Using aggregation functions over windows

TIME SERIES ANALYSIS IN SQL SERVER

Maham Faisal Khan
Senior Data Science Content Developer

datacamp

Ranking functions

`ROW_NUMBER()`

Unique, ascending integer value starting from 1.

`RANK()`

Ascending integer value starting from 1. Can have ties. Can skip numbers.

`DENSE_RANK()`

Ascending integer value starting from 1. Can have ties. Will not skip numbers.

RunsScored
8
7
7
6
6
3

Calculating row numbers

```
SELECT
    s.RunsScored,
    ROW_NUMBER() OVER (
      ORDER BY s.RunsScored DESC
    ) AS rn
FROM dbo.Scores s
ORDER BY
    s.RunsScored DESC;
```

RunsScored	rn
8	1
7	2
7	3
6	4
6	5
3	6

Calculating ranks and dense ranks

```
SELECT
    s.RunsScored,
    RANK() OVER (
      ORDER BY s.RunsScored DESC
    ) AS rk,
    DENSE_RANK() OVER (
      ORDER BY s.RunsScored DESC
    ) AS dr
FROM dbo.Scores s
ORDER BY
    s.RunsScored DESC;
```

RunsScored	rk	dr
8	1	1
7	2	2
7	2	2
6	4	3
6	4	3
3	6	4

Partitions

```
SELECT
    s.Team,
    s.RunsScored,
    ROW_NUMBER() OVER (
      PARTITION BY s.Team
      ORDER BY s.RunsScored DESC
    ) AS rn
FROM dbo.Scores s
ORDER BY
    s.RunsScored DESC;
```

Team	RunsScored	rn
AZ	8	1
AZ	6	2
AZ	3	3
FLA	7	1
FLA	7	2
FLA	6	3

Aggregate functions

```
SELECT
    s.Team,
    s.RunsScored,
    MAX(s.RunsScored) OVER (
      PARTITION BY s.Team
    ) AS MaxRuns
FROM dbo.Scores s
ORDER BY
    s.RunsScored DESC;
```

Team	RunsScored	MaxRuns
AZ	8	8
AZ	6	8
AZ	3	8
FLA	7	7
FLA	7	7
FLA	6	7

Aggregations with empty windows

```
SELECT
    s.Team,
    s.RunsScored,
    MAX(s.RunsScored) OVER() AS MaxRuns
FROM dbo.Scores s
ORDER BY
    s.RunsScored DESC;
```

Team	RunsScored	MaxRuns
AZ	8	8
AZ	6	8
AZ	3	8
FLA	7	8
FLA	7	8
FLA	6	8

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

Calculating running totals and moving averages

TIME SERIES ANALYSIS IN SQL SERVER

Maham Faisal Khan
Senior Data Science Content Developer

datacamp

Calculating running totals

Team	Game	RunsScored
AZ	1	8
AZ	2	6
AZ	3	3
FLA	1	7
FLA	2	7
FLA	3	6

Team	Game	RunsScored	TotalRuns
AZ	1	8	8
AZ	2	6	14
AZ	3	3	17
FLA	1	7	7
FLA	2	7	14
FLA	3	6	20

Running totals

```
SELECT
    s.Team,
    s.Game,
    s.RunsScored,
    SUM(s.RunsScored) OVER (
        PARTITION BY s.Team
        ORDER BY s.Game ASC
        RANGE BETWEEN
              UNBOUNDED PRECEDING
              AND CURRENT ROW
    ) AS TotalRuns
FROM #Scores s;
```

- `Team`, `Game`, `RunsScored` columns
- `SUM(s.RunsScored)`
- `OVER()`
- `PARTITION BY s.Team`
- `ORDER BY s.Game ASC`
- `RANGE BETWEEN`
- `UNBOUNDED PRECEDING`
- `AND CURRENT ROW`

RANGE and ROWS

`RANGE`

- Specify a range of results
- "Duplicates" processed all at once
- Only supports `UNBOUNDED` and `CURRENT ROW`

`ROWS`

- Specify number of rows to include
- "Duplicates" processed a row at a time
- Supports `UNBOUNDED`, `CURRENT ROW`, and number of rows

Calculating moving averages

```
SELECT
    s.Team,
    s.Game,
    s.RunsScored,
    AVG(s.RunsScored) OVER (
        PARTITION BY s.Team
        ORDER BY s.Game ASC
        ROWS BETWEEN 1 PRECEDING
            AND CURRENT ROW
    ) AS AvgRuns
FROM #Scores s;
```

Team	Game	RunsScored	AvgRuns
AZ	1	8	8
AZ	2	6	7
AZ	3	3	4
FLA	1	7	7
FLA	2	7	7
FLA	3	6	6

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

Working with LAG() and LEAD()

TIME SERIES ANALYSIS IN SQL SERVER

Maham Faisal Khan
Senior Data Science Content Developer

datacamp

The LAG() window function

```
SELECT
    dsr.CustomerID,
    dsr.MonthStartDate,
    LAG(dsr.NumberOfVisits) OVER (PARTITION BY dsr.CustomerID ORDER BY dsr.MonthStartDate) AS Prior,
    dsr.NumberOfVisits
FROM dbo.DaySpaRollup dsr;
```

CustomerID	MonthStartDate	Prior	NumberOfVisits
1	2018-12-01	NULL	49
1	2019-01-01	49	117
1	2019-02-01	117	104

The LEAD() window function

```
SELECT
    dsr.CustomerID,
    dsr.MonthStartDate,
    dsr.NumberOfVisits,
    LEAD(dsr.NumberOfVisits) OVER (PARTITION BY dsr.CustomerID ORDER BY dsr.MonthStartDate) AS Next
FROM dbo.DaySpaRollup dsr;
```

CustomerID	MonthStartDate	NumberOfVisits	Next
1	2018-12-01	49	117
1	2019-01-01	117	104
1	2019-02-01	104	108

Specifying number of rows back

```
SELECT
   dsr.CustomerID,
   dsr.MonthStartDate,
   LAG(dsr.NumberOfVisits, 2) OVER (PARTITION BY dsr.CustomerID ORDER BY dsr.MonthStartDate) AS Prior2,
   LAG(dsr.NumberOfVisits, 1) OVER (PARTITION BY dsr.CustomerID ORDER BY dsr.MonthStartDate) AS Prior1,
   dsr.NumberOfVisits
FROM dbo.DaySpaRollup dsr;
```

CustomerID	MonthStartDate	Prior2	Prior	NumberOfVisits
1	2018-12-01	NULL	NULL	49
1	2019-01-01	NULL	49	117
1	2019-02-01	49	117	104

Windows and filters

```
SELECT
    Date,
    LAG(Val, 1) OVER(ORDER BY DATE) AS PriorVal,
    Val
FROM t;
```

Date	PriorVal	Val
2019-01-01	NULL	3
2019-01-02	3	6
2019-01-03	6	4

```
SELECT
    Date,
    LAG(Val, 1) OVER(ORDER BY DATE) AS PriorVal,
    Val
FROM t
WHERE
    t.Date > '2019-01-02';
```

Date	PriorVal	Val
2019-01-03	**NULL**	4

Windows and filters and CTEs

```
WITH records AS (
  SELECT
      Date,
      LAG(Val, 1) OVER(ORDER BY Date) AS PriorVal,
      Val
  FROM t
)
SELECT
    r.Date,
    r.PriorVal,
    r.Val
FROM records r
WHERE
    r.Date > '2019-01-02';
```

Date	PriorVal	Val
2019-01-03	6	4

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

Finding maximum levels of overlap

TIME SERIES ANALYSIS IN SQL SERVER

Maham Faisal Khan
Senior Data Science Content Developer

datacamp

Start with some data

StartTime	EndTime	ProductsOrdered
2019-07-08 14:35:00	2019-07-08 16:01:00	13
2019-07-08 15:35:00	2019-07-08 17:01:00	13
2019-07-08 16:35:00	2019-07-08 18:01:00	17
2019-07-08 17:35:00	2019-07-08 19:01:00	15
2019-07-08 17:55:00	2019-07-08 17:57:00	1
2019-07-08 20:35:00	2019-07-08 22:01:00	13

Reasoning through the problem

StartTime	EndTime	ProductsOrdered
2019-07-08 14:35:00	**2019-07-08 16:01:00**	13
2019-07-08 15:35:00	2019-07-08 17:01:00	13
2019-07-08 16:35:00	2019-07-08 18:01:00	17
2019-07-08 17:35:00	2019-07-08 19:01:00	15
2019-07-08 17:55:00	2019-07-08 17:57:00	1
2019-07-08 20:35:00	2019-07-08 22:01:00	13

Reasoning through the problem

StartTime	EndTime	ProductsOrdered
2019-07-08 14:35:00	2019-07-08 16:01:00	13
2019-07-08 15:35:00	2019-07-08 17:01:00	13
2019-07-08 16:35:00	2019-07-08 18:01:00	17
2019-07-08 17:35:00	2019-07-08 19:01:00	15
2019-07-08 17:55:00	**2019-07-08 17:57:00**	1
2019-07-08 20:35:00	2019-07-08 22:01:00	13

Algorithm, step 1

CTE StartStopPoints:

```
SELECT
    o.StartTime AS TimeUTC,
    1 AS EntryCount,
    ROW_NUMBER() OVER (ORDER BY o.StartTime) AS StartOrdinal
FROM #Orders o
UNION ALL
SELECT
    o.EndTime AS TimeUTC,
    -1 AS EntryCount,
    NULL AS StartOrdinal
FROM #Orders o
```

Algorithm, step 1

TimeUTC	EntryCount	StartOrdinal
14:35:00	1	1
15:35:00	1	2
16:35:00	1	3
17:35:00	1	4
17:55:00	1	5
20:35:00	1	6

TimeUTC	EntryCount	StartOrdinal
16:01:00	-1	NULL
17:01:00	-1	NULL
18:01:00	-1	NULL
19:01:00	-1	NULL
17:57:00	-1	NULL
22:01:00	-1	NULL

Algorithm, step 2

CTE StartStopOrder:

```
SELECT
    s.TimeUTC,
    s.EntryCount,
    s.StartOrdinal,
    ROW_NUMBER() OVER (ORDER BY TimeUTC, StartOrdinal) AS StartOrEndOrdinal
FROM StartStopPoints s
```

Algorithm, step 2

TimeUTC	EC	SO	StartEndOrdinal
14:35:00	1	1	1
15:35:00	1	2	2
16:01:00	-1	NULL	3
16:35:00	1	3	4
17:01:00	-1	NULL	5
17:35:00	1	4	6

TimeUTC	EC	SO	StartEndOrdinal
17:55:00	1	5	7
17:57:00	-1	NULL	8
18:01:00	-1	NULL	9
19:01:00	-1	NULL	10
20:35:00	1	6	11
22:01:00	-1	NULL	12

Algorithm, step 2

TimeUTC	EC	SO	StartEndOrdinal
14:35:00	**1**	1	1
15:35:00	**1**	2	2
16:01:00	**-1**	NULL	3
16:35:00	**1**	3	4
17:01:00	**-1**	NULL	5
17:35:00	**1**	4	6

TimeUTC	EC	SO	StartEndOrdinal
17:55:00	1	5	7
17:57:00	-1	NULL	8
18:01:00	-1	NULL	9
19:01:00	-1	NULL	10
20:35:00	1	6	11
22:01:00	-1	NULL	12

Algorithm, step 3

TimeUTC	StartOrdinal	StartEndOrdinal	Calc	Result
14:35:00	1	1	(2*1) - 1	1
15:35:00	2	2	(2*2) - 2	2
16:01:00	NULL	3	NULL	NULL
16:35:00	3	4	(2*3) - 4	2
17:01:00	NULL	5	NULL	NULL
17:35:00	4	6	(2*4) - 6	2

Algorithm, step 3

```
SELECT
    MAX(2 * s.StartOrdinal - s.StartOrEndOrdinal) AS MaxConcurrentVisitors
FROM StartStopOrder s
WHERE s.EntryCount = 1;
```

MaxConcurrentVisitors
3

Let's practice!

TIME SERIES ANALYSIS IN SQL SERVER

datacamp

Wrapping up

TIME SERIES ANALYSIS IN SQL SERVER

Maham Faisal Khan
Senior Data Science Content Developer

Working with dates

- Combine `DATEADD()` and `DATEDIFF()` to round dates and times.
- Format with `CAST()` and `CONVERT()` when performance matters. `FORMAT()` is useful but slow.
- Calendar tables are a valuable asset.

Building dates

- `CAST()`, `CONVERT()`, and `PARSE()` can all turn strings into dates.
- Use `TRY_CAST()`, `TRY_CONVERT()`, and `TRY_PARSE()` for safe date conversions.
- `SWITCHOFFSET()` and `TODATETIMEOFFSET()` are useful functions for working with offsets.

Time-based aggregates

- Aggregate functions include `COUNT()`, `MIN()`, `MAX()`, and `SUM()`.
- Statistical aggregate functions include `AVG()`, `STDEV()`, `VAR()`, `STDEVP()`, and `VARP()`.
- `ROLLUP`, `CUBE`, and `GROUPING SETS` allow you to refine your aggregations.

Common (and uncommon) time series problems

- Windows work over ranking functions (`ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`, and `NTILE()`) as well as aggregate functions (including statistical functions).
- Running totals are a use of `SUM()` over a window.
- Moving averages are a use of `AVG()` over a window.
- `LAG()` and `LEAD()` let us peek backward and forward in time.
- Pivot and transform date data to calculate concurrency.

Grazie!

TIME SERIES ANALYSIS IN SQL SERVER