SQL Server: Query Plan Analysis

Module 4: Common Operators

Joe Sack

Joe@SQLskills.com

Module Introduction

- **This module will review the more common operators that you may expect to see in a query execution plan**

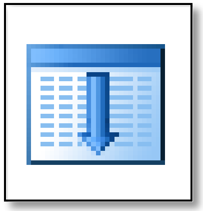
- **Don’t believe statements that specific operators or behaviors translate to an absolute problem**
  - I’ll call out things to watch for in the next module, but again, see them as areas of investigation

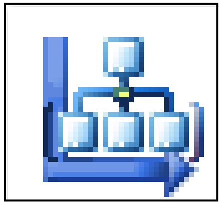
Table and Index Scans

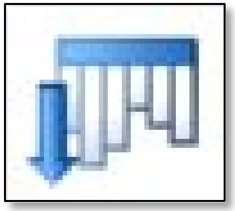
- **Table scan**
  - Indicating a retrieval of ALL rows from a table without a clustered index

- **Clustered Index scan**
  - Indicating a retrieval of ALL rows from a table with a clustered index

- **Columnstore Index Scan**
  - New as of SQL Server 2012, columnar storage index



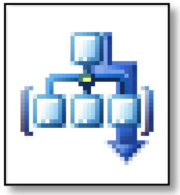


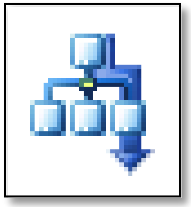


Index Seeks

- **Clustered Index Seek**
  - Retrieving rows based on a SEEK predicate from clustered index

- **Nonclustered Index Seek**
  - Same, but from a nonclustered index

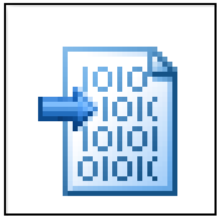


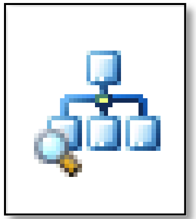


Lookups

- **Key Lookup**
  - Bookmark lookup on table with clustered index (always via Nested Loop)

- **RID Lookup**
  - Is just a bookmark lookup to a heap (using the RID)
  - Just like with Key Lookups, you’ll only see this with Nested Loop Joins





Join Considerations

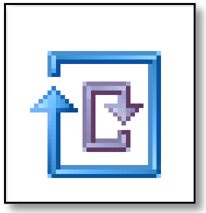
- **Beware of advice telling you that specific join types (or operators, for that matter) are “good” or “bad”**
- **Key factors:**
  - Tables to be joined, order of joins, algorithm used, cardinality
- **Join hints and/or forcing order = red flag**
  - Generally, “edge” cases or extreme tuning scenarios require their use
  - Otherwise, ask questions and find out why this is happening

Outer/Inner Terminology

- **When it comes to joins, you may hear the “outer” vs. “inner” table terminology**
  - But it is NOT related to the order you write them in your query, unless you’re forcing it
  - Outer = top = left
    - Nested Loop: for each row in outer, find all rows in inner
    - Merge Join: inner/outer – not as important (will discuss why)
    - Hash Join: outer table is the “build” hash table
  - Inner = bottom = right
    - Hash Match: inner table is probe table

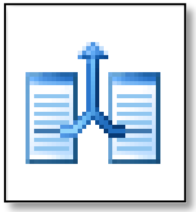
Nested Loop

- **Supports:**
  - Inner join, left outer join, left semi join, left anti semi join, cross join, cross apply, outer apply
- **Algorithm:**
  - For one row in the outer (top) table, find matching rows in the inner (bottom) table and return them
  - After no matching rows on the inner table are found, retrieve the next row from the outer (top) table and repeat until end of outer (top) table rows



Merge Join

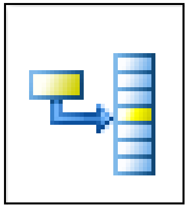
- **Logical operations:**
  - Requires one equijoin predicate
    - Except for full outer join transformation in many-to-many scenario
- **Joins two inputs (sorted on joining columns)**
  - Pre-existing sorting (via index) is ideal, but sorts can be automatically added (noteworthy if you see this)
- **Algorithm:**
  - Retrieve row from outer and inner tables
  - If a match: return the row
  - If no match: get a new row from the smaller input and iterate

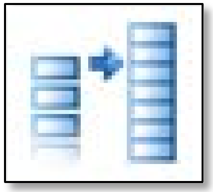


Hash Match Join

- **Requires one equijoin predicate**
- **Joins can be on unsorted inputs**
- **Algorithm:**
  - Build a hash table (hash buckets) via computed hash key values for each row of the “build” input (top/outer table)
    - For each probe row (bottom/inner table), compute a hash key value and evaluate for matches in the “build” hash table (buckets)
      - Output matches (or output based on logical operation)

- **Build Hash**
  - Build of a batch hash table for a columnstore index





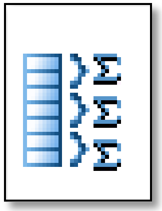
Filter

- **Predicates can be evaluated within operators that read data from table/indexes**
- **Query Optimizer aims (when possible) to "push" filter down the tree (leaf level) to reduce rows moved**
- **Sometimes a predicate cannot be pushed and you'll see a Filter operator instead**
- **When you see these, take note of where they are happening**
  - Late in the data flow can translate to higher overhead as the operators pull data

Stream Aggregate

- **Groups rows by one or more columns**
  - Calculates one or more aggregate expressions
    - Does this one group at a time
  - Can do:
    - Scalar aggregates (no GROUP BY) e.g. SUM
    - Group aggregates (have GROUP BY)
      - Requires ordered (sorted) input for grouping columns
      - If unordered then Query Optimizer can add a Sort operator

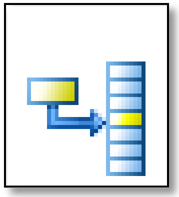
- **Non blocking (streams)**
- **Minimal memory**



Hash Match (Aggregate)

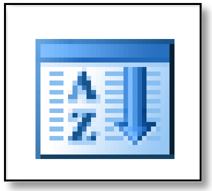
- **Several aspects of hash join apply to hash match**
  - Requires memory for hash table
  - Unsorted inputs are okay
  - Is stop-and-go on the build table

- **For hash match, hash table generated for groupings of rows**
  - Hash table values based on grouping columns
    - 1) Generate hash
    - 2) Check for existing row in hash table
    - 3) Generate row if no match or update matching row



Sort

- **As named: orders rows received from input**
  - For example, due to ORDER BY in query
- **Variation is “Distinct Sort” to remove duplicates**



Spools

- **Eager Spool**
  - Takes entire input, placing row(s) in hidden tempdb work file
  - When spool's parent operator asks for first row, spool grabs all rows from the input and stores them in tempdb

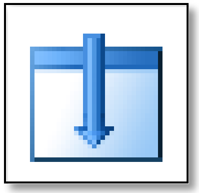
- **Lazy Spool**
  - When the lazy spool's parent requests a row, the spool grabs the row from the input operator and stores it in the tempdb spool table
    - It does not retrieve all rows like the eager spool
  - Lazy spools are non-blocking

Eager Spool and “Halloween Protection”

- **Identified by IBM researchers back in 1976**
  - Performing an update involved, conceptually a read cursor and a write cursor
  - The write cursor was updating the read cursor, causing a moving target of re-updates to the same set of rows
- **Eager spools, when needed, prevent the write cursor from impacting the read cursor**
  - Example of when its not needed?
    - If you’re NOT updating the index key itself (causing movement that could make rows scanned > 1)
    - Other blocking operators already in use (such as Sort)

Constant Scan

- **Introduces >= 1 constant rows that can be built upon by parent operators**
  - You can see this in data modification plans as well as SELECT plans
- **SQL Server 2005: seen with partitioning as well, defining applicable partition numbers (driven by predicates)**
  - Nested Loop operator joining Constant Scan (outer) and partitioned table (inner)

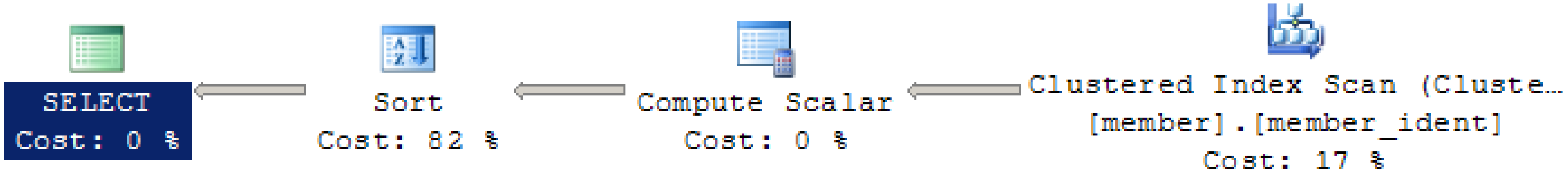


Assert

- **Verifies existence of a specific condition**
    - □ Check constraints
    - □ Referential integrity
    - □ Enforce return of one-row for scalar sub-query

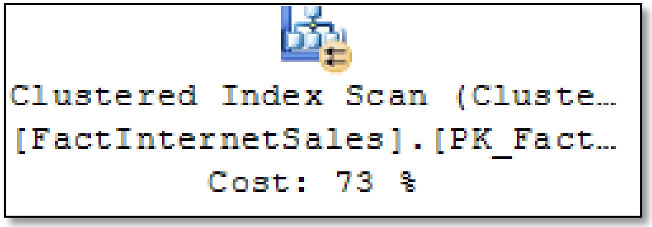
Compute Scalar

- **Evaluates an expression, producing a scalar value (e.g. GETDATE() value)**
- **Estimated CPU cost is often low – for example 0.001 for an estimated 9,615 rows passing through an UPPER function**
    - May be a placeholder definition of an expression calculated in another operator – for example…
        - Compute Scalar
            - [Expr1003] = Scalar Operator(upper([Credit].[dbo].[member].[lastname]))
        - Sort
            - [Credit].[dbo].[member].member_no, [Credit].[dbo].[member].curr_balance, Expr1003



Identifying Parallelism in the Plan

- Parallelism operators (Distribution/Repartition/Gather)
- You’ll also see the parallelism icon in the graphic for operators that can run in parallel or serial modes



```
Clustered Index Scan (Cluste…
[FactInternetSales].[PK_Fact…
Cost: 73 %
```

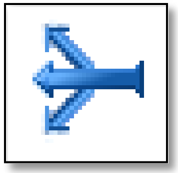
- If looking at XML Showplan you’ll see RelOp:
    - Parallel="true"

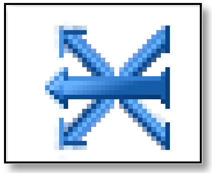
Exchange Operators

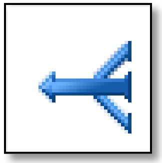
- **Distribute Streams**
  - Takes one input and produces multiple output data streams

- **Repartition Streams**
  - Takes in multiple streams and then produces multiple streams
    - Can be used in conjunction with bitmap filter to reduce rows in output
    - May “rebalance” thread skew

- **Gather Streams**
  - Operator takes in multiple streams and produces a single stream

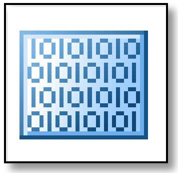






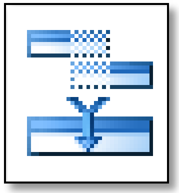
Bitmap

- **Performs bitmap filtering for parallel plans with hash or merge joins**
  - □ Optimization that eliminates rows with key values that wouldn’t produce join records
  - □ Reduces rows being passed to parent operators



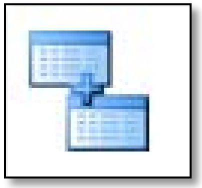
Merge Interval

- Merges multiple potentially overlapping intervals used in predicates
- Preceded by compute scalar and constant scan
- Goal is to minimize redundant scans



Concatenation

- Scans multiple inputs, returning each row
- Seen via UNION ALL



Segment and Sequence Project

- **Segment divides input rows into related segments based specific columns**
  - □ Seen with windowing functions
  - □ Columns used for segmenting shown in argument properties
  - □ Extra segment column created which tracks if value has changed from the previous row

- **Sequence Project adds columns to perform computations on an ordered set**
  - □ Outputs one segment at a time