The World's Largest Community of SQL Server Professionals

Introduction to Graphical Execution Plans in SQL Server 2005/2008

Brad M McGehee
Director of DBA Education
Red Gate Software
www.bradmcgehee.com

My Assumptions About You

- You are a SQL Server DBA or developer with novice to intermediate knowledge of SQL Server.
- You have a basic understanding of indexing and Transact-SQL.
- You have little or no experience using Execution Plans.

Our Focus Today

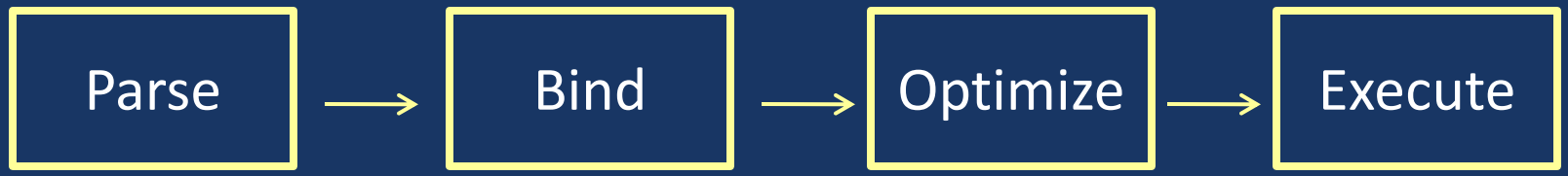
- The study of execution plans is a huge topic, one that we can't cover entirely in a short session.
- The goal of this session is to provide an easy introduction to:
  - Why execution plans are important
  - How executions plans are created
  - Learn about different type of execution plans
  - Learn the difference between estimated and actual execution plans
  - Learn different ways to capture execution plans
  - Learn the basics of how to read & interpret graphical execution plans
- After today's session, I highly recommend you read Grant Fritchey's free eBook on *SQL Server Execution Plans*.

Why are Execution Plans Important?

- Simply put, an execution plan describes the data retrieval/storage methods chosen by the Query Optimizer to execute a specific query.
- The Query Optimizer's goal is to identify the least expensive way to execute any particular query. This is because the lower the cost to execute a query, the faster it runs.
- So why is it important to be able to read execution plans? Because you want to answer this question:
  - Why are my queries running slow, and what can I do to boost their performance (reduce the query's cost)?
  - The answer: By understanding how to read execution plans, you can help identify why a particular query is running slower than expected, and to help fix it.

How Execution Plans are Created

1. Query is sent to the query optimizer.
2. The query is parsed to check if it is written correctly.
3. The query then goes through a binding process, where validation steps occur, which resolves all the names of the various objects, tables and columns, among other tasks.
4. The query then goes through the optimization process, where different execution plans are explored, and then one is selected based on low cost. (More on next slide)
5. Then the query is executed by the query execution engine.



How Does Optimization Work?

1. If a valid plan is cached, use cached plan, otherwise, next step.
2. Is the query a trivial plan? If yes, then run trivial plan, otherwise next step.
3. Attempt to simplify the query, if plan is very low cost, then run it, otherwise proceed to formal query optimization.
4. Begin formal cost-based optimization process:
    1. Apply basic optimization rules, if plan cost is less than .2, use plan.
    2. Apply more advanced rules, if plan cost is less than 1, use plan.
    3. If plan cost is over 1, consider parallelism (if applicable), compare cost of serial and parallelism plans, then send least cost plan to next step.
    4. Explore even more options, but after a limited time, stop trying and use least expensive plan found up to this point in time.

Execution Plans Are Cached

- Once an optimized execution plan has been created for a query, it is generally stored in the plan cache, and then executed. Exceptions include trivial and DDL plans.
- When the same query is run again, hopefully it can be found in the plan cache and reused.
- Reusing a cached execution plan can save time because a new execution plan does not have to be recreated each time the same query is re-executed. [This, of course, is why SQL Server caches execution plans in the first place.]
- Plan caching is not perfect, but that's the subject of a different talk.

Execution Plans Come in Different Forms

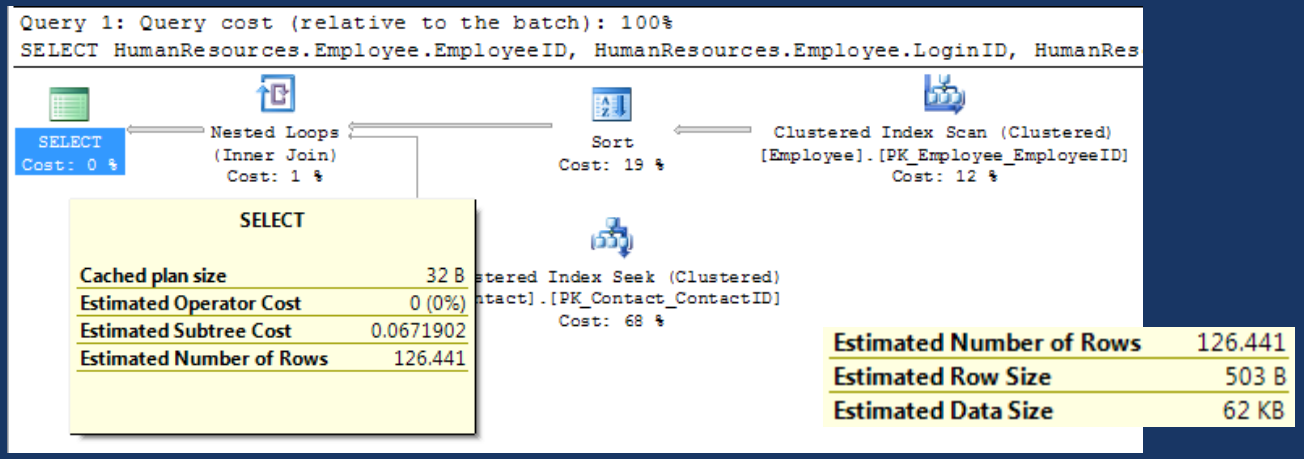
- Text (Deprecated)
- XML (A Storage Format)
- Graphical (A Display Format)
- Demo

Actual vs. Estimated Execution Plans

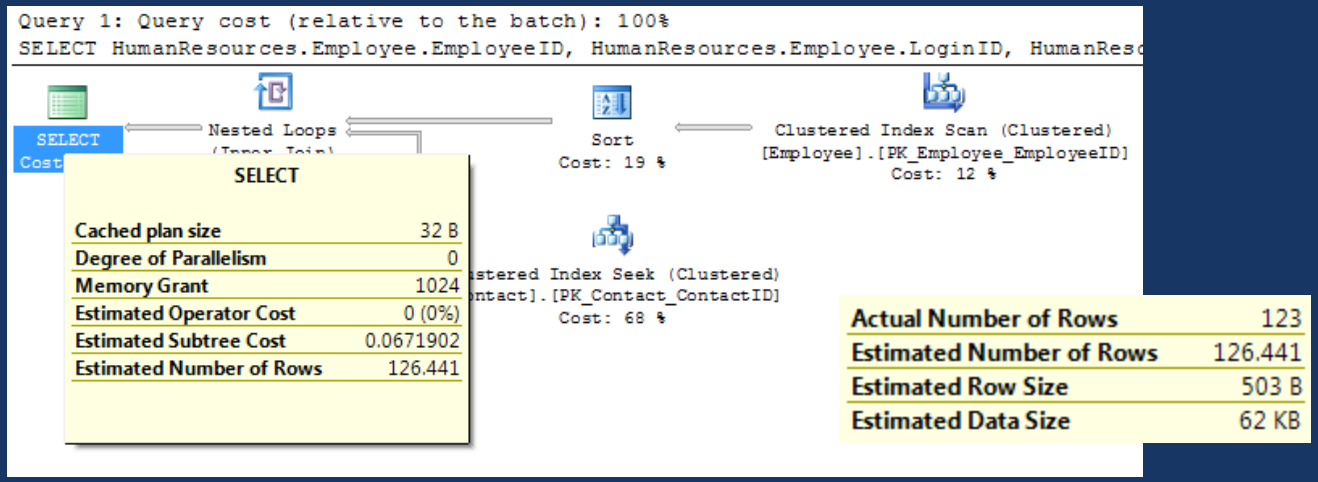
- Actual Execution Plan
  - Produced after a query actually runs.
  - Displays a combination of estimated and actual results.
- Estimated Execution Plan
  - Produced without running the query.
  - Estimated execution plans have some disadvantages:
    - Can't be used if the query creates objects it needs to use: i.e. temp table.
    - Displays estimated data only (based on existing index and column statistics).
    - Will not reflect parallel query plans correctly.
    - Not guaranteed to represent actual query plan. For example, the optimizer might force a statistics update when a query is executed, which might change the actual plan substantially.
  - Why use an estimated execution plan?
    - Can save time and resources when testing long running queries.
    - Allows you to explore data modification queries without modifying data.
- Demo

How Actual & Estimated Plans Vary

Estimated



Actual



How to Capture/Produce Execution Plans

- SSMS (our choice for today)
- Profiler (Showplan XML Event)
- SQL Server 2005 Performance Dashboard
- SQL Server 2008 Data Collector
- DMVs (requires some fancy queries)
- SET commands (shown in previous demo)

Graphical Execution Plan Basics

- Demo
  - Overview of Graphical Execution Plan Screen
  - Using the Zoom Button (Around, In, Out, Fit)
  - Learning to Read from *Right to Left* and from *Bottom* to *Top*
  - Learning about Operators
  - Learning about Tool Tips (and Properties)
  - Learning about Arrows
  - Executing Multiple Queries and Viewing Multiple Execution Plans

Execution Plans are Made Up of Operators

- Each operator receives rows, performs some operation on them, then sends the rows to another operator. For example:
  - Scanning data from a table
  - Seeking data in a table
  - Sorting
  - Joining
- In total, there are 79 different operators that can be included in an execution plan.

Graphical Execution Plan Operators

	Arithmetic Expression
	Assert
	Bitmap
	Bookmark Lookup
	Clustered Index Delete
	Clustered Index Insert
	Clustered Index Scan
	Clustered Index Seek
	Clustered Index Update
	Collapse
	Compute Scalar
	Concatenation
	Constant Scan
	Delete
	Deleted Scan

	Eager Spool
	Filter
	Hash Match
	Hash Match Root
	Hash Match Team
	Insert
	Inserted Scan
	Iterator Catchall
	Lazy Spool
	Log Row Scan
	Merge Interval
	Merge Join
	Nested Loops
	Nonclustered Index Delete
	Nonclustered Index Insert

Graphical Execution Plan Operators

	Nonclustered Index Scan
	Nonclustered Index Seek
	Nonclustered Index Spool
	Nonclustered Index Update
	Online Index Insert
	Parameter Table Scan
	Remote Delete
	Remote Insert
	Remote Query
	Remote Scan
	Remote Update
	RID Lookup
	Row Count Spool
	Segment
	Sequence

	SequenceProject
	Sort
	Split
	Spool
	Stream Aggregate
	Switch
	Table Delete
	Table Insert
	Table Scan
	Table Spool
	Table Update
	Table-valued Function
	Top
	UDX
	Update

Graphical Execution Plan Operators

Icon	Language element
	Assign
	Convert
	Declare
	If
	Intrinsic
	Language Element Catchall
	Result
	While

Icon	Cursor physical operator
	Cursor Catchall
	Dynamic
	Fetch Query
	Keyset
	Population Query
	Refresh Query
	Snapshot

Icon	Parallelism physical operator
	Distribute Streams
	Repartition Streams
	Gather Streams

Common Operators

- SELECT
- Clustered Index Scan
- Clustered Index Seek
- Non-Clustered Index Scan
- Non-Clustered Index Seek
- Key Lookup
- Sort
- Joins (loop, merge, hash)

Demo

- Let's see a simple example of how execution plans, and our knowledge of SQL Server, can be used to optimize a poorly written query.

SELECT

- Represents the end results of a SELECT query.
- Specifies the type of optimization done on the query.
- Good place to start when evaluating most execution plans. Check to look for number of rows returned.
- To optimize performance, the number of rows that are returned should be the minimum number of rows necessary to meet the needs of the query.

Clustered Index Scan

- A clustered index scan is a scan of the rows of a table that has a clustered index.
- Clustered index scans can be slow and use up lots of server resources (for large tables).
- If you see a clustered index scan, you should investigate to see why it is occurring. It is a hint that a query may be having performance problems. Perhaps the WHERE clause is not restrictive enough, or not sargable.
- On some occasions, when tables are small or many rows are returned, then a clustered index scan might be the fastest way to return data, so they are not always bad for performance.

Clustered Index Seek

- If there is an available and useful index, and there is a sargeable WHERE clause, the query optimizer can very quickly identify the rows to be returned and return them without having to scan each row of the table.
- Generally speaking, if you see a clustered index seek in an execution plan, this is a good thing.
- One possible exception to this is if the clustered index seek is repeated over and over again. Check the operator's "Number of Executions" to find out.

Non-Clustered Index Scan

- Rows in a non-clustered index are scanned, and all rows that match the WHERE clause are returned.
- As with all scans, it can be slow and require extra I/O resources.
- Generally, seeing a non-clustered index scan should be seen as a hint of potential performance problems.
- Sometime, these scans can be turned into seeks if you modify the WHERE clause so that it is more restrictive or is sargable.

Non-Clustered Index Seek

- A non-clustered index is used to identify exactly which row(s) are to be returned, so every row does not need to be scanned (assumes sargeable WHERE clause).
- This is generally much faster than a non-clustered index scan.
- Like clustered index seeks, non-clustered index seeks are generally a good thing.
- One exception is if bookmark lookups occur as part of the non-clustered index seek, then performance may lag if many rows are returned.

Key Lookup

- A key lookup is a bookmark lookup on a table with a clustered index.
- While key lookups are often faster than "scans," this is not always the case. If the query returns a small number of rows, a key lookup is probably OK.
- But if many rows are returned, then it may cause a performance problem that needs correction.
- Often, key lookups can be eliminated with the addition of a covering/included index.

Sort

- Sorts occur when you specify that returned data be ordered, or because the query optimizer needs to sort data internally to produce the desired results.
- Sorts are normal and aren't generally a problem.
- But if you return too much data, then sorts may take a lot of resources (including tempdb), and you should identify ways of reducing the number of rows returned to boost query performance.



Joins (loop, merge, hash)

- The nested loop join compares each row from one table (the “outer table”) to each row from the other table (the “inner table”), looking for rows that satisfy the join predicate.
- The merge join works by simultaneously reading and comparing the two sorted inputs one row at a time. For each step, it compares the next row from each input. If the rows are equal, it outputs a joined row and continues. If the rows are not equal, it discards the lesser of the two inputs and continues.
- The hash join algorithm executes in two phases known as the “build” and “probe” phases. During the build phase, it reads all rows from the first input, hashes the rows on the equijoin keys, and creates or builds an in-memory hash table. During the probe phase, it reads all rows from the second input (often called the right or probe input), hashes these rows on the same equijoin keys, and looks or probes for matching rows in the hash table.

Joins

- There is no “ideal join,” it all depends on the data being joined.
- From a resource perspective, a nested loop join generally uses fewer resources, and seeing one generally is often an indicator of good overall performance. Usually best for smaller joins.
- Merge joins are often used for moderate to large data sets, and are most efficient if the joined columns are pre-sorted, otherwise they have to be sorted before the merge join can occur.
- Hash joins are often used when very large data sets are used. Hash joins parallelize and scale better than other joins and are good at minimizing response times for OLAP queries. Seeing one may be a hint that too much data is being returned, especially in OLTP applications.

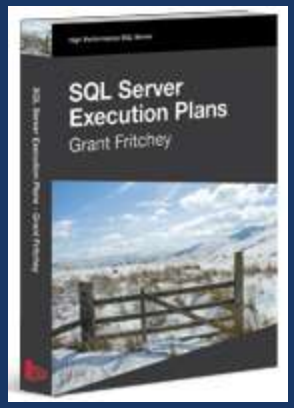
Take Aways From This Session

- This is just a small sample of the things you can do with Graphical Execution Plans.
- Graphical Execution Plans are a powerful tool to help DBAs understand how a query executes.
- The learning curve for learning about Graphical Execution Plans is high, but worth the effort.
- Based on the information provided by an execution plan, and the DBA's knowledge of how SQL Server works, many queries can often be optimized to perform better.

Q&A

- Please ask your questions clearly and loudly.
- If you don't get your questions answered now, see me after the session, or e-mail me.

Find Out More



Free E-Books:

- www.sqlservercentral.com/Books

Check these out:

- www.SQLServerCentral.com
- http://www.simple-talk.com/sql/performance/graphical-execution-plans-for-simple-sql-queries/

Contact me at:

bradmcgehee@hotmail.com

Blogs:

www.bradmcgehee.com

www.twitter.com/bradmcgehee

Click Here for a free 14-day trial of the Red Gate SQL Server Toolbelt