Article
Muhammad Waseem · Jan 24, 2023 4m read

5 useful SQL functions to take your SQL skills to the next level

Hi Community,

In this article, I listed 5 useful SQL functions with explanations and query examples

These 5 functions are

- COALESCE
- RANK
- DENSERANK
- ROWNUMBER
- Function to Get Running Totals

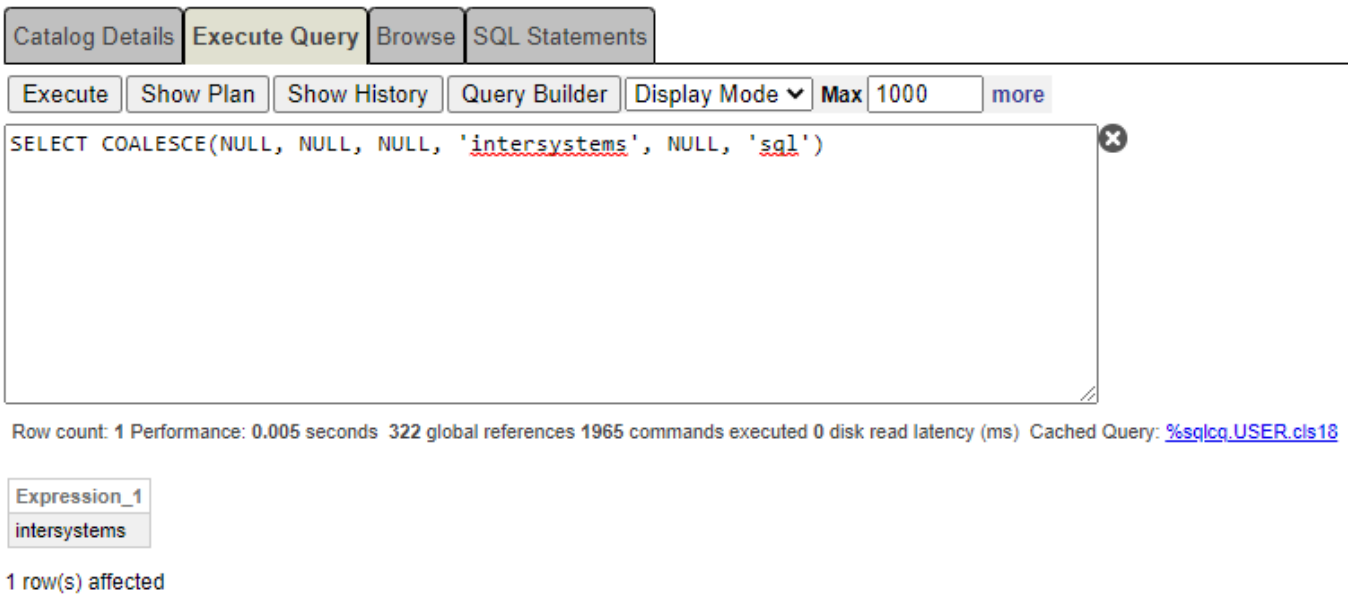
So Let us start with COALESCE function

#COALESCE

The COALESCE function evaluates a list of expressions in left-to-right order and returns the value of the first non-NULL expression. If all expressions evaluate to NULL, NULL is returned.

Following statement will return first not null value which is 'intersystems'

```
SELECT COALESCE(NULL, NULL, NULL,'intersystems', NULL,'sql')
```



Let us create below table for further example

```
CREATE TABLE EXPENSES(
```

```
    TDATE     DATE NOT NULL,
    EXPENSE1   NUMBER NULL,
    EXPENSE2   NUMBER NULL,
    EXPENSE3   NUMBER NULL,
    TTYPE  CHAR(30) NULL)
```

Now let us insert some dummy data to test our function

```
INSERT INTO sqluser.expenses (tdate, expense1,expense2,expense3,ttype )
 SELECT {d'2023-01-01'}, 500,400,NULL,'Present'
 UNION ALL
 SELECT {d'2023-01-01'}, NULL,50,30,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-01'}, NULL,NULL,30,'Clothes'
 UNION ALL
 SELECT {d'2023-01-02'}, NULL,50,30 ,'Present'
 UNION ALL
 SELECT {d'2023-01-02'}, 300,500,NULL,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-02'}, NULL,400,NULL,'Clothes'
 UNION ALL
 SELECT {d'2023-01-03'}, NULL,NULL,350 ,'Present'
 UNION ALL
 SELECT {d'2023-01-03'}, 500,NULL,NULL,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-04'}, 200,100,NULL,'Clothes'
 UNION ALL
 SELECT {d'2023-01-06'}, NULL,NULL,100,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-06'}, NULL,100,NULL,'Clothes'
```

Catalog Details | Execute Query | Browse | SQL Statements

Execute | Show Plan | Show History | Query Builder | Display Mode | Max 1000 | more

```
INSERT INTO sqluser.expenses (tdate, expense1,expense2,expense3,ttype )
 SELECT {d'2023-01-01'}, 500,400,300,'Present'
 UNION ALL
 SELECT {d'2023-01-01'}, NULL,50,230,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-01'}, NULL,NULL,330,'Clothes'
 UNION ALL
 SELECT {d'2023-01-02'}, NULL,50,430 ,'Present'
 UNION ALL
 SELECT {d'2023-01-02'}, 300,500,200,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-02'}, NULL,400,250,'Clothes'
 UNION ALL
 SELECT {d'2023-01-03'}, NULL,NULL,350 ,'Present'
 UNION ALL
 SELECT {d'2023-01-03'}, 500,NULL,100,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-04'}, 200,100,140,'Clothes'
 UNION ALL
 SELECT {d'2023-01-06'}, NULL,NULL,240,'SuperMarket'
 UNION ALL
 SELECT {d'2023-01-06'}, NULL,100,230,'Clothes'
```

Row count: **11** Performance: **0.006** seconds **355** global references **17144** commands executed **0** disk read latency (ms) Cached Query: %sqlcq.USER.cls59

11 row(s) affected

Select the data

Catalog Details | Execute Query | Browse | SQL Statements

Execute | Show Plan | Show History | Query Builder | Display Mode | Max 1000 | more

```
SELECT * FROM sqluser.expenses
```

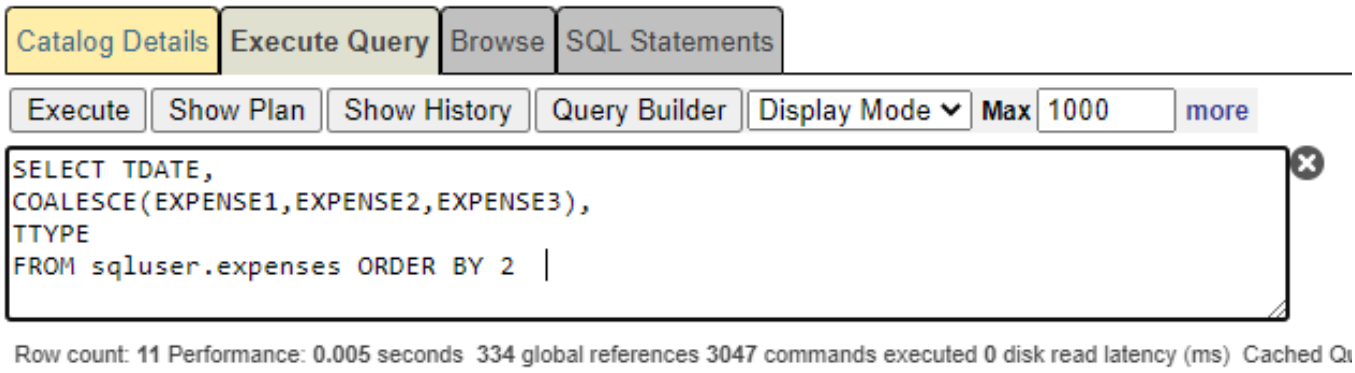
Row count: **11** Performance: **0.005** seconds **334** global references **2174** commands executed **0** disk read latency (ms) Cache

TDATE	EXPENSE1	EXPENSE2	EXPENSE3	TTYPE
01/01/2023	500	400	300	Present
01/01/2023		50	230	SuperMarket
01/01/2023			330	Clothes
01/02/2023		50	430	Present
01/02/2023	300	500	200	SuperMarket
01/02/2023		400	250	Clothes
01/03/2023			350	Present
01/03/2023	500		100	SuperMarket
01/04/2023	200	100	140	Clothes
01/06/2023			240	SuperMarket
01/06/2023		100	230	Clothes

11 row(s) affected

Now by using COALESCE function we will retrieve first not NULL value from expense1,expense2 and expense 3 columns

```
SELECT TDATE,
COALESCE(EXPENSE1,EXPENSE2,EXPENSE3),
TTYPE
FROM sqluser.expenses ORDER BY 2
```



TDATE	Expression_2	TTYPE
01/01/2023	50	SuperMarket
01/02/2023	50	Present
01/06/2023	100	Clothes
01/04/2023	200	Clothes
01/06/2023	240	SuperMarket
01/02/2023	300	SuperMarket
01/01/2023	330	Clothes
01/03/2023	350	Present
01/02/2023	400	Clothes
01/01/2023	500	Present
01/03/2023	500	SuperMarket

11 row(s) affected

#RANK vs DENSERANK vs ROWNUMBER functions

- RANK()— assigns a ranking integer to each row within the same window frame, starting with 1. Ranking integers can include duplicate values if multiple rows contain the same value for the window function field.
- ROWNUMBER() — assigns a unique sequential integer to each row within the same window frame, starting with 1. If multiple rows contain the same value for the window function field, each row is assigned a unique sequential integer.
- DENSERANK() leaves no gaps after a duplicate rank.

In SQL, there ' s several ways that you can assign a rank to a row, which we ' ll dive into with an example. Consider once again the same example as before, but now we want to know what is the highest expenses.

We want to know where do I spend the most money. There are different ways to do it. We can use all ROWNUMBER() , RANK() and DENSERANK() . We will order the previous table using all three functions and see what are the main differences between them using the following query:

Below is our query:

Catalog Details | Execute Query | Browse | SQL Statements

Execute | Show Plan | Show History | Query Builder | Display Mode | Max 1000 | more

```
SELECT
     TDATE,EXPENSE3,TTYPE,
     ROW_NUMBER() OVER (ORDER BY EXPENSE3 DESC) AS Row_Number,
     RANK()       OVER (ORDER BY EXPENSE3 DESC) AS RANK,
     DENSE_RANK() OVER (ORDER BY EXPENSE3 DESC) AS DENSE_RANK
FROM sqluser.expenses
ORDER BY ROW_NUMBER
```

Row count: 11 Performance: 0.005 seconds 334 global references 6251 commands executed 0 disk read latency (ms) Cached Query:

TDATE	EXPENSE3	TTYPE	Row_Number	RANK	DENSE_RANK
01/02/2023	430	Present	1	1	1
01/03/2023	350	Present	2	2	2
01/01/2023	330	Clothes	3	3	3
01/01/2023	300	Present	4	4	4
01/02/2023	250	Clothes	5	5	5
01/06/2023	240	SuperMarket	6	6	6
01/01/2023	230	SuperMarket	7	7	7
01/06/2023	230	Clothes	8	7	7
01/02/2023	200	SuperMarket	9	9	8
01/04/2023	140	Clothes	10	10	9
01/03/2023	100	SuperMarket	11	11	10

11 row(s) affected

The main difference between al three functions is the way they deal with ties. We will further deep-dive their differences:

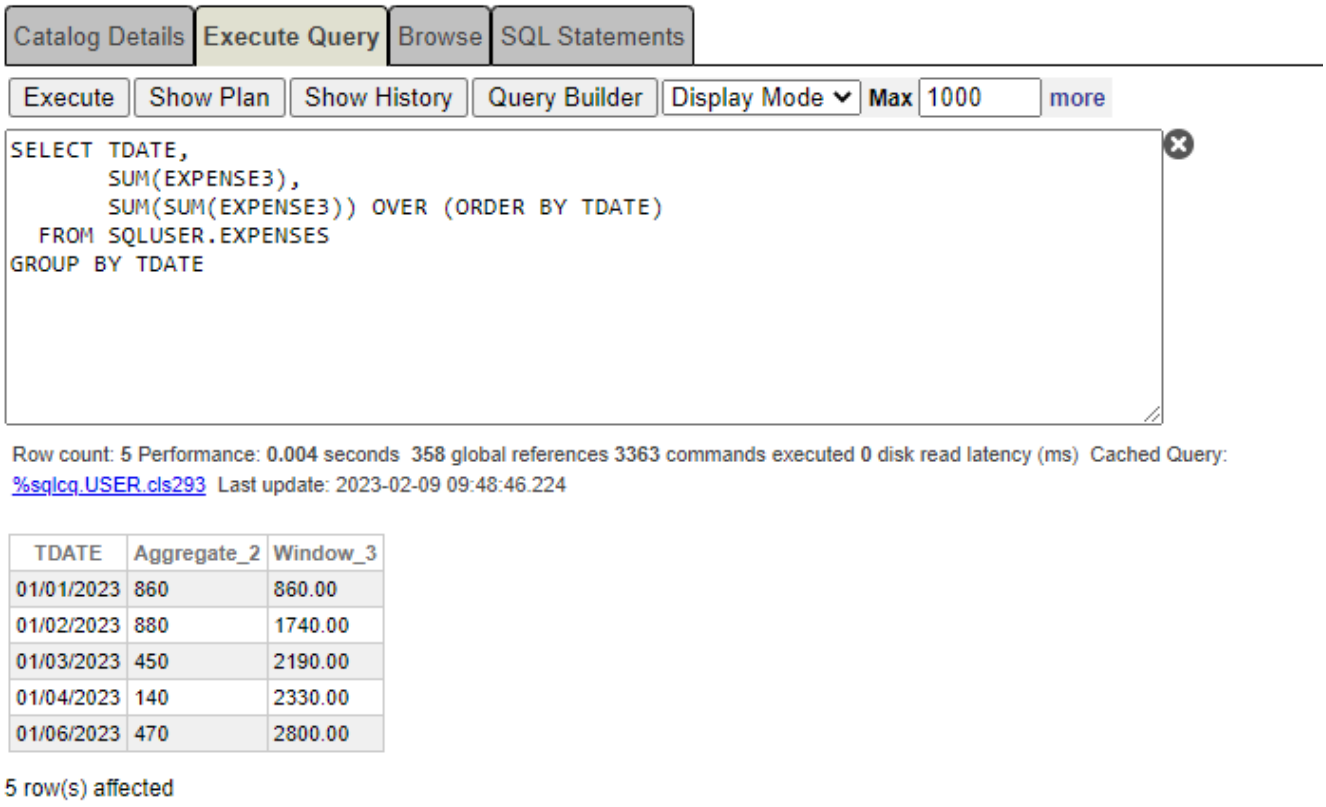
- ROWNUMBER()returns a unique number for each row starting at 1. When there are ties, it arbitrarily assigns a number if a second criteria is not defined.
- RANK()returns a unique number for each row starting at 1, except for when there are ties, then it will assign the same number. As well, a gap will follow a duplicate rank.
- DENSERANK() leaves no gaps after a duplicate rank.

#Calculating Running Totals

The running total is probably one of the most useful window functions especially when you want to visualize growth. Using a window function with SUM(), we can calculate a cumulative aggregation.

To do so, we just need to sum a variable using the aggregator SUM() but order this function using a TDATE column.

You can observe the corresponding query as follows:



As you can observe in the table above, now we have the accumulated aggregation of the amount of money spent as the date passes by.

Conclusion

SQL is great. Functions used above might be useful when dealing data analysis, data science, and any other data-related field.

This is why you should care to keep improving your SQL skills.

Thanks

#Beginner #Best Practices #SQL #Tips & Tricks #Caché