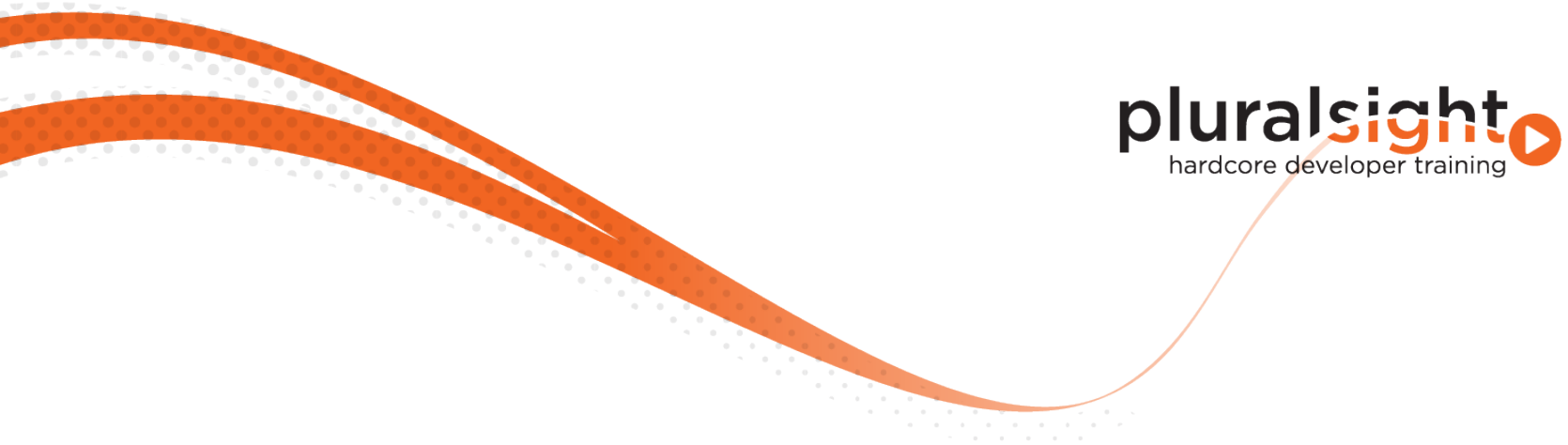
SQL Server: Query Plan Analysis

Module 5: Noteworthy Patterns

Joe Sack
Joe@SQLskills.com



pluralsight
hardcore developer training

Module Introduction

- There are several patterns and indicators within a query execution plan that should draw your attention
- Some of these noteworthy patterns may end up not being an issue, but some may be tied to issues that must be addressed
- This module discusses the various patterns and indicators that you can use for successfully analyzing a plan and coming up with actionable next steps

The “Basics”

- **There are several common areas that people will pay attention to within a query execution plan, including:**
  - High estimated cost across queries (query cost relative to batch)
    - As shown earlier in the course, this can be flawed
  - High estimated cost within a query
  - Scans
    - Table or Index
  - Thick vs. skinny data flow lines
  - Missing index warnings

Lookups

- **Why do we care?**
  - For each row in the nonclustered index, an associated clustered index or heap I/O (random) is required
  - Even if all applicable pages are cached, you can STILL have inflated overhead (compared to a covering index) due to the increased number of random logical reads

Nested Loop Patterns

- **For a Nested Loop join:**
  - Memory requirements are lower comparatively
    - OLTP vs. DW
  - Look for “smaller” table as outer (top) table
  - Look for under-estimates for inner table or index scans (very costly)
  - Nested Loops may be associated with inflated random I/Os when the row estimates are significantly skewed from the actual number of rows

Merge Join Patterns

- **For a Merge Join:**
  - Memory requirements also lower (generally)
    - Many-to-many joins have overhead (worktables)
      - Look for ManyToMany attribute
  - Outer (top) / inner (bottom) requires sort on join key
    - If the sort is “injected” into the plan by the Query Optimizer – take note of it
  - Query Optimizer injected sorts have a risk of spilling to disk (tempdb)

Hash Join Patterns

- **For a Hash Join (Match):**
  - Typical case is that the smaller table is the “build table” (red flag if you see otherwise)
    - Hash table must be generated FIRST before the probe begins, so a smaller table would ideally reduce the latency between phases
  - Hash build or probe can spill to disk typically due to cardinality estimates
    - Hash Warning events can be found in SQL Trace and Extended Events
    - As of SQL Server 2012 – spill notifications also appear within the plan

Stop-and-Go Operators

- **Stop-and-go operators must read ALL rows from the child operator before it can pass rows to parent or perform specific actions**
- **Examples of stop-and-go operators include:**
  - Sort
  - Eager Spool
  - Hash Join
    - Outer (top) table in “blocking input”, not inner
  - Hash Aggregate
- **Examples of operators that can keep streaming one row for every row that is read:**
  - Nested loop
  - Compute scalar

Sort Patterns

- **Extraneous sorting**
  - Is the sort required in the first place (from a logical perspective)?
- **Query Optimizer injected sorts**
  - Would a sorted index help reduce the overhead?
- **Sort Tempdb spills**
  - Sort can occur in tempdb for large data sets and memory constraints (see Sort Warnings)
  - May not be needed if you have supporting indexes or remove unnecessary ORDER BY

Aggregates

- **Does the stream aggregate also cause an injected sort?**
- **Is the hash aggregate associated with concurrency issues due to memory grants?**
  - Can sorting the input help remove this requirement?

Predicates

- **Seek Predicate**
  - Used in actual index seek operation
    - Leveraging index keys

- **Predicate (Residual Predicate)**
  - Search condition that isn't SARGable – so it remains as an extra predicate
    - For Merge Join: check for "Residual" keyword
    - For Hash Match: check for "Probe Residual"

Spool Overhead

- *Might* be an optimization when used
- Index spools may point to a need for an index
- Might see them with remote tables (linked servers / distributed queries) for local access

Parallelism Performance Aspects

- **Not inherently bad or good**
  - Context matters, for example OLTP vs. DSS
- **Indicates higher cost query, so that should attract attention**
- **Watch for memory pressure**
  - Increased requirements for parallel operations

Data Modification Plan Patterns

- **Not common to troubleshoot data modification plans purely on the data modification plan shape itself**
  - Usually other issues in play, such as transaction duration, row-by-row coding, concurrency, transaction logging overhead

- **Plan shapes:**
  - Narrow plans - also called a “per row” plan
  - Wide plan – also called “per index” plan

Cardinality Estimate issues

- **Major red flag to watch for**
  - Skewed estimated rows vs. actual rows
- **Magnification and distortion as we move through the plan tree**
- **Several reasons why this could occur and I cover them in depth in the following Pluralsight course:**
  - SQL Server: Troubleshooting Query Plan Quality Issues (http://bit.ly/WRwSpD)

Data Type Conversions

- **Explicit = CONVERT or CAST**
- **Implicit data type conversion, for example joining two table columns with different data types**
  - Data type with higher precedence “wins”
- **Will can find CONVERT_IMPLICIT in a plan**
  - You can parse plans from DMVs
  - Not always surfaced!

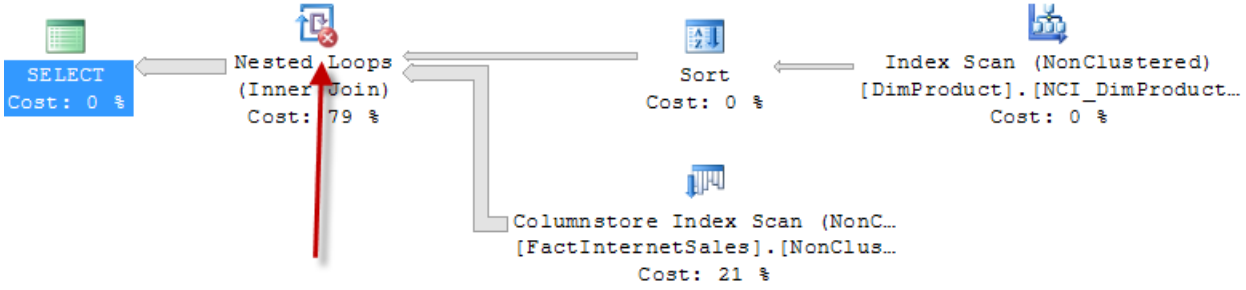
PlanAffectingConvert

- Introduced in SQL Server 2012
- Warn when a convert might have a poor impact on plan choice - "Cardinality Estimate" or "Seek Plan"

NoJoinPredicate Warning

- Warning of missing join predicates

```
SELECT      p.ProductLine,
            f.SalesAmount
FROM [dbo].[FactInternetSales] AS f,
     [dbo].[DimProduct] AS p
ORDER BY p.ProductLine;
```




```
<Warnings NoJoinPredicate="true" />
```

ColumnsWithNoStatistics Warning

- **Missing column statistics**
  - Check for disabled auto-create statistics

Table Scan
[DimProduct] [p]
Cost: 0 %

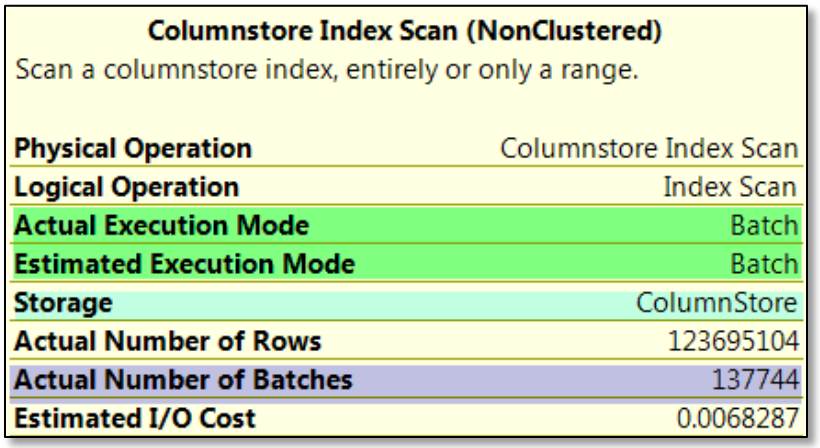
```
<Warnings>
   <ColumnsWithNoStatistics>
<ColumnReference Database="[BigFactTable]" Schema="[dbo]" Table="[DimProduct]"
Alias="[p]" Column="ProductKey" />
<ColumnReference Database="[BigFactTable]" Schema="[dbo]" Table="[DimProduct]"
Alias="[p]" Column="ProductLine" />
   </ColumnsWithNoStatistics>
</Warnings>
```

Parameter Sniffing

- **The plan contains valuable information on compiled vs. runtime value**
  - Parameter sniffing can be good OR bad
  - If parameter sniffing is an issue, validate:
    - ParameterCompiledValue
    - ParameterRuntimeValue
  - Then test cold-cache or recompiled versions using separate values to see the plan shapes generated

Columnstore Index Execution Mode

- **Testing the benefits of columnstore indexes?**
  - Check the execution mode for “batch” vs. “row”



Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Storage	ColumnStore
Actual Number of Rows	123695104
Actual Number of Batches	137744
Estimated I/O Cost	0.0068287

  - SQL Server Columnstore Index FAQ, http://bit.ly/wWZVBE

Course Summary

- Query plan analysis is a critical activity used to more efficiently tune poorly performing queries
- Query plans tell *part* of the overall story – so be sure to make use of other diagnostic tools as well such as SET STATISTICS IO and SET STATISTISTICS TIME
- Throughout this course you've learned how to capture query execution plans, and learned how to interpret plans and also build pattern recognition for noteworthy plan shapes that can inform your next performance tuning steps
- Thanks for watching!