

Window Functions Unleashed

Steen Dybboe



#275 | DENMARK 2014

Presenter Introduction

- **Steen Dybboe**
- Senior Consultant, Platon A/S
- Many years of experience designing, developing and maintaining software solutions based on Microsoft database and development platforms



A big thanks to our sponsors



A big thanks to our sponsors



```
SELECT ID, Salesdate
, SUM( Quantity)
  OVER (PARTITION BY SalesDate ORDER BY salesdate
  ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) Value
FROM dbo.SaleAll
ORDER BY Salesdate
```

Agenda

- Window Principles
- Window Functions
- Window Functions Optimizations
- Missing Language Support
- Usage

Window Function

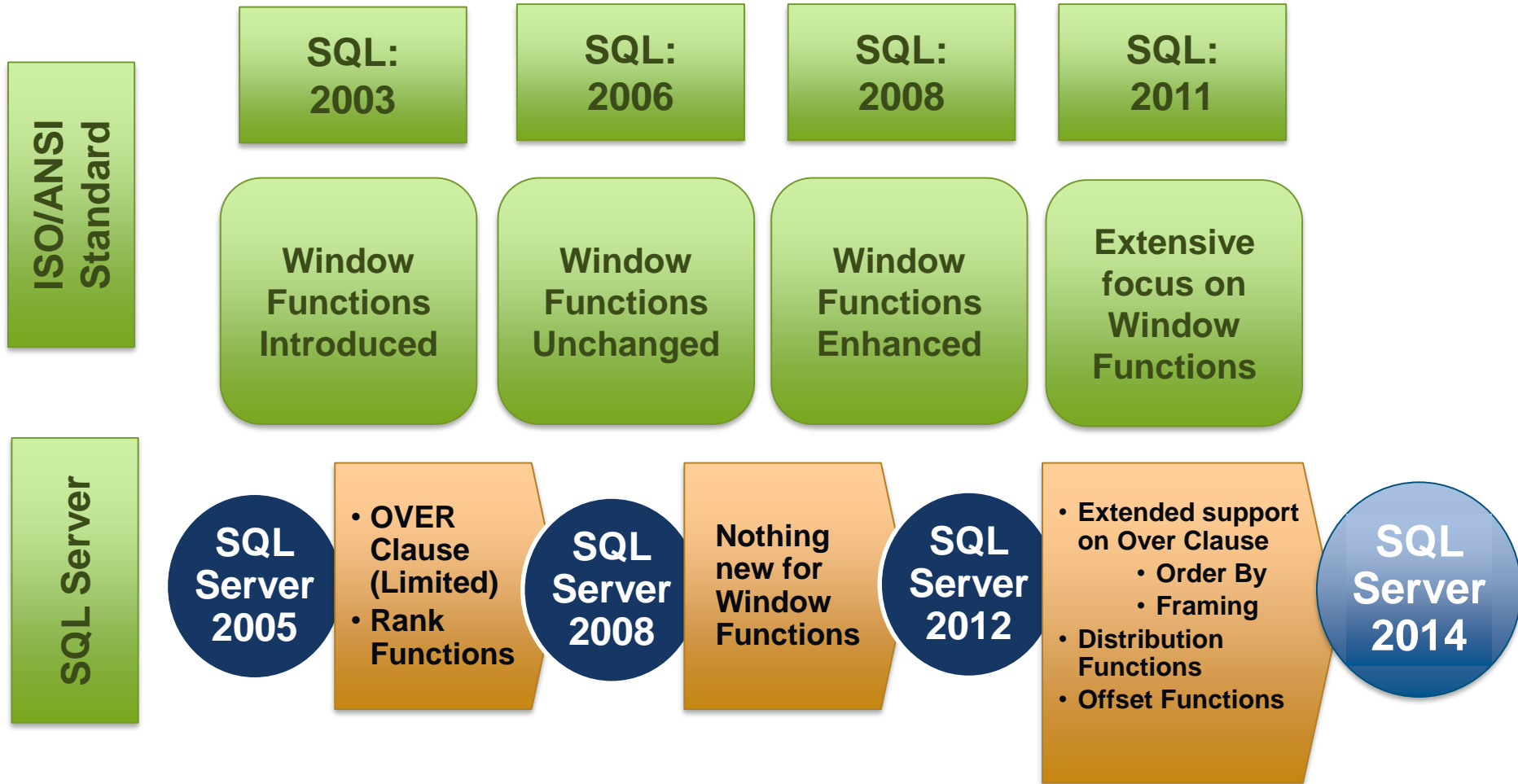
A window function is a function applied to a set of rows.

WINDOW is the term standard SQL uses to describe the context for the function to operate in.

SQL uses the OVER to provide the window specification.

```
SELECT actid, tranid, val
, AVG(val) OVER (PARTITION BY actid
                 ORDER BY tranid
                 ROWS BETWEEN 2 PRECEDING
                 AND 2 FOLLOWING) AS RunAvg
FROM dbo.Transactions;
```

ANSI Standards and SQL Server Implementation



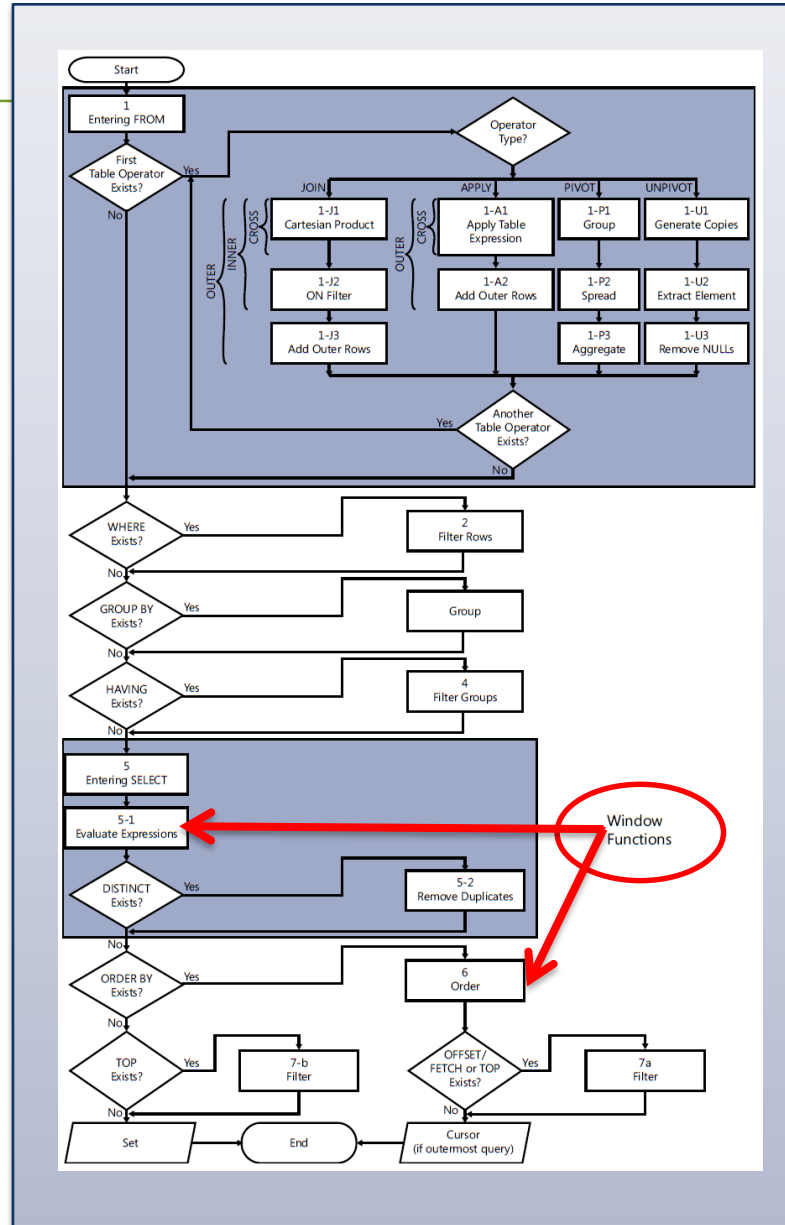
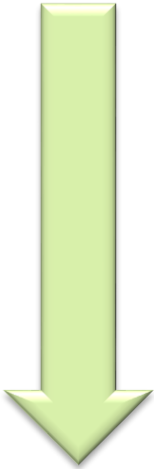
Window Data Computing - Alternatives

- GROUP queries
 - Force all calculations in the query to be done in context of groups
 - Imperative Approach – Transact SQL
- Correlated Subqueries
 - Each subquery requires separate visit to data
- CROSS APPLY queries
- Window functions
 - Starting point is underlying query's resultset
 - (hence only allowed in SELECT and ORDER BY)
 - Can utilize same access to data for multiple functions

Logical Query Processing

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

FROM
WHERE
GROUP BY
HAVING
Resultset
SELECT
ORDER BY




Window Functions

Window Function Categories

- Aggregation Functions
- Rank Functions
- Distribution Functions (SQL Server 2012)
- Offset Functions (SQL Server 2012)

SUM STDEVP
VAR CUME DISC
ROW_NUMBER
PERCENT_RANK
LAG
STDEV
DENSE_RANK PERCENTILE_DISC
LAST_VALUE CHECKSUM_AGG
RANK FIRST_VALUE
MAX MIN
COUNT_BIG





DEMO (1)
Window
Functions

Window Function Concepts

- **Partitioning**
 - Filter only rows with same values in partitioning elements
- **Ordering**
 - With aggregate functions, ordering just gives meaning to the framing option.
- **Framing**
 - placing boundaries on a defined order within a window
 - Once ordering is defined, framing identifies two bounds in the window partition, and only the rows between those two bounds are used.

Framing

- **[ROWS|RANGE] BETWEEN <Start expr> AND <End expr>**
- **<Start expr>** is one of:
 - *UNBOUNDED PRECEDING*: The window starts in the first row of the partition
 - *CURRENT ROW*: The window starts in the current row
 - *<unsigned integer literal> PRECEDING* or *FOLLOWING*
- **<End expr>** is one of:
 - *UNBOUNDED FOLLOWING*: The window ends in the last row of the partition
 - *CURRENT ROW*: The window ends in the current row
 - *<unsigned integer literal> PRECEDING* or *FOLLOWING*
- Defines the **Size of each window** which contributes to the aggregated value in the Row

“Window Functions Windows” Defined

- Partitioning
- Ordering
- Framing

```
SELECT name, dat, val
, SUM (val) OVER (
    PARTITION BY name
    ORDER BY dat
    ROWS BETWEEN UNBOUNDED PRECEDING
    AND CURRENT ROW
)
FROM dbo.MyData
```

Ken	20130415	5
Ken	20130417	4
Ken	20130416	2
Bob	20130416	2
Bob	20130415	5
Bob	20130419	4
Ken	20130419	3

Partitioning

Ken	20130415	5
Ken	20130417	4
Ken	20130416	2
Ken	20130419	3

Bob	20130415	5
Bob	20130419	4
Bob	20130416	2

Ordering

Ken	20130415	5
Ken	20130416	2
Ken	20130417	4
Ken	20130419	3

Bob	20130415	5
Bob	20130416	2
Bob	20130419	4

Framing

Ken	20130415	5	} →	5
Ken	20130416	2		7
Ken	20130417	4		11
Ken	20130419	3		14

Bob	20130415	5	} →	5
Bob	20130416	2		7
Bob	20130419	4		11

Running
sum

RANGE and ROWS

- How the Framing size is defined
- ROWS – Fixed or predictable number of ROWS
- RANGE – Range of data (not definable)
 - Not supported by the In-memory spool (more later)
- For now Stick with ROWS

```
ROWS BETWEEN
  UNBOUNDED PRECEDING |
    <n> PRECEDING      |
    <n> FOLLOWING       |
  CURRENT ROW
AND
  UNBOUNDED FOLLOWING |
    <n> PRECEDING      |
    <n> FOLLOWING       |
  CURRENT ROW
```

```
SELECT salesdate, val
, AVG( val) OVER (
  PARTITION BY id
  ORDER BY salesdate
  RANGE BETWEEN 1 MONTH PRECEDING
                AND 1 MONTH FOLLOWING) MonthAvg
FROM fact.AllSalesData
```

```
RANGE BETWEEN
  UNBOUNDED PRECEDING |
  CURRENT ROW
AND
  UNBOUNDED FOLLOWING |
  CURRENT ROW
```

Common T-SQL Challenges

Period to Date Aggregation

Identifying Gaps and Islands

Filling NULL Values with Preceding None
Null Value

Running and Sliding Aggregation

Calculating Median Value


Max Concurrent Intervals

Identifying Overlaps

De-duplicating Data

....





**DEMO (2)
Window Functions
and Framing**

Indexing principles

- POC = *Partitioning, Ordering, Covering*
- Index Keys should be the window partition columns followed by the window order columns,
- Index should include the rest of the columns that the query refers to.
 - Nonclustered index – List in INCLUDE clause
 - Clustered Index - includes all table columns in the leaf rows.

```
SELECT actid, tranid, val  
, ROW_NUMBER() OVER(PARTITION BY actid ORDER BY val) AS rownum  
FROM dbo.Transactions;
```

P **O** **C**

```
CREATE INDEX IX_actid_val_i_tranid  
ON dbo.Transactions(actid, val)  
INCLUDE(tranid);
```



WINDOW FUNCTIONS OPTIMIZATIONS

Window Spool Implementation

- Window Spool Operator
- Stream Aggregate Operator
 - Enhanced
- In-Memory or On-Disk Work Table



Stream Aggregate
(Aggregate)
Cost: 6 %



Window Spool
Cost: 24 %

Stream Aggregate	
Compute summary values for groups of rows in a suitably sorted stream.	
Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	2000000
Actual Number of Batches	0
Estimated I/O Cost	0
Estimated Operator Cost	1,256 (6%)
Estimated CPU Cost	1,256
Estimated Subtree Cost	13,4692
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	2000000
Estimated Row Size	47 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	8
Output List	
[TSQL2012].[dbo].[Transactions].actid; [TSQL2012].[dbo].[Transactions].tranid; [TSQL2012].[dbo].[Transactions].val; CumulativeBottom1017; CumulativeBottom1019	
Group By	
WindowCount1031	

Window Spool	
Expands each row into the set of rows that represent the window associated with it.	
Physical Operation	Window Spool
Logical Operation	Window Spool
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	199495100
Actual Number of Batches	0
Estimated Operator Cost	54,4284 (58%)
Estimated I/O Cost	0
Estimated CPU Cost	48,846
Estimated Subtree Cost	75,9319
Estimated Number of Executions	1
Number of Executions	4
Estimated Number of Rows	200000000
Estimated Row Size	55 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	2
Output List	
[TSQL2012].[dbo].[Transactions].actid; [TSQL2012].[dbo].[Transactions].tranid; [TSQL2012].[dbo].[Transactions].val; RowNumber1010; WindowCount1015; Segment1014	

In Memory

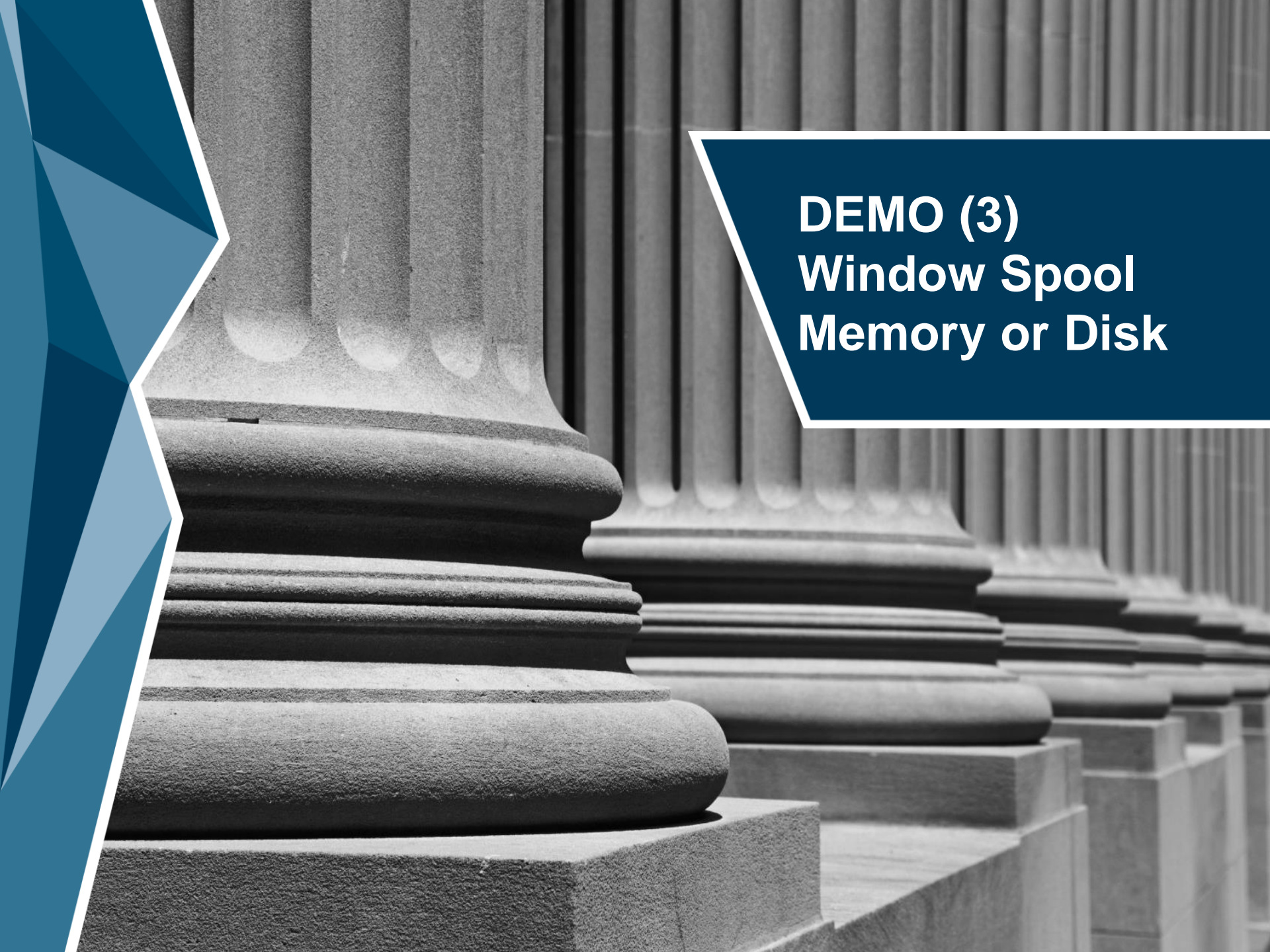


On Disk



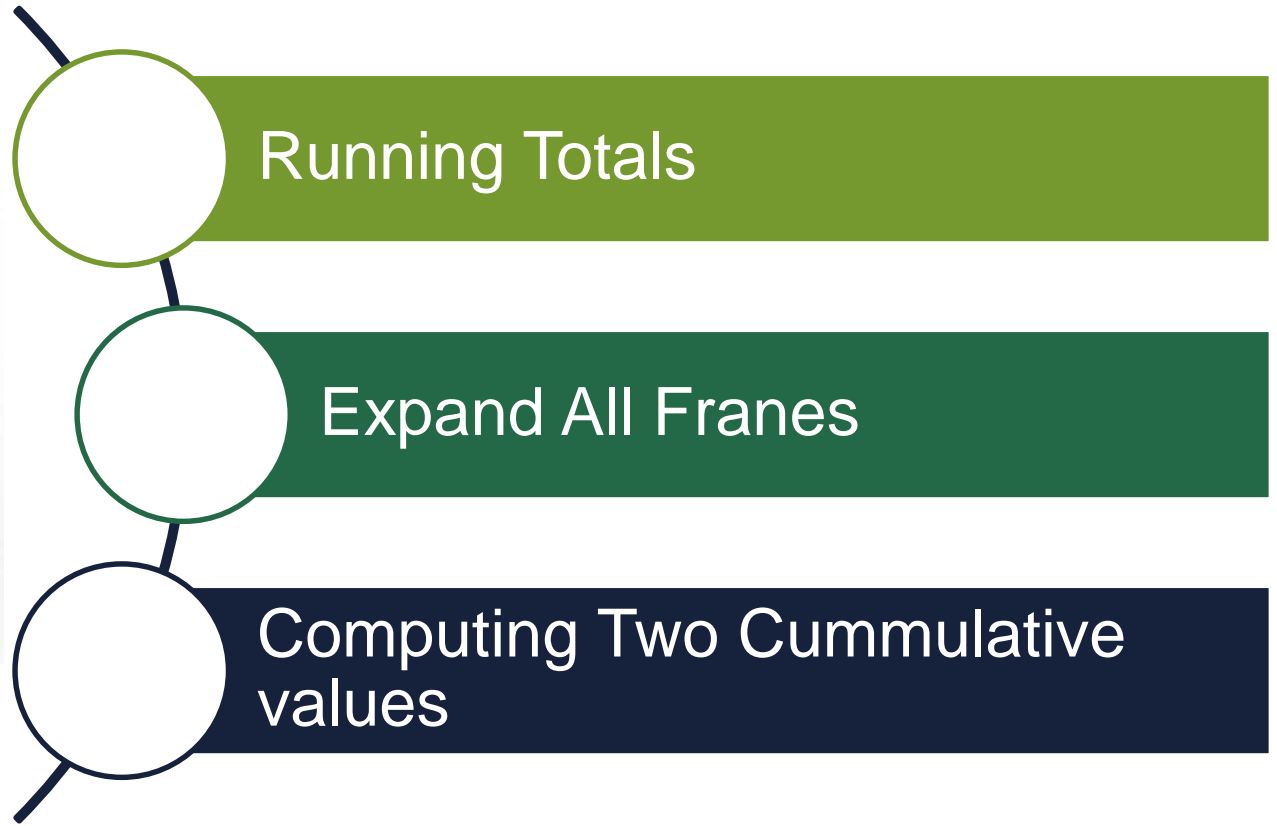
On-Disk or In-Memory

- A Spool uses a Work Table
- On-Disk overhead (locking, latches, I/O, ...)
- On-Disk detection
 - STATISTICS IO or Extended Events)
- On-Disk when:
 - RANGE Queries
 - Distance between two extreme points in frame exceeds 10.000
 - Using LAG or LEAD with expression as the offset



DEMO (3)
Window Spool
Memory or Disk

Optimizer Strategies – Three cases



Optimizer – Strategy 1

- Running Totals
 - using UNBOUNDED PRECEDING
- Method
 - Avoid expanding all frames
 - Summing up for Current Row separate
 - Append Current Row

```
SELECT actid, tranid, val
, SUM(val) OVER(PARTITION BY actid ORDER BY tranid
                ROWS BETWEEN UNBOUNDED PRECEDING
                AND CURRENT ROW) AS balance
FROM dbo.Transactions;
```

Optimizer – Strategy 2

- Expand All Frames
- Requirements
 - Not UNBOUNDED PRECEDING
 - Four or less rows in frame OR a NON cumulative Aggregate
 - **MIN, MAX, FIRST_VALUE, LAST_VALUE, CHECKSUM_AGG**
- If the frame has four rows or fewer, it isn't worthwhile to the optimizer to use optimization

```
SELECT actid, tranid, val
, SUM(val) OVER(PARTITION BY actid ORDER BY tranid
                ROWS BETWEEN 5 PRECEDING AND 2 PRECEDING)
FROM dbo.Transactions;
```

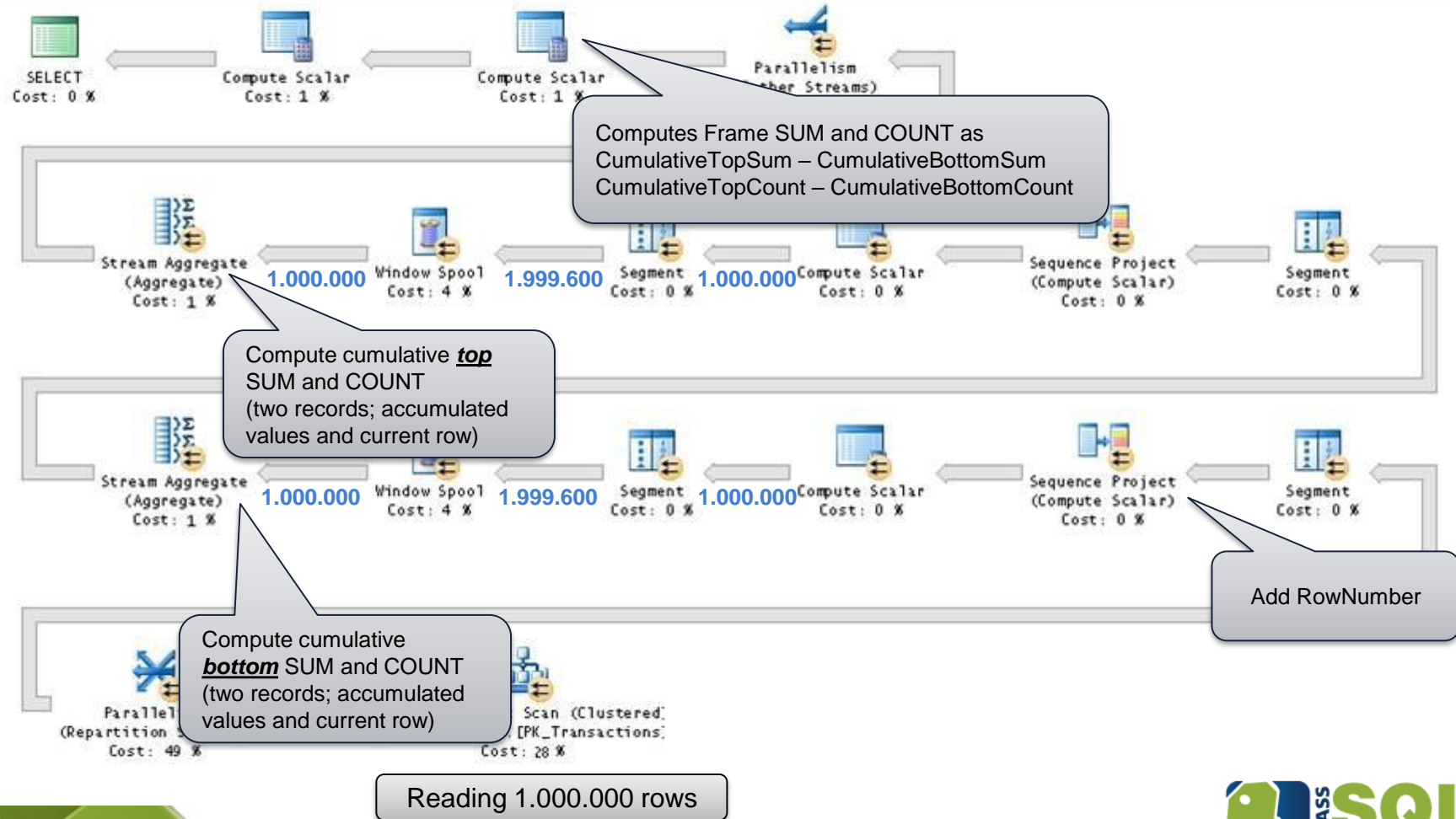
Optimizer – Strategy 3

- Optimize by computing two Cumulative values
 - Not UNBOUNDED PRECEDING
 - Four or less rows in frame
 - Cumulative Aggregate
 - SUM, COUNT, COUNT_BIG, AVG, STDEV, STDEVP, VAR, VARP
- Method
 - Calculates two cumulative values; bottom and top for frame
 - Compute result by subtracting top aggregate from bottom aggregate

```
SELECT actid, tranid, val
, SUM(val) OVER(PARTITION BY actid ORDER BY tranid
                ROWS BETWEEN 100 PRECEDING AND 2 PRECEDING)
FROM dbo.Transactions
```

Optimizer – Strategy 3

Query 1: Query cost (relative to the batch): 100%
 SELECT actid, tranid, val, SUM(val) OVER(PARTITION BY actid ORDER BY tranid ROWS BETWEEN 100 PRECEDING AND 2 PRECEDING) AS sumval FROM dbo.Transactions;



SAMPLES OF MISSING LANGUAGE CONSTRUCTS

Missing in Sql Server 2012

```
SELECT empid, ordermonth, qty,  
       SUM(qty) OVER (PARTITION BY empid  
                     ORDER BY ordermonth  
                     ROWS BETWEEN UNBOUNDED PRECEDING  
                           AND CURRENT ROW) AS run_sum_qty,  
       AVG(qty) OVER (PARTITION BY empid  
                     ORDER BY ordermonth  
                     ROWS BETWEEN UNBOUNDED PRECEDING  
                           AND CURRENT ROW) AS run_avg_qty,  
       MIN(qty) OVER (PARTITION BY empid  
                     ORDER BY ordermonth  
                     ROWS BETWEEN UNBOUNDED PRECEDING  
                           AND CURRENT ROW) AS run_min_qty,  
       MAX(qty) OVER (PARTITION BY empid  
                     ORDER BY ordermonth  
                     ROWS BETWEEN UNBOUNDED PRECEDING  
                           AND CURRENT ROW) AS run_max_qty  
FROM Sales.EmpOrders;
```

Reuse Window Definitions

```
SELECT empid, ordermonth, qty,  
       SUM(qty) OVER W1 AS run_sum_qty,  
       AVG(qty) OVER W1 AS run_avg_qty,  
       MIN(qty) OVER W1 AS run_min_qty,  
       MAX(qty) OVER W1 AS run_max_qty  
FROM Sales.EmpOrders  
WINDOW W1 AS (  
         PARTITION BY empid  
         ORDER BY ordermonth  
         ROWS BETWEEN UNBOUNDED PRECEDING  
         AND CURRENT ROW );
```



Missing in Sql Server 2012

Filtering on Window frames

```
SELECT orderid, orderdate, val  
FROM Sales.OrderValues  
QUALIFY RANK() OVER(ORDER BY val DESC) <= 5;
```



```
SELECT orderid, orderdate, val,  
RANK() OVER(ORDER BY val DESC) AS rnk  
FROM Sales.OrderValues  
QUALIFY rnk <= 5;
```



SUMMING UP

Preferred Modus Operandi

	Imperative Approach	Correlated Sub Queries	Window Function
PROS ↑	Commonly used in other programming languages	Follows a SET based paradigm	Improved optimization Easy to write Follows a SET based paradigm
CONS ↓	Not Set Based Individual queries only are optimized Many code lines	Bad performance Non-trivial to write	Not always optimal solution Not always possible to use



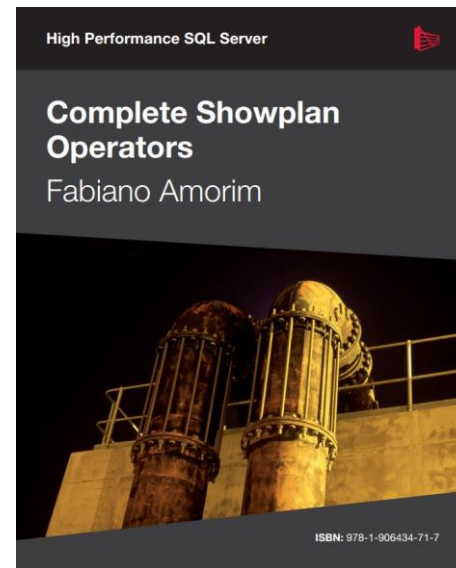
WINDOW FUNCTIONS ROCKS

Acknowledgments

MVP, Itzik Ben-Gan



MVP, Fabiano Amorim



Questions

