When and how to use the SQL PARTITION BY clause

Written By Rajendra Gupta (https://blog.quest.com/author/rajendragupta/)
July 27, 2021

In this article, we will explore when and how to use the SQL PARTITION BY clause and compare it to using the GROUP BY clause.

Understanding the Window function

Database users use aggregate functions such as MAX(), MIN(), AVERAGE() and COUNT() for performing data analysis. These functions operate on an entire table and return single aggregated data using the GROUP BY clause. Sometimes, we require aggregated values over a small set of rows. In this case, the Window function combined with the aggregate function helps achieve the desired output. The Window function uses the OVER() clause, and it can include the following functions:

- **Partition By:** This divides the rows or query result set into small partitions.
- **Order By:** This arranges the rows in ascending or descending order for the partition window. The default order is ascending.
- **Row or Range:** You can further limit the rows in a partition by specifying the start and endpoints.

In this article, we will focus on exploring the SQL PARTITION BY clause.

Preparing sample data

Suppose we have a table [SalesLT].[Orders] that stores customer order details. It has a column [City] that specifies the customer city of where the order was placed.

```
CREATE TABLE [SalesLT].[Orders]
(
orderid INT,
orderdate DATE,
customerName VARCHAR(100),
City VARCHAR(50),
amount MONEY
)
INSERT INTO [SalesLT].[Orders]
SELECT 1,'01/01/2021','Mohan Gupta','Alwar',10000
UNION ALL
SELECT 2,'02/04/2021','Lucky Ali','Kota',20000
UNION ALL
SELECT 3,'03/02/2021','Raj Kumar','Jaipur',5000
UNION ALL
SELECT 4,'04/02/2021','Jyoti Kumari','Jaipur',15000
UNION ALL
SELECT 5,'05/03/2021','Rahul Gupta','Jaipur',7000
UNION ALL
SELECT 6,'06/04/2021','Mohan Kumar','Alwar',25000
UNION ALL
SELECT 7,'07/02/2021','Kashish Agarwal','Alwar',15000
UNION ALL
SELECT 8,'08/03/2021','Nagar Singh','Kota',2000
UNION ALL
SELECT 9,'09/04/2021','Anil KG','Alwar',1000
Go
```

Let’ s say we want to know the total orders value by location (City). For this purpose, we use the SUM() and GROUP BY function as shown below.

```
SELECT City AS CustomerCity
,sum(amount) AS totalamount FROM [SalesLT].[Orders]
GROUP BY city
ORDER BY city
```

```
SELECT City AS CustomerCity
,sum(amount) AS totalamount FROM [SalesLT].[Orders]
GROUP BY city
ORDER BY city
```

100 %

Results | Messages

	CustomerCity	totalamount
1	Alwar	51000.00
2	Jaipur	27000.00
3	Kota	22000.00

In the result set, we cannot use the non-aggregated columns in the SELECT statement (https://blog.quest.com/how-to-use-update-from-select-in-sql-server/). For example, we cannot display [CustomerName] in the output because it is not included in the GROUP BY clause.

SQL Server gives the following error message if you try to use the non-aggregated column in the column list.

```
SELECT City AS CustomerCity, CustomerName
,sum(amount) AS totalamount FROM [SalesLT].[Orders]
GROUP BY city
ORDER BY city
```

200 %

Messages

```
Msg 8120, Level 16, State 1, Line 3
Column 'SalesLT.Orders.customerName' is invalid in the select list because
it is not contained in either an aggregate function or the GROUP BY clause.
```

```
SELECT City AS CustomerCity, CustomerName,amount,
SUM(amount) OVER(PARTITION BY city) TotalOrderAmount
FROM [SalesLT].[Orders]
```

As shown below, the PARTITION BY clause creates a smaller window (set of data rows), performs the aggregation and displays it. You can also view non-aggregated columns as well in this output.

```
SELECT City AS CustomerCity, CustomerName,amount,
SUM(amount) OVER(PARTITION BY city) TotalOrderAmount
FROM [SalesLT].[Orders]
```

200 %

Results | Messages

	CustomerCity	CustomerName	amount	TotalOrderAmount
1	Alwar	Mohan Gupta	10000.00	51000.00
2	Alwar	Mohan Kumar	25000.00	51000.00
3	Alwar	Kashish Agarwal	15000.00	51000.00
4	Alwar	Anil KG	1000.00	51000.00
5	Jaipur	Raj Kumar	5000.00	27000.00
6	Jaipur	Jyoti Kumari	15000.00	27000.00
7	Jaipur	Rahul Gupta	7000.00	27000.00
8	Kota	Lucky Ali	20000.00	22000.00
9	Kota	Nagar Singh	2000.00	22000.00

Similarly, you can use functions AVG(), MIN(), MAX() to calculate the average, minimum and maximum amount from the rows in a window.

```
SELECT City AS CustomerCity, CustomerName,amount,
SUM(amount) OVER(PARTITION BY city) TotalOrderAmount,
Avg(amount) OVER(PARTITION BY city) AvgOrderAmount,
Min(amount) OVER(PARTITION BY city) MinOrderAmount,
MAX(amount) OVER(PARTITION BY city) MaxOrderAmount
FROM [SalesLT].[Orders]
```

```
SELECT City AS CustomerCity, CustomerName,amount,
SUM(amount) OVER(PARTITION BY city) TotalOrderAmount,
Avg(amount) OVER(PARTITION BY city) AvgOrderAmount,
Min(amount) OVER(PARTITION BY city) MinOrderAmount,
MAX(amount) OVER(PARTITION BY city) MaxOrderAmount
FROM [SalesLT].[Orders]
```

100 %

Results | Messages

	CustomerCity	CustomerName	amount	TotalOrderAmount	AvgOrderAmount	MinOrderAmount	MaxOrderAmount
1	Alwar	Mohan Gupta	10000.00	51000.00	12750.00	1000.00	25000.00
2	Alwar	Mohan Kumar	25000.00	51000.00	12750.00	1000.00	25000.00
3	Alwar	Kashish Agarwal	15000.00	51000.00	12750.00	1000.00	25000.00
4	Alwar	Anil KG	1000.00	51000.00	12750.00	1000.00	25000.00
5	Jaipur	Raj Kumar	5000.00	27000.00	9000.00	5000.00	15000.00
6	Jaipur	Jyoti Kumari	15000.00	27000.00	9000.00	5000.00	15000.00
7	Jaipur	Rahul Gupta	7000.00	27000.00	9000.00	5000.00	15000.00
8	Kota	Lucky Ali	20000.00	22000.00	11000.00	2000.00	20000.00
9	Kota	Nagar Singh	2000.00	22000.00	11000.00	2000.00	20000.00

Using the SQL PARTITION BY clause with the ROW_NUMBER() function

Previously, we got the aggregated values in a window using the PARTITION BY clause. Suppose that instead of the total, we require the cumulative total in a partition.

A cumulative total works in the following ways.

Row	Cumulative total
1	Rank 1+ 2
2	Rank 2+3
3	Rank 3+4

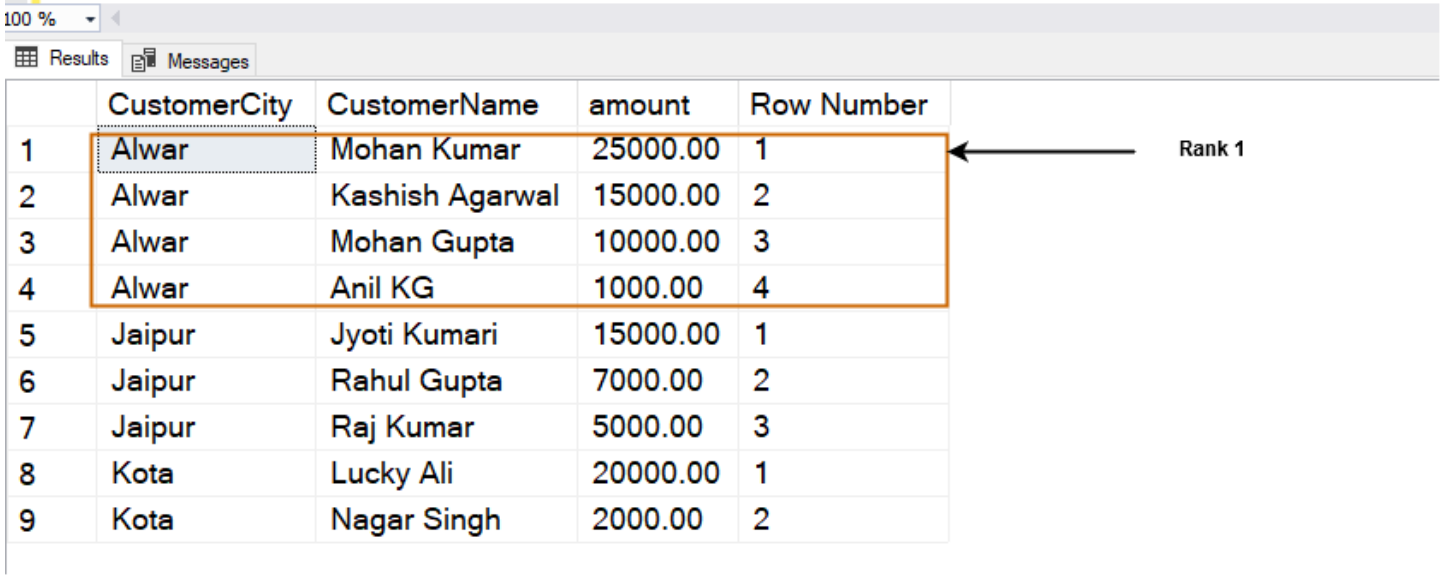
The row rank is calculated using the function ROW_NUMBER(). Let’ s first use this function and view the row ranks.

- The ROW_NUMBER() function uses the OVER and PARTITION BY clause and sorts results in ascending or descending order. It starts ranking rows from 1 per the sorting order.

```
SELECT City AS CustomerCity, CustomerName,amount,
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number]
FROM [SalesLT].[Orders]
```

For example, in the [Alwar] city, the row with the highest amount (25000.00) is in row 1. As shown below, it ranks rows in the window specified by the PARTITION BY clause. For example, we have three different cities [Alwar], [Jaipur] and [Kota], and each window (city) gets its row ranks.

```
SELECT City AS CustomerCity, CustomerName,amount,
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number]
FROM [SalesLT].[Orders]
```



	CustomerCity	CustomerName	amount	Row Number
1	Alwar	Mohan Kumar	25000.00	1
2	Alwar	Kashish Agarwal	15000.00	2
3	Alwar	Mohan Gupta	10000.00	3
4	Alwar	Anil KG	1000.00	4
5	Jaipur	Jyoti Kumari	15000.00	1
6	Jaipur	Rahul Gupta	7000.00	2
7	Jaipur	Raj Kumar	5000.00	3
8	Kota	Lucky Ali	20000.00	1
9	Kota	Nagar Singh	2000.00	2

To calculate the cumulative total, we use the following arguments.

- CURRENT ROW: It specifies the starting and ending point in the specified range.
- 1 following: It specifies the number of rows (1) to follow from the current row.

```
SELECT City AS CustomerCity, CustomerName,amount,
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number],
SUM(amount) OVER(PARTITION BY city ORDER BY amount DESC ROWS BETWEEN
CURRENT ROW AND 1 FOLLOWING) AS CumulativeSUM
FROM [SalesLT].[Orders]
```

The following image shows that you get a cumulative total instead of an overall total in a window specified by the PARTITION BY clause.

```
SELECT City AS CustomerCity, CustomerName,amount,
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number],
 SUM(amount) OVER(PARTITION BY city ORDER BY amount DESC ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS CumulativeSUM
FROM [SalesLT].[Orders]
```

100 %

Results | Messages

	CustomerCity	CustomerName	amount	Row Number	CumulativeSUM
1	Alwar	Mohan Kumar	25000.00	1	40000.00
2	Alwar	Kashish Agarwal	15000.00	2	25000.00
3	Alwar	Mohan Gupta	10000.00	3	11000.00
4	Alwar	Anil KG	1000.00	4	1000.00
5	Jaipur	Jyoti Kumari	15000.00	1	22000.00
6	Jaipur	Rahul Gupta	7000.00	2	12000.00
7	Jaipur	Raj Kumar	5000.00	3	5000.00
8	Kota	Lucky Ali	20000.00	1	22000.00
9	Kota	Nagar Singh	2000.00	2	2000.00

Sum of Row Number 1 and 2

Sum of Row Number 2 and 3

If we use **ROWS UNBOUNDED PRECEDING** in the SQL PARTITION BY clause, it calculates the cumulative total in the following way. It uses the current rows along with the rows having the highest values in the specified window.

Row	Cumulative total
1	Rank 1
2	Rank 1+2
3	Rank 1+2+3

```
SELECT City AS CustomerCity, CustomerName,amount,
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number],
SUM(amount) OVER(PARTITION BY city ORDER BY amount DESC
ROWS UNBOUNDED PRECEDING) AS CumulativeSUM
FROM [SalesLT].[Orders]
```

```
SELECT City AS CustomerCity, CustomerName,amount,
ROW_NUMBER() OVER(PARTITION BY city ORDER BY amount DESC) AS [Row Number],
 SUM(amount) OVER(PARTITION BY city ORDER BY amount DESC  ROWS UNBOUNDED PRECEDING) AS CumulativeSUM
FROM [SalesLT].[Orders]
```

100 %

Results | Messages

	CustomerCity	CustomerName	amount	Row Number	CumulativeSUM
1	Alwar	Mohan Kumar	25000.00	1	25000.00
2	Alwar	Kashish Agarwal	15000.00	2	40000.00
3	Alwar	Mohan Gupta	10000.00	3	50000.00
4	Alwar	Anil KG	1000.00	4	51000.00
5	Jaipur	Jyoti Kumari	15000.00	1	15000.00
6	Jaipur	Rahul Gupta	7000.00	2	22000.00
7	Jaipur	Raj Kumar	5000.00	3	27000.00
8	Kota	Lucky Ali	20000.00	1	20000.00
9	Kota	Nagar Singh	2000.00	2	22000.00

Row Number 1 + 2 + 3

Comparing the GROUP BY and SQL PARTITION BY clause

GROUP BY	PARTITION BY
It returns one row per group after calculating the aggregate values.	It returns all rows from the SELECT statement along with additional columns of aggregated values.
We cannot use the non-aggregated column in the SELECT statement.	We can use required columns in the SELECT statement, and it does not produce any errors for the non-aggregated column.
It requires using the HAVING clause to filter records from the SELECT statement.	The PARTITION function can have additional predicates in the WHERE clause apart from the columns used in the SELECT statement.
The GROUP BY is used in regular aggregates.	PARTITION BY is used in windowed aggregates.
We cannot use it for calculating row numbers or their ranks.	It can calculate row numbers and their ranks in the smaller window.

Putting it to use

It’ s recommended to use the SQL PARTITION BY clause while working with multiple data groups for the aggregated values in the individual group. Similarly, it can be used to view original rows with the additional column of aggregated values.

database management (https://blog.quest.com/tag/database-management/), performance tuning (https://blog.quest.com/tag/performance-tuning/)