



#37 PARMA 2023

# T-SQL performance Tips & Tricks

Sergio Govoni



Slide and demos: <https://bit.ly/sg-datasat37>



UNIVERSITÀ DEGLI STUDI DI PARMA



Bi Factory

DATA KNOWLEDGE ADVISOR



# Sergio Govoni



Sergio Govoni



[linkedin.com/in/sgovoni](https://linkedin.com/in/sgovoni)



[github.com/segovoni](https://github.com/segovoni)



[twitter.com/segovoni](https://twitter.com/segovoni)

# Agenda

- SARGable predicates
- Query mode execution
- Join order
- Temp table cache contention on tempdb





1



# SARGable predicates

# The definition of SARGable

Wikipedia ([en.wikipedia.org/wiki/Sargable](https://en.wikipedia.org/wiki/Sargable)) defines **SARGability** in this way:

In relational databases, a condition (or predicate) in a query is said to be sargable if the DBMS engine can take advantage of an index to speed up the execution of the query. The term is derived from a contraction of **Search ARGument ABLE**.

# The definition of SARGable

A query failing to be sargable is known as a non-sargable query and typically has a negative effect on query time, so one of the steps in query optimization is to convert them to be sargable

The effect is similar to searching for a specific term in a book that has no index, **beginning at page one each time**, instead of **jumping to a list of specific pages** identified in an index!

# SARGable predicates

- SARGable means that the predicate can be evaluated/executed using a Seek
- Predicates



<column expression> <operator> <expression>



<column><operator><expression>





1



# DEMO



2

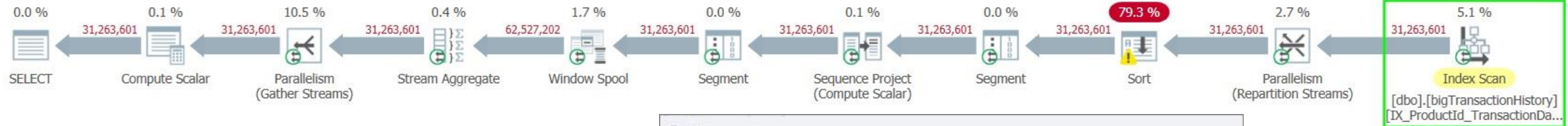


# Query mode processing

# Row mode execution

- Row mode execution is a **query processing method** used with traditional RDBMS tables, where data is stored in row format
- When a query is executed and accesses data in row store tables, the execution tree operators and child operators **read each required row, across all the columns** specified in the table schema
- From each row that is read, SQL Server retrieves the columns that are required for the result set, as referenced by a SELECT statement, JOIN predicate, or filter predicate

# Row mode execution



**Sort**  
Reorders the input.  
Node ID: 7  
Physical Operation: Sort  
Logical Operation: Sort  
**Estimated Execution Mode: Row**  
Estimated Rows: 31,263,600  
Estimated I/O Cost: 983.4300000  
Estimated CPU Cost: 805.0430000  
Estimated Executions: 1.0  
Estimated Operator Cost: 1,788.4730000 (79.3%)  
Estimated Subtree Cost: 1,964.1900000  
Estimated Row Size: 19 B  
Estimated Data Size: 566 MB  
Memory Fractions:  
In: 100.00%  
Out: 100.00%

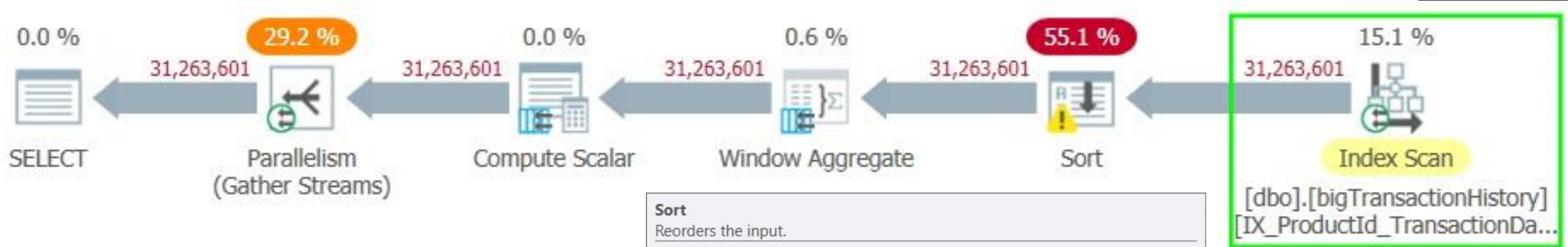
**Order By:**  
[AdventureWorks2017].[dbo].[bigTransactionHistory].[ProductID] Ascending  
[AdventureWorks2017].[dbo].[bigTransactionHistory].[TransactionID] Ascending

**Output List:**  
[AdventureWorks2017].[dbo].[bigTransactionHistory].[ProductID]  
[AdventureWorks2017].[dbo].[bigTransactionHistory].[Quantity]  
[AdventureWorks2017].[dbo].[bigTransactionHistory].[TransactionID]

# Batch mode execution


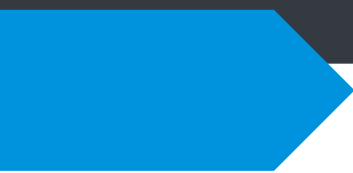
- Batch mode execution is a **query processing method** used to **process multiple rows together**, query operators process data more efficiently
- Each column within a batch is stored as a vector in a separate area of memory, so batch mode processing is **vector-based**
- Batch mode processing operates on compressed data when possible and eliminates the exchange operator used by row mode execution. The result is **better parallelism** and **faster performance**!

# Batch mode execution




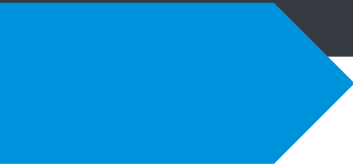
# Columnstore and query mode execution



- SQL Server 2012 introduced a new feature to accelerate analytical workloads: columnstore indexes
  - SQL Server expanded the use cases and improved the performance of columnstore indexes in each subsequent release
  - SQL Server 2016 enables the creation of empty filtered columnstore indexes
- 
- 

# Columnstore and query mode execution



- Up to SQL Server 2017 batch mode processing requires a columnstore index to be enabled!
  - Starting with SQL Server 2019 (15.x) and in Azure SQL Database, batch mode execution no longer requires columnstore indexes, the feature is called **Batch mode on rowstore!**
- 
- 





2

DEMO





3



# Join order

# Join order

- Query Optimizer must find the optimal sequence of joins between the tables used in the query, it defines the join order
- Finding the optimal join order is one of the most difficult problems in query optimization and it has to be done within the available time

# Join order

- Does the Query Optimizer analyze all possible join orders?
- No, it doesn't! 😞
- It finds a balance between the optimization time and the quality of the resulting plan

# Join order

Please, consider this query...

```
SELECT
  C.CustomerName, PS.SupplierName
FROM Sales.Customers AS C
INNER JOIN Sales.Orders AS O
  ON O.CustomerID=C.CustomerID
INNER JOIN Sales.OrderLines AS OL
  ON O.OrderID=OL.OrderID
INNER JOIN Warehouse.StockItems AS S
  ON OL.StockItemID=S.StockItemID
INNER JOIN Purchasing.Suppliers AS PS
  ON S.SupplierID=PS.SupplierID;
```

Supplier-Customer that have joint activity

Now imagine, you want to preserve customers who have no orders...

# Join order

```
SELECT
  C.CustomerName, PS.SupplierName
FROM Sales.Customers AS C
LEFT OUTER JOIN Sales.Orders AS O
  ON O.CustomerID=C.CustomerID
INNER JOIN Sales.OrderLines AS OL
  ON O.OrderID=OL.OrderID
INNER JOIN Warehouse.StockItems AS S
  ON OL.StockItemID=S.StockItemID
INNER JOIN Purchasing.Suppliers AS PS
  ON S.SupplierID=PS.SupplierID;
```

Query optimizer has detected a contradiction...

Hash Keys Build	[WideWorldImporters].[Sales].[Customers].Custo
Alias	[C]
Column	CustomerID
Database	[WideWorldImporters]
Schema	[Sales]
Table	[Customers]
Hash Keys Probe	[WideWorldImporters].[Sales].[Orders].Custome
Alias	[O]
Column	CustomerID
Database	[WideWorldImporters]
Schema	[Sales]
Table	[Orders]
Logical Operation	Inner Join



3



# DEMO





# Tempdb

- It stores
  - User objects
  - Work objects (worktable for Sort and Spool, etc.)
  - Version Store (Row Versioning)
- It's always recreated after SQL Server restart
- It uses simple recovery model
- One tempdb for the entire instance = It's a bottleneck by design!

# User objects in tempdb

- Local temporary tables
  - Prefix “#”, scope limited to the local session
  - Auto dropped after the session is closed
- Global temporary tables
  - Prefix “##”, visible in all sessions
  - Auto dropped after the session is closed
- Table variables and tables returned from the “Table Valued Functions”

# Creating a temp table on tempdb means

- Reading the SGAM page (2:1:3) to find an extent with free space
  - An exclusive latch is active during the update
- Reading the PFS page (2:1:1) to find a free page within the extent
  - An exclusive latch is active during the update
- A PAGELATCH\_\* wait type occurs
  - Resources have the form 2:x:x
  - 2:1:1, 2:1:2 and 2:1:3

# Temp table cache contention

- Temp table caching helped address metadata contention by allowing us to reuse tables
- Cache a temp table object
  - When you delete that table SQL Server doesn't actually drop the metadata
  - SQL Server keeps a cache of all the temporary objects that are used through a stored procedure and then it reuses the metadata for those objects



4



# DEMO



# Summary

- One of the steps in the query optimization process is to convert non-sargable predicates to **sargable predicates**
  - Pay attention to NULLs
- SQL Server 2016 enables the creation of **empty filtered columnstore** indexes that you can use to enable batch mode execution in the OLTP scenarios without maintenance costs on columnstore indexes
- The logical join ordering is determined by the **order of ON clauses**
- Temp table caching helped address metadata contention by allowing us to **reuse tables**

# Resources



- Sargable predicates and NULLs in SQL Server
  - <https://segovoni.medium.com/sargable-predicates-and-null-values-in-sql-server-c43ec3d8b108>
- Query mode execution
  - <https://segovoni.medium.com/sql-server-query-mode-execution-and-columnstore-indexes-fa05152c0753>
  - <https://bit.ly/3Hmcyuf>
- Thinking Big (Adventure) by Adam Machanic
  - <http://dataeducation.com/thinking-big-adventure>

Data Saturday Parma wants your feedback!

Data Saturday #37 Feedback Form







#37 PARMA 2023

Grazie!!!

