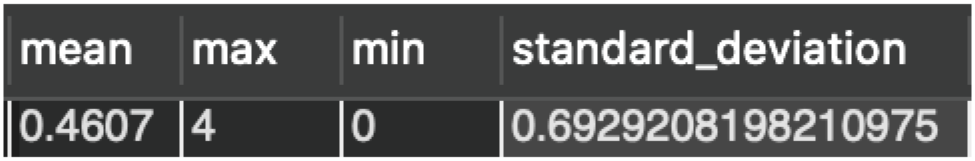
```
select

avg(counts) as mean,

max(counts) as max,
min(counts) as min,
stddev_samp(counts) as sd
from(
SELECT DAYOFYEAR(cout) AS Day,
SUM(CASE
WHEN title LIKE '%data science%' Then 1
ELSE 0 END) AS 'counts'
FROM spl_2016.inraw x
WHERE
YEAR(cout) = 2019
GROUP BY DAYOFYEAR(cout)
ORDER BY DAYOFYEAR(cout)
) y
```

CSV:

🗎 Week7_QueryB - Week7_QueryB.pdf



mean	max	min	standard_deviation
0.4607	4	0	0.6929208198210975

From the output, we can see that the mean is 0.4607, the maximum is 4, the minimum is 0, and the standard deviation is 0.6929208198210975. This would be the same output had I imported the data set into R and run a similar statistical analysis. It’s encouraging to see that the basics of statistics can also be made here within SQL.

For the next query, I decided to try out the substring function as well as using multiple case functions. I had to choose a topic, so I chose three of Michael Jackson’s albums during a three year span (2008-2010). Since Michael Jackson passed during the year 2009, I just chose the year before and after (**Query C**).

Query C

```
SELECT

bibNumber, title, callNumber,

COUNT(bibNumber) AS Counts,

SUM(CASE

WHEN (YEAR(cout) = 2008) THEN 1

ELSE 0

END) AS '2008”',

SUM(CASE

WHEN (YEAR(cout) = 2009) THEN 1

ELSE 0

END) AS '2009',

SUM(CASE

WHEN (YEAR(cout) = 2010) THEN 1

ELSE 0

END) AS '2010'

FROM spl_2016.inraw

WHERE
```

SUBSTRING(itemType, 3, 4) = 'cd' and

title = 'Thriller' or

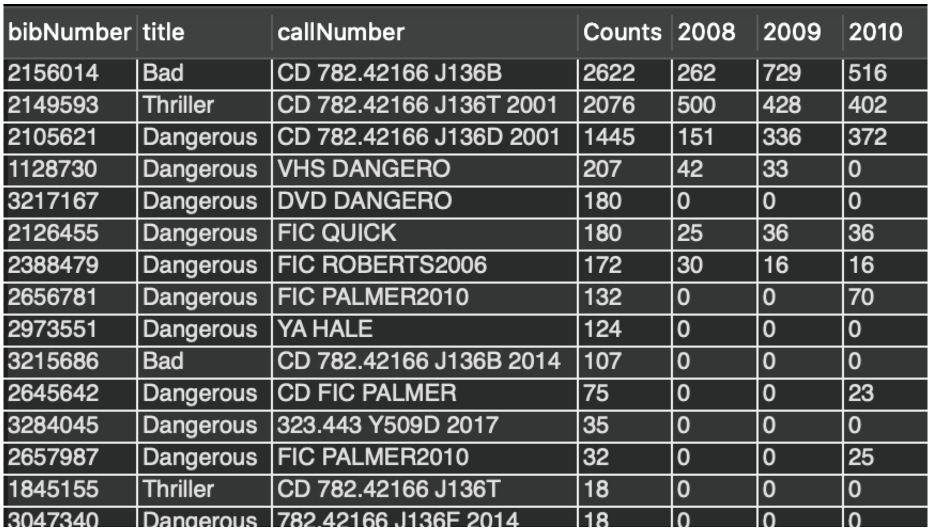
title = 'Bad' or

title = 'Dangerous'

GROUP BY bibNumber , title, callNumber ORDER BY Counts DESC

<u>CSV:</u>

- Week7_QueryC - Week7_QueryC.pdf



bibNumber	title	callNumber	Counts	2008	2009	2010
2156014	Bad	CD 782.42166 J136B	2622	262	729	516
2149593	Thriller	CD 782.42166 J136T 2001	2076	500	428	402
2105621	Dangerous	CD 782.42166 J136D 2001	1445	151	336	372
1128730	Dangerous	VHS DANGERO	207	42	33	0
3217167	Dangerous	DVD DANGERO	180	0	0	0
2126455	Dangerous	FIC QUICK	180	25	36	36
2388479	Dangerous	FIC ROBERTS2006	172	30	16	16
2656781	Dangerous	FIC PALMER2010	132	0	0	70
2973551	Dangerous	YA HALE	124	0	0	0
3215686	Bad	CD 782.42166 J136B 2014	107	0	0	0
2645642	Dangerous	CD FIC PALMER	75	0	0	23
3284045	Dangerous	323.443 Y509D 2017	35	0	0	0
2657987	Dangerous	FIC PALMER2010	32	0	0	25
1845155	Thriller	CD 782.42166 J136T	18	0	0	0
3047340	Dangerous	782.42166 J136F 2014	18	0	0	0

From the output, we can see that for several years, 2009 is the maximum for each type of item per title, but not always the case. Seeing the data laid out this way is useful if later I wanted to dive deeper into this search. Using multiple case functions ultimately made this dataset much more useful if we wanted to see a progress in growth per year.

For the final query, I decided to try out an entirely new function that I could not find from a basic SQL handbook. I ran into a function called REGEXP

MySQL allows you to match patterns right in the SQL statements by using the REGEXP operator. This statement performs a pattern match of a string_column against a pattern.If a value in the string_column matches the pattern, the expression in the WHERE clause returns true, otherwise it returns false.

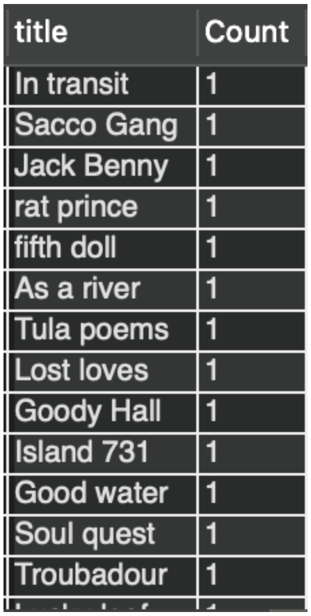
If either string_column or pattern is NULL, the result is NULL. To find the product whose name contains exactly 10 characters, you use ‘^' and ‘$ to match the beginning and end of the product name, and repeat {10} times of any character ‘.' in between as shown in the following query (**Query D**):

Query D

```
SELECT
DISTINCT title,
count(cout) as Count
FROM spl_2016.inraw
WHERE
title REGEXP '^.{10}$'
AND year(cout)= 2021
GROUP BY title
ORDER BY count(cout)
LIMIT 100;
```

CSV:

🗎 **Week7_QueryD - Week7_QueryD (1).pdf**



title	Count
In transit	1
Sacco Gang	1
Jack Benny	1
rat prince	1
fifth doll	1
As a river	1
Tula poems	1
Lost loves	1
Goody Hall	1
Island 731	1
Good water	1
Soul quest	1
Troubadour	1
[illegible]	[illegible]

While using this specific operand isn't something I see myself using in comparison to my previous projects, I thought it was interesting enough to try out and initially a bit challenging to understand. The purpose of this query was mainly to try something that perhaps none of us have seen (at least to my knowledge).

III. Resources

https://www.mysqltutorial.org/mysql-regular-expression-regexp.aspx

https://dev.mysql.com/doc/refman/8.0/en/explain.html

https://www.w3schools.com/mysql/mysql_examples.asp