Window Functions in SQL

Breanna Hansen
breanna@tf3604.com
@tf3604

Breanna Hansen

breanna@tf3604.com

@tf3604.com

children.org

- 25+ Years working with SQL Server
  - Development work since 7.0
  - Administration going back to 6.5
  - Fascinated with SQL internals

www.tf3604.com/rownumber

Overview

Window functions

- Introduced in SQL Server 2005
  - Limited number of functions
  - Not all functionality included
- Vastly extended in SQL Server 2012
- Defined in ANSI standard 2003
  - Implemented by many RDBMS vendors

What are window functions?

- "In the SQL database query language, window functions allow access to data in the records right before and after the current record. A window function defines a frame or window of rows with a given length around the current row, and performs a calculation across the set of data in the window." (Wikipedia)
- May be easier to understand by example

An example:
Running total

What are window functions?

CustID	OrderDate	Price	Rtotal
1	2019-01-10	1.00	
1	2019-02-11	2.10	
1	2019-03-12	3.20	
1	2019-04-13	4.30	
1	2019-05-14	5.40	
2	2019-01-20	6.50	
2	2019-01-21	7.60	

What are window functions?

CustID	OrderDate	Price	Rtotal
1	2019-01-10	1.00	1.00
1	2019-02-11	2.10	3.10
1	2019-03-12	3.20	6.30
1	2019-04-13	4.30	10.60
1	2019-05-14	5.40	16.00
2	2019-01-20	6.50	6.50
2	2019-01-21	7.60	14.10

What are window functions?

CustID	OrderDate	Price	Rtotal
1	2019-01-10	1.00	
1	2019-02-11	2.10	
1	2019-03-12	3.20	
1	2019-04-13	4.30	
1	2019-05-14	5.40	
2	2019-01-20	6.50	
2	2019-01-21	7.60	

```
select CustId,
    sum(Price) Total
from #sales
group by CustId;
```

CustId	Total
1	16.00
2	14.10

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	←
1	2019-02-11	2.10		
1	2019-03-12	3.20		
1	2019-04-13	4.30		
1	2019-05-14	5.40		
2	2019-01-20	6.50		
2	2019-01-21	7.60		

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	←
1	2019-03-12	3.20		
1	2019-04-13	4.30		
1	2019-05-14	5.40		
2	2019-01-20	6.50		
2	2019-01-21	7.60		

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	←
1	2019-04-13	4.30		
1	2019-05-14	5.40		
2	2019-01-20	6.50		
2	2019-01-21	7.60		

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	←
1	2019-05-14	5.40		
2	2019-01-20	6.50		
2	2019-01-21	7.60		

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	←
2	2019-01-20	6.50		
2	2019-01-21	7.60		

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	
2	2019-01-20	6.50	6.50	←
2	2019-01-21	7.60		

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	
2	2019-01-20	6.50	6.50	
2	2019-01-21	7.60	14.10	←

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	
2	2019-01-20	6.50	6.50	
2	2019-01-21	7.60	14.10	←

Alternatives

- Cursor

```
declare rtcsr cursor for
select ID, CustID, Price
from #sales
order by CustID, OrderDate;

declare @lastCustID int = 1;
declare @ID int, @custId int;
declare @price numeric(7,2);
declare @rt numeric(7,2) = 0.00;

open rtcsr;
fetch next from rtcsr into @ID, @custId, @price;
while @@fetch_status = 0 begin
    if @custId != @lastCustID set @rt = 0.00;
    set @rt += @price;
    update #sales set RTotal = @rt where ID =
        @ID;
    fetch next from rtcsr into @ID, @custId,
        @price;
end
close rtcsr;
deallocate rtcsr;
```

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	
2	2019-01-20	6.50	6.50	
2	2019-01-21	7.60	14.10	←

Alternatives

- Cursor

```
declare rtcsr cursor for
select ID, CustID, Price
from #sales
order by CustID, OrderDate;

declare @lastCustID int = 1;
declare @ID int, @custId int;
declare @price numeric(7,2);
declare @rt numeric(7,2) = 0.00;

open rtcsr;

fetch next from rtcsr into @ID, @custId, @price;
while @@fetch_status = 0 begin
     if @custId != @lastCustID
     begin
          set @rt = 0.00;
          set @lastCustID = @custId;
     end
     set @rt += @price;
     update #sales set RTotal = @rt where ID =
          @ID;
     fetch next from rtcsr into @ID, @custId,
          @price;
end
close rtcsr;
deallocate rtcsr;
```

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	
2	2019-01-20	6.50	6.50	
2	2019-01-21	7.60	14.10	←

```
select s1.CustId, s1.OrderDate,
s1.Price,
(
    select sum(Price)
    from #sales s2
    where s1.CustId = s2.CustId
    and s2.OrderDate <=
        s1.OrderDate
) as RTotal
from #sales s1
```

Alternatives

- ~~Cursor~~
- Triangle join

What are window functions?

CustID	OrderDate	Price	Rtotal	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.10	3.10	
1	2019-03-12	3.20	6.30	
1	2019-04-13	4.30	10.60	
1	2019-05-14	5.40	16.00	
2	2019-01-20	6.50	6.50	
2	2019-01-21	7.60	14.10	←

```
select CustID,
    OrderDate,
    Price,
    sum(Price)
        over (partition by CustId
        order by OrderDate)
        as RTotal
from #sales;
```

Alternatives

- ~~Cursor~~
- ~~Triangle join~~
- Window function

Another example:

Rolling average

(most recent 3 records)

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	←
1	2019-02-11	2.50		
1	2019-03-12	3.00		
1	2019-04-13	9.99		
1	2019-05-14	12.22		
1	2019-06-15	1.86		
1	2019-07-16	7.59		
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	←
1	2019-03-12	3.00		
1	2019-04-13	9.99		
1	2019-05-14	12.22		
1	2019-06-15	1.86		
1	2019-07-16	7.59		
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	←
1	2019-04-13	9.99		
1	2019-05-14	12.22		
1	2019-06-15	1.86		
1	2019-07-16	7.59		
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	
1	2019-04-13	9.99	5.16	←
1	2019-05-14	12.22		
1	2019-06-15	1.86		
1	2019-07-16	7.59		
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	
1	2019-04-13	9.99	5.16	
1	2019-05-14	12.22	8.40	←
1	2019-06-15	1.86		
1	2019-07-16	7.59		
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	
1	2019-04-13	9.99	5.16	
1	2019-05-14	12.22	8.40	
1	2019-06-15	1.86	8.02	←
1	2019-07-16	7.59		
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	
1	2019-04-13	9.99	5.16	
1	2019-05-14	12.22	8.40	
1	2019-06-15	1.86	8.02	
1	2019-07-16	7.59	7.22	←
1	2019-08-17	6.09		
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	
1	2019-04-13	9.99	5.16	
1	2019-05-14	12.22	8.40	
1	2019-06-15	1.86	8.02	
1	2019-07-16	7.59	7.22	
1	2019-08-17	6.09	5.18	←
1	2019-09-18	4.85		

Rolling average (past 3 records)

CustID	OrderDate	Price	RollAvg	
1	2019-01-10	1.00	1.00	
1	2019-02-11	2.50	1.75	
1	2019-03-12	3.00	2.17	
1	2019-04-13	9.99	5.16	
1	2019-05-14	12.22	8.40	
1	2019-06-15	1.86	8.02	
1	2019-07-16	7.59	7.22	
1	2019-08-17	6.09	5.18	
1	2019-09-18	4.85	6.18	←

```
select CustId,
    OrderDate,
    Price,
    avg(Price)
        over (partition by CustId
        order by OrderDate
        rows between 2 preceding
            and current row)
        as RollAvg
from #sales;
```

That OVER clause

The secret sauce: The OVER clause

- `PARTITION BY` clause (one or more expression)
    - The "`GROUP BY`" experience
- `ORDER BY` clause (one or more expression)
    - Determine sort order
- `ROWS/RANGE` clause (see next slide)
    - Frame the window
- Any or all of these may be optional or required

ROWS/RANGE syntax

- Most commonly we use `BETWEEN` syntax
- `ROWS` | `RANGE` `between` X `and` Y
  - `UNBOUNDED PRECEDING`
  - *n* `PRECEDING`
  - `CURRENT ROW`
  - *n* `FOLLOWING`
  - `UNBOUNDED FOLLOWING`

ROWS/RANGE syntax

- Less common (valid only for `ROWS`)
  - Omit `BETWEEN` and only include preceding portion
  - Example: `ROWS 2 PRECEDING`
  - End of frame is implied to be `CURRENT ROW`

ROWS vs. RANGE

- ROWS ignores ties in the ORDER BY clause
- RANGE treats ties as equal
- Beware of nondeterministic results using ROWS

OrderDate	Price	SumRows	SumRange
01/10/2019	1.00	1.00	1.00
02/20/2019	2.00	3.00	6.00
02/20/2019	3.00	6.00	6.00

```
select OrderDate,
    Price,
    sum(Price)
        over (order by OrderDate
        rows between unbounded
        preceding and current row)
        as SumRows,
    sum(Price)
        over (order by OrderDate
        range between unbounded
        preceding and current row)
        as SumRange
from #sales;
go
```

ROWS vs. RANGE

- `ROWS` typically performs better than `RANGE`
- Default is `RANGE BETWEEN UNBOUNDED PRECEEDING AND CURRENT ROW`
- This default can cause unexpected results

A side note

Where can window functions appear?

```
select sum() over (...)              ✓
from TableA a join TableB b
on a.ID = b.sum() over (...)         ✗
where 1000.0 <= sum() over (...)     ✗
group by sum() over (...)            ✗
having sum() over (...) = 1          ✗
order by sum() over (...);           ✓
```

Use CTE or subquery to use elsewhere

```
with RunningTotals as
(
    select CustID, OrderDate, Price,
        sum(Price)
            over (partition by CustId
            order by OrderDate) as RTotal
    from #sales
)
select CustID, OrderDate, Price, RTotal
from RunningTotals
where RTotal >= 10.00;
```

Specific window functions

Types of window functions

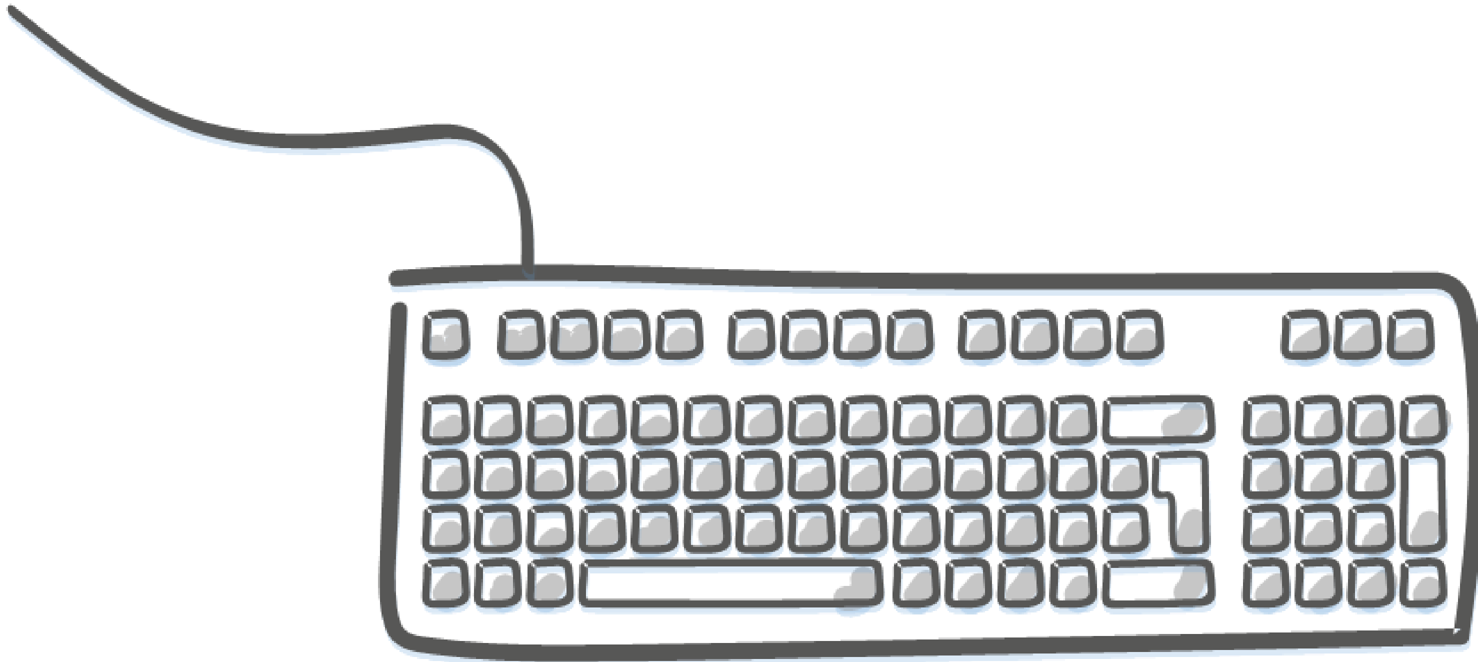
- Ranking
  - ROW_NUMBER, RANK, DENSE_RANK, NTILE
- Aggregate
  - COUNT, COUNT_BIG, SUM, AVG, MIN, MAX, STDDEV, STDDEVP, VAR, VARP, GROUPING, GROUPING_ID, CHECKSUM_AGG
- Analytic
  - LAG, LEAD, FIRST_VALUE, LAST_VALUE, CUME_DIST, PERCENT_RANK, PERCENTILE_CONT, PERCENTILE_DISC

Lots of uses for ROW_NUMBER

- Tally table / function
- Top *n* within group
- Duplicate detection / deletion
- Gaps and islands
- String splitting
- Bulk “identity” generation

Demo

Window Functions



POC Indexes

- POC = Partition + Order + Covering

```
select ic1, ic2, sum(wfcol1)
   over (partition by pc1, pc2
   order by obc1, obc2, obc3)
from tbl;

create index pocIndex on tbl
   (pc1, pc2, obc1, obc2, obc3)
   include (wfcol1, ic1, ic2);
```

WINDOW clause (2022+, Azure)

Provides a way to name windows

```
select TeamName, Wins, Losses,
    row_number() over win as TeamRowNumber,
    rank() over win as TeamRank,
    dense_rank() over win as TeamDenseRank
from #MLB2019RegularSeason
window win as (order by Wins desc)
order by ID;
```

WINDOW clause (2022+, Azure)

Can be nested:

```
select TeamName, League, Division, Wins, Losses,
    row_number() over WinOrder as OverallRowNumber,
    row_number() over WinLeague as LeagueRowNumber,
    row_number() over WinDivision as DivisionRowNumber
from #MLB2019RegularSeason
window WinOrder as (order by Wins desc),
    WinLeague as (WinOrder partition by League),
    WinDivision as (WinOrder partition by League, Division)
order by League, Division, ID;
```

Summary

- Window functions give us greater control over the frame of data that feeds input to a calculation
- Watch out for the default window frame (it's less performant and may give counter-intuitive results)
- Have lots of uses limited only by the imagination
- Sneak peek: future (who knows when) implementation of "nested window functions"

Thank You

- This presentation and supporting materials can be found at www.tf3604.com/windowfunctions.
  - Slide deck
  - Scripts

breanna@tf3604.com • @tf3604