



*The World's Largest Community
of SQL Server Professionals*

An Introduction on How to Interpret Query 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 want to learn the basics of how to read and interpret Execution Plans so that you can better understand how your queries are executing, so that you can figure out better ways of how to optimize them.

The difference between a *run-of-the-mill* DBA/Developer from an exceptional DBA/Developer is that the *exceptional* DBA/Developer thoroughly understands how to use their available tools to their full potential.

Why are Execution Plans Important?

- A execution plan, simply put, **describes the data retrieval/storage methods** chosen by the Query Optimizer to execute a specific query.
- So why is this important to know?
- If you understand how a query was executed by SQL Server, you have the information you need to help you to figure out ways to **potentially** optimize it.

How Execution Plans are Created (1)

- T-SQL is sent to the relational engine.
- DDL code does not need to be optimized, as there is only one way to perform the task.
- T-SQL is parsed to check if it is written correctly.
- DML T-SQL is next run through the algebrizer, which resolves all the names of the various objects, tables and columns.
- Algebrizer output is sent to Query Optimizer.
- Trivial plan may be generated.
- Otherwise, cost-based calculations are performed to identify the least resource intensive way to execute the query.
- Index and column statistics are used during the above process to help the query optimizer determine the least resource intensive execution plan. The better the statistics, the more efficient is the execution plan.

How Execution Plans are Created (2)

- The creation of an execution plan takes time.
- Not every potential execution plan option is always explored, a “good enough” execution plan is often generated, then sent to database engine for execution.
- The execution plan is estimated, and may change when the T-SQL is actually executed.
- Execution plans may be cached (in the plan cache) for later use so they can be reused if an identical (or similar) query is submitted for execution again.
- 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.

Three Types of Execution Plans

- Text (Deprecated)
 - Harder for some people to read
- XML (Storage Format)
 - XML code is not really designed to be read directly by DBAs
 - Can be saved and shared with others (portable)
 - Can be displayed in graphical format
- Graphical (Display Format)
 - Generally easier to read and understand by beginners
 - Is produced from XML execution plan
 - Our focus today
- Demo

Actual vs. Estimated Execution Plans

- Estimated
 - Produced without running the query
 - Displays estimated data only
 - Can save time and resources
 - Can't be used if the query creates objects it needs to use: i.e. temp table
 - Not always accurate, can be misleading, especially if data is skewed, or the statistics are outdated, or parallelism is involved
- Actual
 - Produced after a query actually runs
 - Displays a combination of estimated and actual results
- Demo

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
- SET commands (shown in previous demo)

How to Read Graphical Execution Plans

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

Execution Plans are Made Up of Operators
















- Each operator implements a single basic operation, such as:
 - Scanning data from a table
 - Seeking data in a table
 - Filtering
 - Sorting
 - Joining two data sets
- In total, there are 79 different **operators** that can be included in an execution plan, and they are represented by **graphical icons**.

Graphical Execution Plan Icons

	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 Icons




	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 Icons

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

Icons/Operators Covered Today

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



SELECT

- Represents the end results of a SELECT query.
- 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.
- Good place to start when evaluating any execution plan.



Table Scan

- A table scan indicates there is no clustered index on the table, and the table is a heap.
- A table scan indicates that every row in the table (heap) had to be examined to see if it met the query criteria, which can mean slow performance if there are a large numbers of rows.
- Generally, heaps should be avoided.
- In most cases, you will want to add a clustered index to every table, as it has the potential of boosting the performance of the query, even if a clustered index scan is still conducted. This is because leaf nodes of the clustered index are stored together (logically, and hopefully physically). In a heap, they are not, which can hurt performance.



Clustered Index Scan (Clustered)

Clustered Index Scan

- A clustered index scan is a full or (partial scan) of all the rows of a table that has a clustered index.
- Like a table scan, clustered index scans can be slow and use up lots of server resources.
- Generally, clustered index scans should be avoided, but they are almost always better than table scans.
- 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.



Clustered Index Seek (Clustered)

Clustered Index Seek

- If there is an available and useful index, and there is a sargeable WHERE clause, the query optimizer can usually, very quickly, identify the rows to be returned and return them without having to scan each row of the table.
- In many cases, for best query performance, clustered index seeks should be as used often as feasible.



Index Scan (NonClustered)

Non-Clustered Index Scan

- All records (sometimes a range) in the table 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, non-clustered index scans should be avoided.
- Sometime, these scans can be turned into seeks if you modify the WHERE clause so that it can make more effective use of the index.



Index Seek (NonClustered)

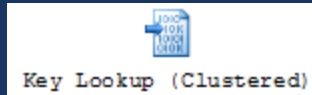
Non-Clustered Index Seek

- A non-clustered index is used to identify the row(s) 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.



RID Lookup

- A RID Lookup is a form of a bookmark index lookup on a table without a clustered index (a heap).
- While RID Lookups are often faster than most “scans,” this is not the case if many rows have to be returned.
- Generally, RID Lookups should be eliminated with the addition of an appropriate clustered index, and if necessary, a covering or included index.



Key Lookup

- A key lookup is a bookmark lookup on a table with a clustered index.
- While key lookups are often faster than most “scans,” this is not always the case.
- Generally, key lookups should be eliminated with the addition of a covering or included index.



Sort

- Sorts occur when you specify that returned data be ordered.
- 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.



Joins (loop, merge, hash)

- The **nested loop** join compares each row from one table (known as the “outer table”) to each row from the other table (known as 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. At 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 (loop, merge, hash)

- SQL Server can use one of three different types of joins.
- There is no “ideal join,” it all depends on the data being joined.
- From a resource perspective, a nested loop join uses fewer resources, and seeing one generally is a good indicator of good overall performance. Often best for smaller joins.
- Merge and hash joints use more resources and may be an indicator that too much data is being returned.
- Hash joins use the most resources.
- Use can experiment with hints to see which join type is ideal for your query, although the query optimizer will generally select the best join type.
- Demo

Take Aways From This Session

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

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/>

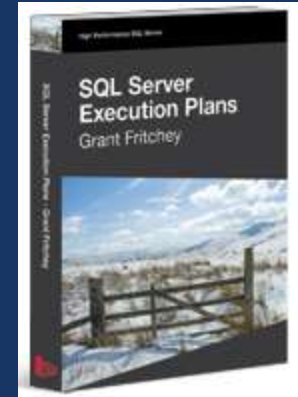
Contact me at:

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](#)