PASS
Data Community
SUMMIT 2021

Advanced T-SQL Querying and Query Tuning

Itzik Ben-Gan
T-SQL Trainer
Lucient

Learning Pathway: Query Tuning

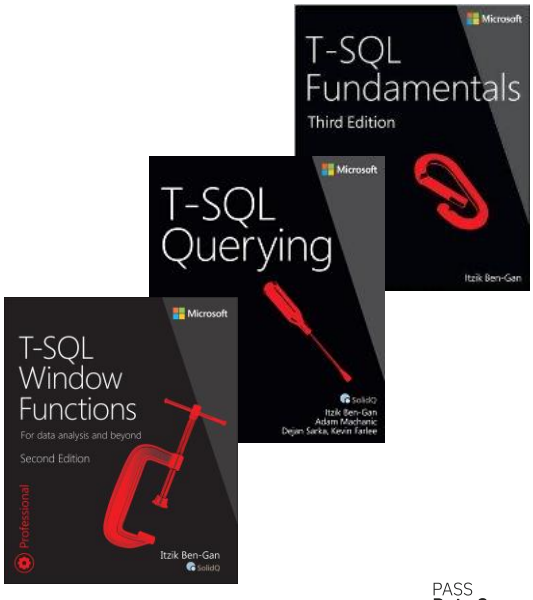
- Identify Poorly Performing Queries - Three Tools You Already Own
  - Grant Fritchey / level 100 / Wednesday Nov 10, 15:15 UTC
- Here's the Execution Plan … Now What?
  - Hugo Kornelis / level 200 / Wednesday Nov 10, 16:45 UTC
- Indexing for Performance
  - Kimberly Tripp / level 200 / Wednesday Nov 10, 21:15 UTC
- Demystifying Statistics in SQL Server
  - Erin Stellato / Level 200 / Thursday Nov 11, 14:30 UTC
- Query Tuning Tips Concerning Table Expressions
  - Itzik Ben-Gan / Level 300 / Thursday Nov 11, 16:45 UTC
- Tackling Monster Stored Procs
  - Allen White / Level 200 / Thursday Nov 11, 19:00 UTC

PASS
Data Community
SUMMIT 2021

Itzik Ben-Gan

T-SQL Trainer

Lucient



@ItzikBenGan

/itzikbengan

sqlperformance.com/author/itzikbengan

PASS Data Community SUMMIT 2021

Agenda

- Search arguments
- Join ordering optimization
- Adaptive / Intelligent QP
  - Batch-mode processing and batch mode on rowstore
  - Adaptive joins
  - Interleaved execution and table variable deferred compilation
  - Scalar UDF inlining
  - Memory grant feedback
- APPLY
- Window functions
- OFFSET-FETCH

Search argument (SARG)

- SARG: filter predicate that enables the optimizer to rely on index order
  - No manipulation of filtered column (OK to manipulate other side)
  - Operator represents a consecutive range of keys
- Careful with NULLs!

```sql
-- Not SARG
SELECT orderid, shippeddate
FROM Sales.Orders
WHERE YEAR(shippeddate) = 2018;

-- SARG
SELECT orderid, shippeddate
FROM Sales.Orders
WHERE shippeddate >= '20180101'
  AND shippeddate < '20190101';
```

```sql
-- SARG, incorrect
SELECT orderid, shippeddate FROM Sales.Orders
WHERE shippeddate = @dt;

-- Correct, not SARG
SELECT orderid, shippeddate FROM Sales.Orders
WHERE ISNULL(shippeddate, '99991231') = ISNULL(@dt, '99991231');

-- Correct, ugly SARG
SELECT orderid, shippeddate FROM Sales.Orders
WHERE shippeddate = @dt
   OR (shippeddate IS NULL AND @dt IS NULL);

-- Correct, elegant SARG
SELECT orderid, shippeddate FROM Sales.Orders
WHERE EXISTS(SELECT shippeddate INTERSECT SELECT @dt);
```


Join ordering optimization

- Joins are commutative (A J B = B J A) and associative (A J B) J C = A J (B J C)
- Optimizer explores candidate plans with different orders
  - Limited transformations with outer joins: A LJ B = B RJ A
  - More flexibility with inner and cross joins; theoretically (2N - 2)!/(N - 1)! permutations
  - To reduce exploration space, certain layouts not considered by default, e.g., bushy
- Use FORCE ORDER query hint or SET FORCEPLAN option for troubleshooting

```sql
SELECT DISTINCT C.companyname AS customer,
  S.companyname AS supplier
FROM Sales.Customers AS C
  INNER JOIN Sales.Orders AS O
    ON O.custid = C.custid
  INNER JOIN Sales.OrderDetails AS OD
    ON OD.orderid = O.orderid
  INNER JOIN Production.Products AS P
    ON P.productid = OD.productid
  INNER JOIN Production.Suppliers AS S
    ON S.supplierid = P.supplierid
OPTION (FORCE ORDER);
```

Join ordering optimization

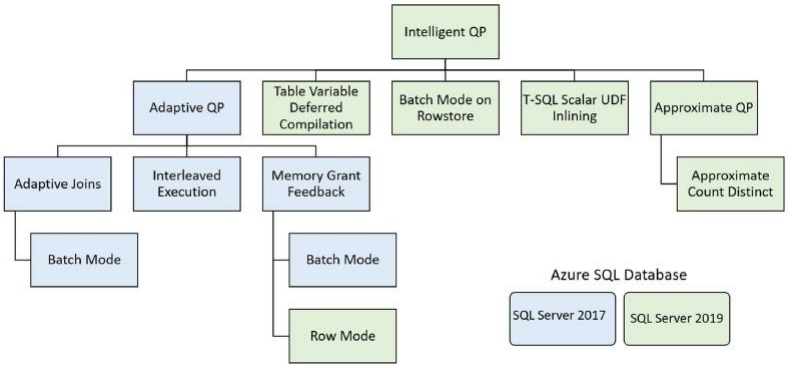
- ON-clause order defines logical join ordering (and physical w/FORCE ORDER)
- For query to be valid ON clause must appear below two units being joined

```
-- Join O with OD first, then C with result
-- Use as physical order
SELECT DISTINCT C.companyname AS customer,
  S.companyname AS supplier
FROM Sales.Customers AS C
  INNER JOIN
    ( Sales.Orders AS O
        INNER JOIN Sales.OrderDetails AS OD
          ON OD.orderid = O.orderid )
      ON O.custid = C.custid
  INNER JOIN Production.Products AS P
    ON P.productid = OD.productid
  INNER JOIN Production.Suppliers AS S
    ON S.supplierid = P.supplierid
OPTION (FORCE ORDER);
```

```
-- Include customers with no matches
SELECT DISTINCT C.companyname AS customer,
  S.companyname AS supplier
FROM Sales.Customers AS C
  LEFT OUTER JOIN
    ( Sales.Orders AS O
        INNER JOIN Sales.OrderDetails AS OD
          ON OD.orderid = O.orderid
        INNER JOIN Production.Products AS P
          ON P.productid = OD.productid
        INNER JOIN Production.Suppliers AS S
          ON S.supplierid = P.supplierid )
      ON O.custid = C.custid;
```


Adaptive / Intelligent QP

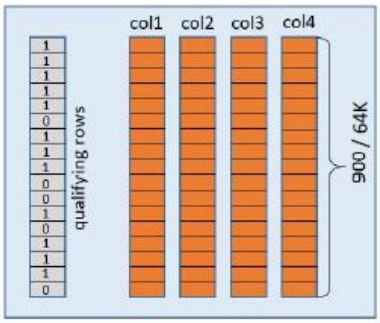
- SQL Server 2017 introduces adaptive query processing capabilities
- SQL Server 2019 enhances those to broader intelligent QP capabilities



Batch-mode processing and batch mode on rowstore

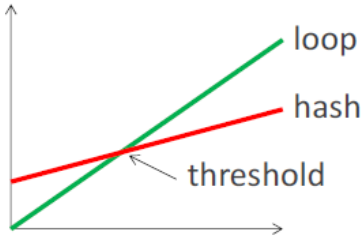
- Row mode: operator iterates once per row, metadata evaluated per row
- Batch mode: batch has vectorized representation of relevant columns, size 64KB/64-900 rows; operator iterates and evaluates metadata once per batch
- Considered automatically when columnstore indexes are present, even if not used
- Considered on rowstore starting with SQL Server 2019 Enterprise Edition under certain conditions (see queryprocessor.com/batch-mode-on-row-store)
- Backdoor: create dummy filtered columnstore index

```
CREATE NONCLUSTERED COLUMNSTORE INDEX idx_cs
  ON dbo.Orders(orderid)
  WHERE orderid = -1 AND orderid = -2;
```



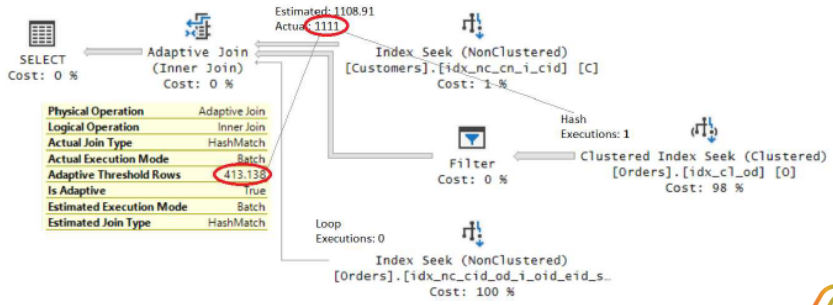
Adaptive joins

- Introduced in SQL Server 2017
- Supports both hash and loop joins, choice which to execute at run time based on threshold
- Same outer input
- One inner input for hash (middle branch)
- Another inner input for loop (bottom branch)
- >= threshold – hash, < threshold – loop
- Initially supported only in batch mode; can use dummy filtered columnstoreindex backdoor to enable



Adaptive joins

```
SELECT C.custid, C.custname,
  O.orderid, O.empid, O.shipperid, O.orderdate
FROM dbo.Customers AS C
  INNER JOIN dbo.Orders AS O
    ON O.custid = C.custid
WHERE C.custname LIKE @custprefix + N'%'
  AND O.orderdate BETWEEN @fromdate AND @todate;
```

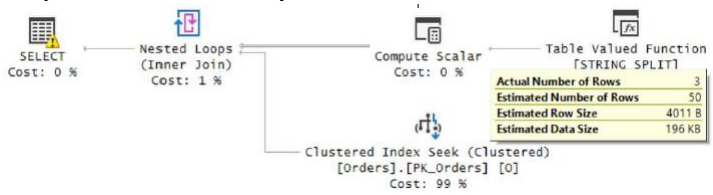


String splitting with the STRING_SPLIT function

- Introduced in SQL Server 2016

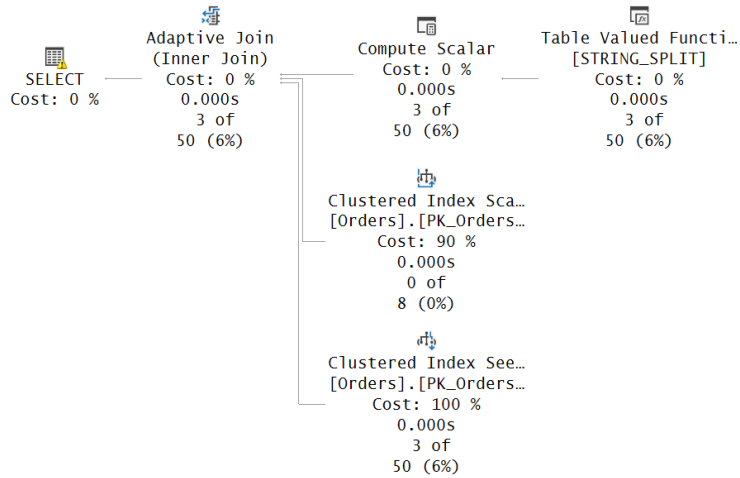
```
CREATE OR ALTER PROC dbo.GetOrders(@orderids AS VARCHAR(8000))
AS
SELECT O.orderid, O.orderdate, O.custid, O.empid
FROM Sales.Orders AS O
  INNER JOIN STRING_SPLIT(@orderids, ',') AS K
    ON O.orderid = CAST(K.value AS INT);
GO

EXEC dbo.GetOrders @orderids = '10248,10249,10250';
```

- Note: cardinality estimate always fixed 50



Adaptive join solves fixed cardinality issue

```
CREATE NONCLUSTERED COLUMNSTORE INDEX idx_dontworrybehappy
  ON Sales.Orders(orderid)
  WHERE orderid = -1 AND orderid = -2;
```



Interleaved execution

- Introduced in SQL Server 2017
- Used with queries involving multi-statement TVFs
- Initially optimize only until node representing table variable
- Once have actual number of rows, optimize remaining plan

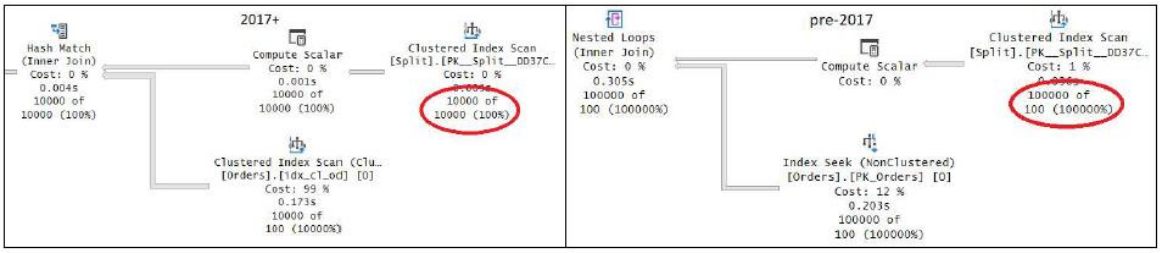
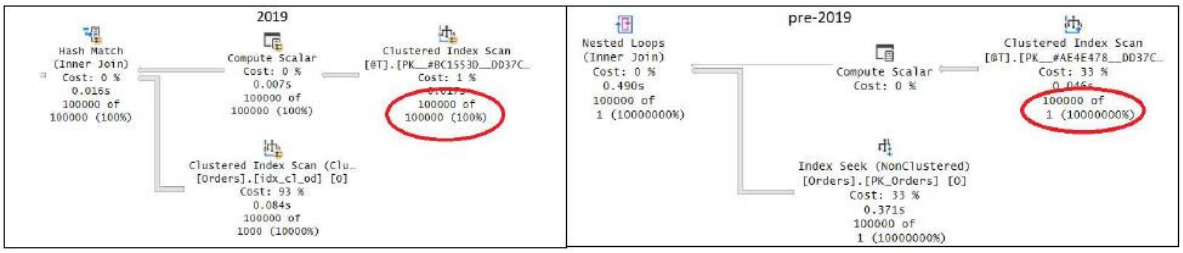


Table variable deferred compilation

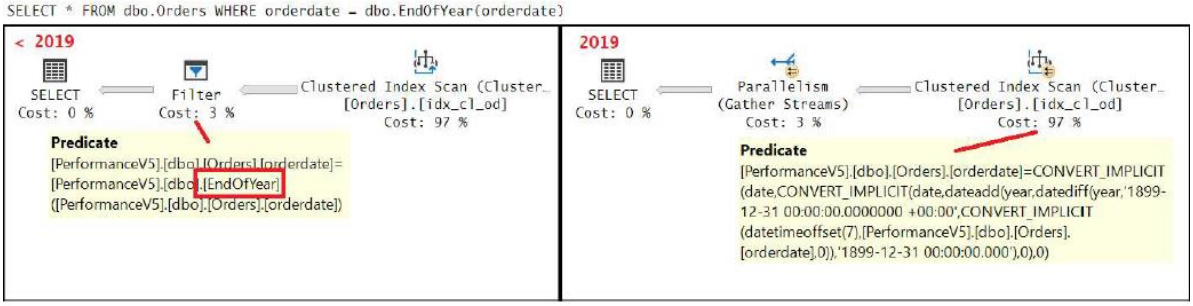
- Introduced in SQL Server 2019
- Similar to interleaved execution, but with regular table variables



Scalar UDF inlining

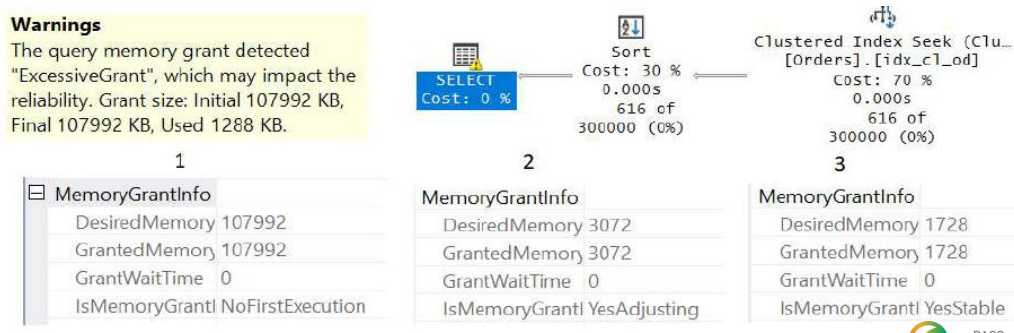
- Pre-2019 scalar UDFs did not get inlined and were a parallelism inhibitor
- SQL Server 2019 introduces scalar UDF inlining under certain conditions https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining
- Not just for RETURN <single expression>, also DECLARE, SET, IF get converted into relational counterparts such as CASE

```
SELECT * FROM dbo.Orders WHERE orderdate = dbo.EndOfYear(orderdate)
```



Memory grant feedback

- Memory grant correction over multiple executions for queries with cached plans
- 2017 introduces support for batch mode memory grant feedback
- 2019 adds support for row mode as well



APPLY

- Apply right table expression to each row from left table
- Table expression can have correlations (lateral/correlated join), can be TVF
- Left rows with no matches: discarded by CROSS APPLY, preserved by OUTER APPLY

```
SELECT C.custid, A.orderid, A.orderdate, A.empid
FROM Sales.Customers AS C
  CROSS APPLY ( SELECT TOP (3) orderid, orderdate, empid
                FROM Sales.Orders AS O
                WHERE O.custid = C.custid
                ORDER BY orderdate DESC, orderid DESC ) AS A;
```

Converting scalar UDFs to inline TVFs

- When scalar UDF does not get inlined

```
SELECT *
FROM dbo.Orders
WHERE orderdate = dbo.EndOfYear(orderdate);
```

- Convert to inline TVF if possible, and execute with APPLY

```
SELECT O.*
FROM dbo.Orders AS O
  CROSS APPLY dbo.EndOfYear(orderdate) AS F
WHERE O.orderdate = F.endofyear;
```


Using APPLY to get a seek-based strategy

- Example with MIN/MAX aggregate per group; normally you get a scan

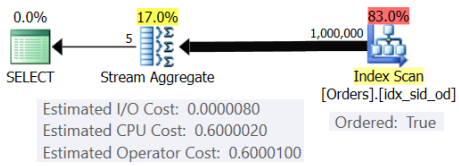
```
SELECT empid, MAX(orderdate) AS maxod
FROM dbo.Orders
GROUP BY empid;
```

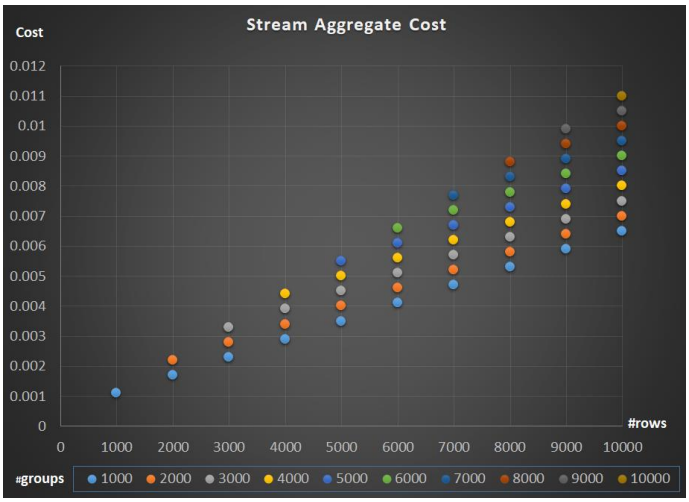
- Normal choices for grouped query:
  - Preordered Stream Aggregate
  - Sort + Stream Aggregate
  - Hash Aggregate

** For details see Optimization Thresholds Parts 1 – 5 at sqlperformance.com/author/itzikbengan*

Preordered Stream Aggregate

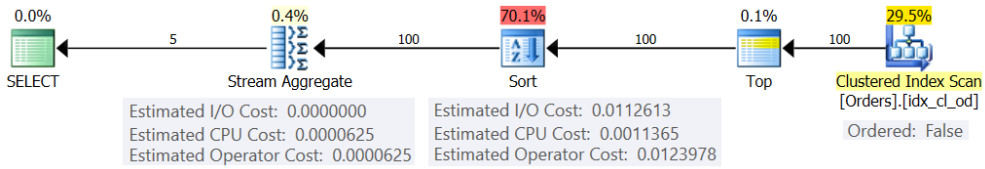
- I/O cost: negligible
- CPU cost:
  #rows * 0.0000006
  + #groups * 0.0000005
- Scaling: linear
- Used when supporting index exists





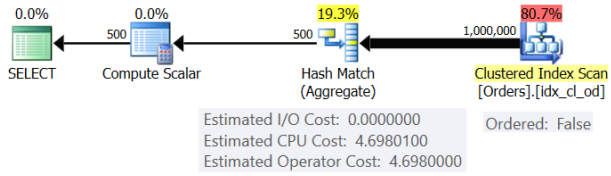
Sort + Stream Aggregate

- Scaling: n log n, negligible startup cost
- Used when:
  - no supporting index
  - small number of rows



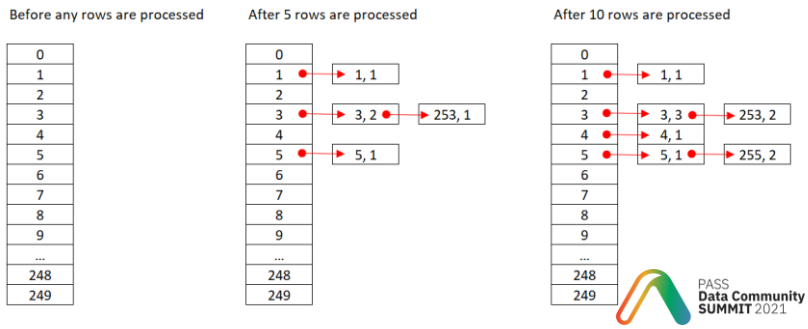
Hash Aggregate

- Scaling: linear, higher startup cost than Sort + SA
- Used when:
  - no supporting index
  - large number of rows

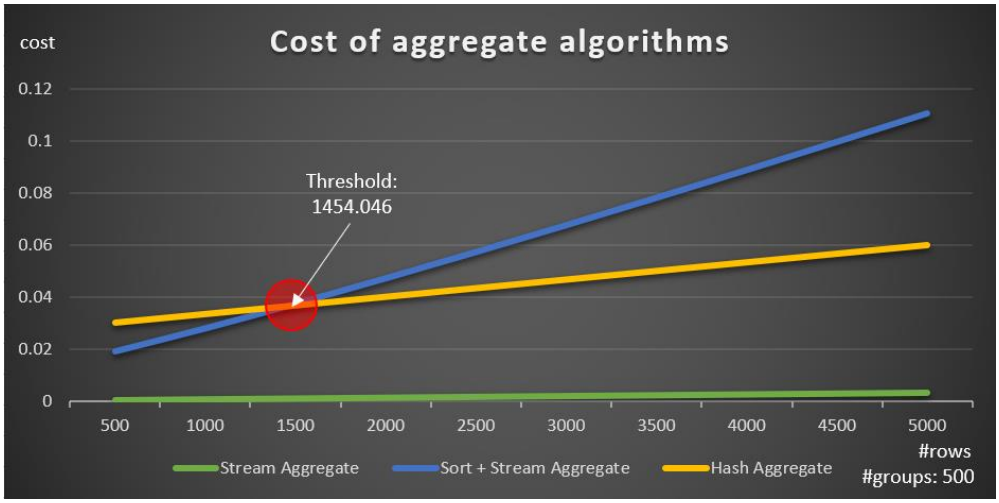


orderid	empid
320	3
30	5
660	253
820	3
850	1
1000	255
700	3
1240	253
350	4
400	255

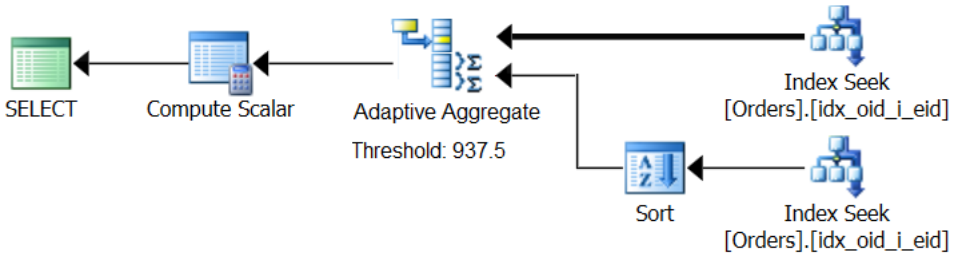
hash function:
empid % 250



Optimization threshold



Potential for Adaptive Aggregate operator

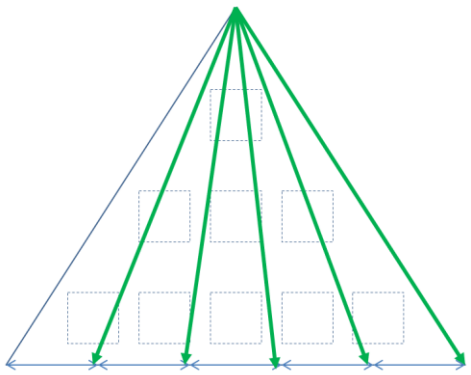


Wishful thinking

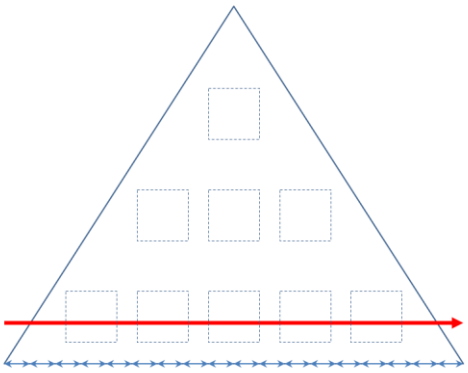
Scan versus seeks in grouped query

- Which is better?

High density: seeks



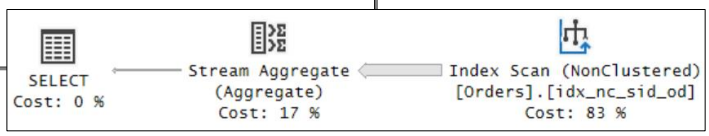
Low density: scan



Scan versus seeks in grouped query

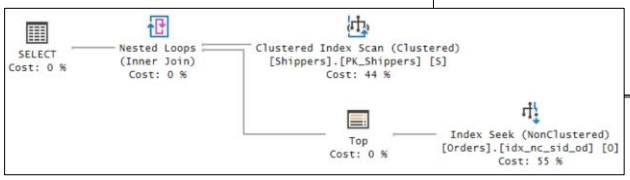
- Grouped query: scan

```
SELECT shipperid, MAX(orderdate) AS maxod
FROM dbo.Orders
GROUP BY shipperid;
```



- APPLY query: seeks

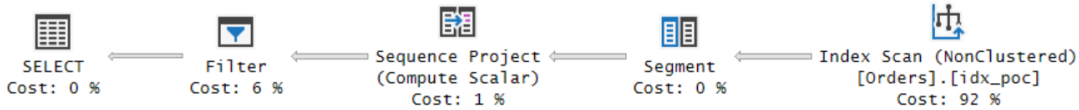
```
SELECT S.shipperid, O.maxod
FROM dbo.Shippers AS S
  CROSS APPLY ( SELECT TOP (1) O.orderdate
                FROM dbo.Orders AS O
                WHERE O.shipperid = S.shipperid
                ORDER BY O.orderdate DESC ) AS O(maxod);
```



Top N per group

- To get a scan use ROW_NUMBER

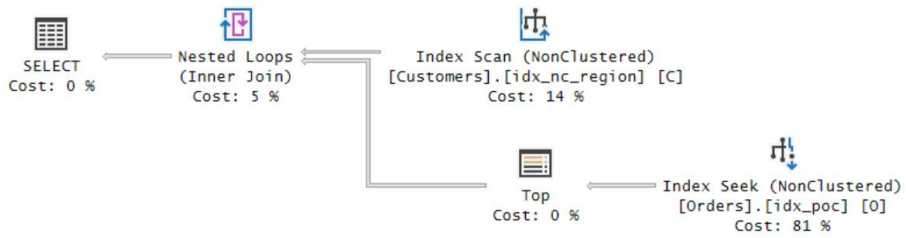
```
WITH C AS
(
  SELECT
    ROW_NUMBER() OVER(
      PARTITION BY custid
      ORDER BY orderdate DESC, orderid DESC) AS rownum,
    orderid, orderdate, custid, empid
  FROM Sales.Orders
)
SELECT custid, orderdate, orderid, empid
FROM C
WHERE rownum <= 3;
```



Top N per group

- To get seeks use APPLY

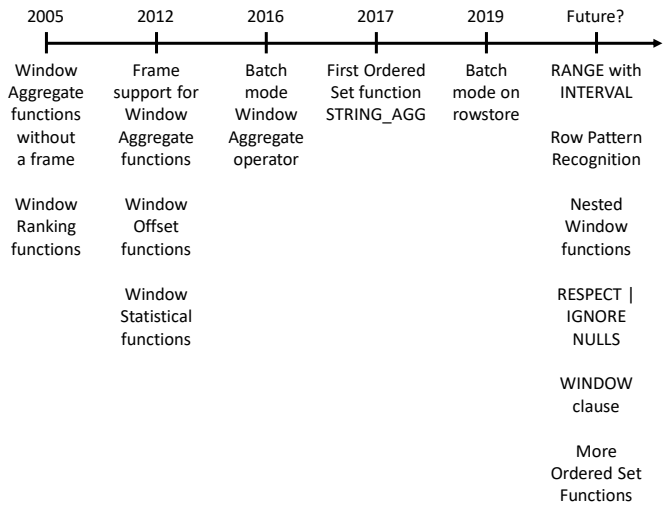
```
SELECT C.custid, A.*
FROM Sales.Customers AS C
  CROSS APPLY ( SELECT TOP (3) orderid, orderdate, empid
                FROM Sales.Orders AS O
                WHERE O.custid = C.custid
                ORDER BY orderdate DESC, orderid DESC ) AS A;
```



Many other uses of APPLY

- Reuse of column aliases
- Aggregate over columns (alternative to GREATEST/LEAST)
- Flexible unpivoting with multiple measures and control over NULLs
- MIN/MAX over partitioned tables
- And many others…

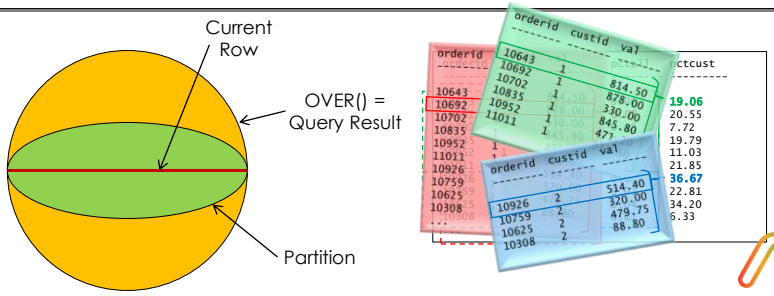
Window functions



Frameless aggregates

- Support optional window partition clause
- Do not involve a window order clause
- Not very optimal under row mode, but very optimal under batch mode

```
SELECT orderid, custid, val,
  CAST(100. * val / SUM(val) OVER()                    AS NUMERIC(5, 2)) AS pctall,
  CAST(100. * val / SUM(val) OVER(PARTITION BY custid) AS NUMERIC(5, 2)) AS pctcust
FROM Sales.OrderValues;
```



Ranking

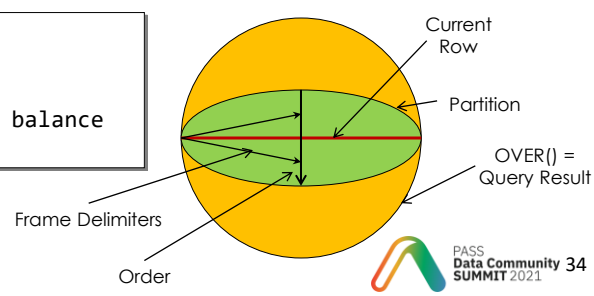
- Provide ranking values to rows in target partition
- Supported functions: ROW_NUMBER, RANK, DENSE_RANK, NTILE
- Support optional window partition clause and mandatory window order clause
- Benefit from POC index to avoid sort (relevant to all window functions)

```
SELECT orderid, qty,
  ROW_NUMBER() OVER(ORDER BY qty) AS rownum,
  RANK()       OVER(ORDER BY qty) AS rnk,
  DENSE_RANK() OVER(ORDER BY qty) AS densernk,
  NTILE(4)     OVER(ORDER BY qty) AS ntile4
FROM dbo.Orders;
```


Framed aggregates

- A frame is a restricted portion of a partition
- Use window order clause to define order in partition
- Use ROWS or RANGE to define delimiters
- Optimization of RANGE under row mode uses an on-disk spool
- When window order clause specified without unit, defaults to RANGE!

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



Offset

- Apply offset calculation to retrieve element from previous (LAG), next (LEAD) first (FIRST_VALUE), last (LAST_VALUE) row
- Support optional window partition clause and mandatory window order clause
- Warning: FIRST_VALUE and LAST_VALUE work with a frame; absent an explicit frame specification, you get RANGE UNBOUNDED PRECEDING by default

```
SELECT custid, orderid, orderdate, qty,
  LAG(qty)  OVER(PARTITION BY custid ORDER BY orderdate, orderid) AS prevqty,
  LEAD(qty) OVER(PARTITION BY custid ORDER BY orderdate, orderid) AS nextqty
FROM dbo.Orders;
```

```
SELECT custid, orderid, orderdate, qty,
  FIRST_VALUE(qty) OVER(PARTITION BY custid ORDER BY orderdate, orderid
                        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS firstqty,
  LAST_VALUE(qty)  OVER(PARTITION BY custid ORDER BY orderdate, orderid
                        ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS lastqty
FROM dbo.Orders;
```


Statistical

- Apply statistical calculations like percentiles and percentile ranks
- Support optional partitioning and mandatory within group ordering
- Not very optimal under row mode, but very optimal under batch mode

```
SELECT DISTINCT empid,
  PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qty) OVER(PARTITION BY empid) AS median_cont,
  PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY qty) OVER(PARTITION BY empid) AS median_disc
FROM dbo.Orders;
```

Batch mode Window Aggregate

- Introduced in SQL Server 2016
- Improves many inefficiencies beyond supporting optimal batch mode
- Pre-2019 requires a columnstore index to be present even if not used
- Can use backdoor with dummy filtered columnstore index to enable

Row mode

Batch mode

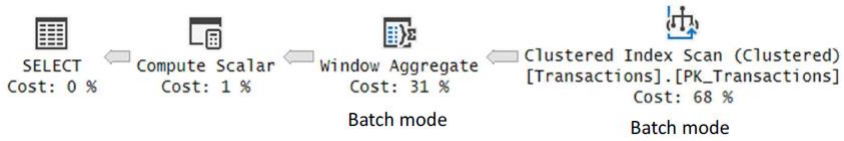
String concatenation

- Introduced in SQL Server 2017, as ordered set function

```
SELECT custid,
  STRING_AGG(CAST(orderid AS VARCHAR(10)), ',')
    WITHIN GROUP(ORDER BY orderdate DESC, orderid DESC) AS orderids
FROM Sales.Orders
GROUP BY custid;
```

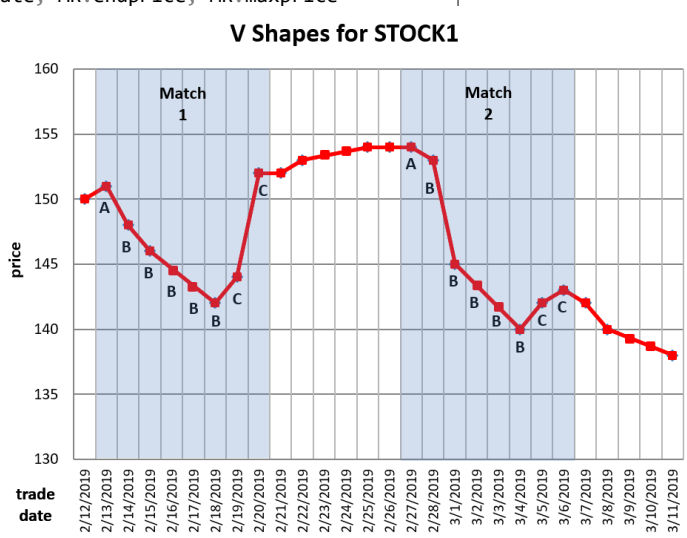
Batch mode on rowstore (covered earlier)

- Introduced in SQL Server 2019
- Doesn't require columnstore indexes to be present
- Allows scanning of rowstore data to happen natively in batch mode with no adapters
- Requires EE and certain conditions based on heuristics to be met (see queryprocessor.com/batch-mode-on-row-store), so backdoor could still be useful!



Missing, row pattern recognition (RPR)

```
SELECT
  MR.symbol, MR.matchnum, MR.startdate, MR.startprice,
  MR.bottomdate, MR.bottomprice, MR.enddate, MR.endprice, MR.maxprice
FROM dbo.Ticker
  MATCH_RECOGNIZE
  (
    PARTITION BY symbol
    ORDER BY tradedate
    MEASURES
      MATCH_NUMBER() AS matchnum,
      A.tradedate AS startdate,
      A.price AS startprice,
      LAST(B.tradedate) AS bottomdate,
      LAST(B.price) AS bottomprice,
      LAST(C.tradedate) AS enddate,
      LAST(C.price) AS endprice,
      MAX(price) AS maxprice
    PATTERN (A B+ C+)
    DEFINE
      -- A defaults to True
      B AS B.price < PREV(B.price),
      C AS C.price > PREV(C.price)
  ) AS MR;
```



Missing, nested window functions

- Interact with points in window as an argument of an aggregate function:
  BEGIN_PARTITION, BEGIN_FRAME, CURRENT_ROW, FRAME_ROW, END_FRAME, END_PARTITION

```
-- cur value minus average without first and last orders for customer
SELECT orderid, custid, orderdate, val,
  val - AVG( CASE
               WHEN ROW_NUMBER(FRAME_ROW) NOT IN ( 1, ROW_NUMBER(END_PARTITION) ) THEN val
             END )
          OVER( PARTITION BY custid ORDER BY orderdate, orderid
                ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING ) AS diff
FROM Sales.OrderValues;

-- cur value minus average without cur order date for customer
SELECT orderid, custid, orderdate, val,
  val - AVG( CASE WHEN orderdate <> VALUE OF orderdate AT CURRENT_ROW THEN val END )
          OVER( PARTITION BY custid ) AS diff
FROM Sales.OrderValues;
```


Missing, NULL treatment clause

- Available in SQL standard to offset window functions (LAG, LEAD, FIRST_VALUE, LAST_VALUE)
- IGNORE NULLs means keep going until a non-NULL value found

```
SELECT id, col1, COALESCE(col1, LAG(col1) IGNORE NULLS OVER(ORDER BY id)) AS lastval
FROM dbo.T1;
```

- Workaround

```
WITH C AS
(
  SELECT id, col1,
    MAX(CASE WHEN col1 IS NOT NULL THEN id END)
      OVER(ORDER BY id ROWS UNBOUNDED PRECEDING) AS grp
  FROM dbo.T1
)
SELECT id, col1,
  MAX(col1) OVER(PARTITION BY grp ORDER BY id ROWS UNBOUNDED PRECEDING)
FROM C;
```

Solutions using window functions

- Gaps
- Islands
- Creating auxiliary table of numbers (see GetNums function)
- Identifying maximum number of concurrent intervals
- Packing intervals
- And many others…

OFFSET-FETCH

- Similar to TOP but with skipping capability
- Useful for paging, but also tasks like median with high density groups
- Connected entirely to the ORDER BY
- OFFSET: rows to skip, FETCH: rows to filter

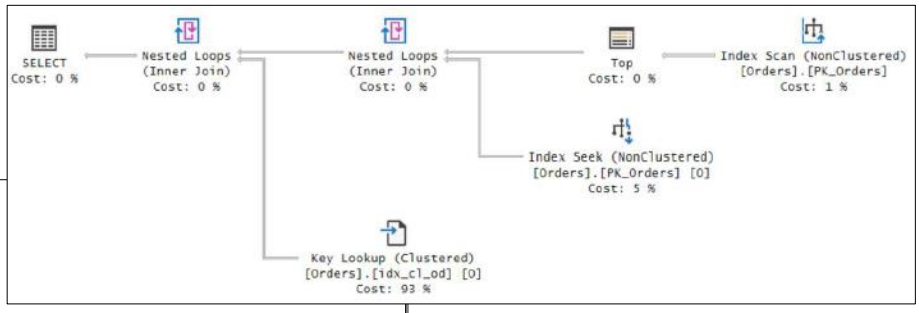
```
SELECT orderid, orderdate, custid, empid
FROM Sales.Orders
ORDER BY orderdate DESC, orderid DESC
OFFSET 50 ROWS FETCH NEXT 25 ROWS ONLY;
```

OFFSET-FETCH

- Optimized with enhanced Top operator
- #rows scanned in index = Offset exp + Top exp

```
SELECT orderid, orderdate, custid, empid
FROM dbo.Orders
ORDER BY orderid
OFFSET (@pagenum - 1) * @pagesize ROWS FETCH NEXT @pagesize ROWS ONLY;
```

OFFSET-FETCH




```
WITH Keys AS
(
  SELECT orderid
  FROM dbo.Orders
  ORDER BY orderid
  OFFSET (@pagenum - 1) * @pagesize ROWS
  FETCH NEXT @pagesize ROWS ONLY
)
SELECT O.orderid, O.orderdate, O.custid, O.empid
FROM dbo.Orders AS O
  INNER JOIN Keys AS K
    ON O.orderid = K.orderid;
```

- Use CTE+join to avoid unnecessary lookups

Review

- Search arguments
- Join ordering optimization
- Adaptive / Intelligent QP
  - Batch-mode processing and batch mode on rowstore
  - Adaptive joins
  - Interleaved execution and table variable deferred compilation
  - Scalar UDF inlining
  - Memory grant feedback
- APPLY
- Window functions
- OFFSET-FETCH

Session evaluation

Your feedback is important to us

Evaluate this session at:

www.PASSDataCommunitySummit.com/evaluation

PASS Data Community SUMMIT 2021

Thank you

All you need is SQL
All you need is SQL
All you need is SQL, SQL
SQL is all you need

Itzik Ben-Gan

@ItzikBenGan

/itzikbengan

sqlperformance.com/author/itzikbengan

PASS
Data Community
SUMMIT 2021