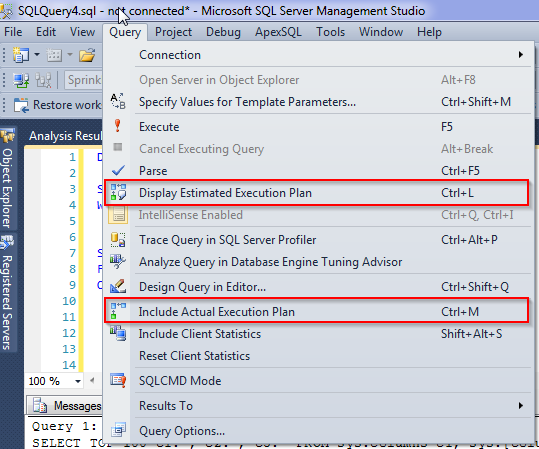
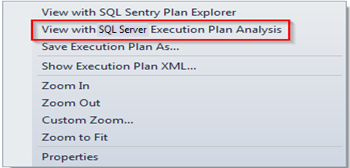
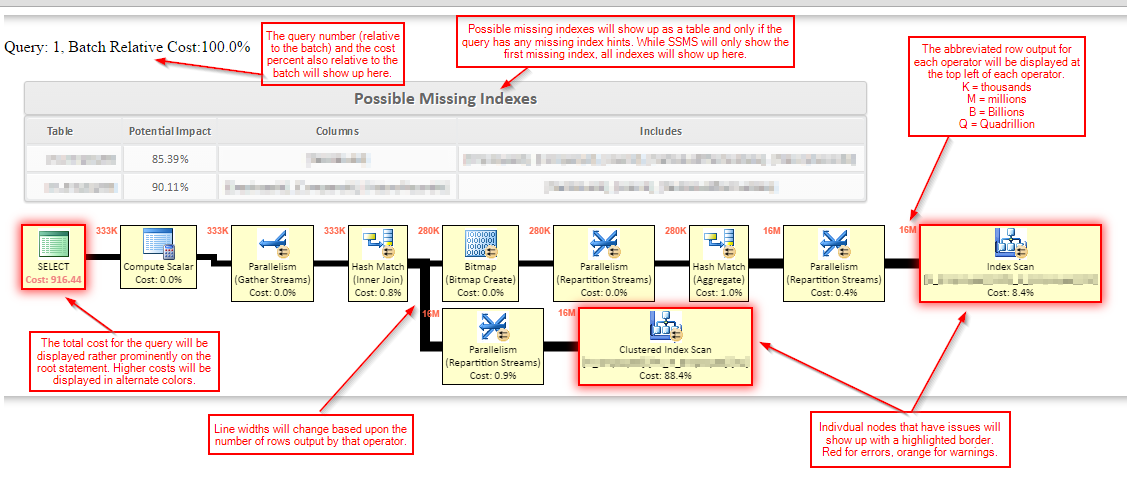
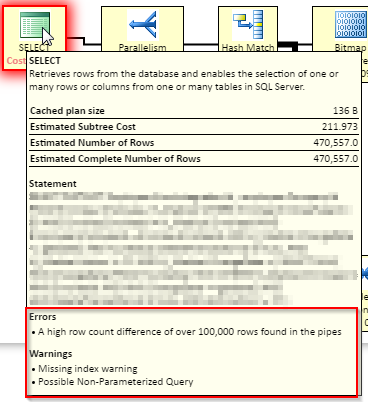
SQL Server Execution Plan Analysis

Purpose: The SQL Server Execution Plan Analysis product provides an easy to use and read graphical representation of a query plan with all of the potential issues highlighted. It is runnable either as a standalone application as well as a SQL Server Management Studio (SSMS) Add-in. It makes it easy for both the expert and the novice to be able to read an execution plan with ease. This should help elevate the level of T-Sql written for SQL Server.

NOTE: Not all issues can always be fixed for various reasons. However, it is important to understand this and try to fix everything that you can.

# Add-in Usage

1. Install the Add-In from the MSI.
2. Open up a query that you wish to optimize, or open an existing plan in SSMS.
3. Turn on either “Include Actual Execution Plan” or click “Display Estimated Execution Plan”. Of note is that SQL Server uses the estimated plan when processing queries.  
   
4. After the Execution Plan displays in the results window, you can right click the plan and select: “View with SQL Server Execution Plan Analysis”. This will analyze the plan for any issues and launch the new view of the plan into a browser window.  
   
   1. When launched in IE security will block the JavaScript from running and you will have to click the “Allow blocked content” button to allow the JavaScript to work.  
      
   2. IE does not properly display very large plans as there is a canvas width/height limitation that differs from Chrome and Firefox. This means that the lines may get cut off in IE with large plans. However it will display small to medium plans quite well.
5. When the plan is launched in the browser you will see something similar to this.   
   
6. Hovering over any of the operators will give you almost the exact same information you will find in the SQL Server Management Studio. When you hover over one of the errors or warnings you will also get the added information of what errors and warnings were violated.  
   

# What’s Checked

* ImplicitConversion: Implicit conversions occur when SQL Server has to automatically convert one datatype to another during processing. This causes row by row (RBAR) processing and negates the use of any indexes. Possible causes:
  + A value used in a where clause or join clause has a different data type than the column used in the comparison.
* MissingIndex: Missing index hints, are suggested by the query engine as a possible solution to speed up that ONE query. These hints should be approached with caution and a lot of knowledge as the hints suggested could make things worse. They should still be investigated however.
* KeyLookup, RIDLookup: A lookup occurs when and index was used, but not all of the columns were found in the index so SQL Server has to resort to looking up these column values in the main table. This lookup has to occur for each row returned from the index. Possible causes
  + Missing columns in existing index from include clause
  + No index
* ClusteredIndexScan, IndexScan: A scan occurs when a suitable index was found, but something caused a Seek to not be used. These are some factors that cause a scan:
  + Non-sargeable join or where clause
  + Aggregate functions
  + Implicit Conversions
  + Using LIKE with ‘%endswith’ or ‘%contains%’
  + A low row count. In this case SQL Server makes the decision to scan, no matter what you do.
* FatPipes: This occurs when you pull a large number of rows typically on the right side of the plan, but then filter down to a small number on the left or output side. This is easily remedied by limiting the row count through the life of the query. Possible causes:
  + Scan being used versus a seek (See Scan possible causes)
  + No Where clause
  + Invalid join or cross join
* MediumCostOperator: This is flagged when a query cost falls within the range of 200-1000
* HighCostOperator: This is flagged when a query cost exceeds a cost of 1000
* IndexSpool, TableSpool: Spools occur when SQL Server does not have enough memory to accommodate the amount of data in memory. So the data is spooled to tempdb. Possible causes:
  + Scans (See Scan possible causes)
  + Invalid joins or cross joins
  + Fat pipes
* HighDesiredMemory: A plan will request a large amount of memory at times. This is usually an indicator of an overly complex plan or FatPipes.
* NoLock: Nolock has quite few issues that it can cause, and should be used sparingly in very limited cases. Some causes of the Nolock hint:
  + Dirty reads
  + Missing rows
  + Reading rows twice
  + Reading multiple versions of the same row
  + Index corruption (when used in action queries)
  + Read errors
* Cursor: Cursors have a bad impact upon the SQL Server and should be avoided at all costs and if possible replaced with set based queries or a temp table that gets updated incrementally.
* HighCostSort: Sorts can be one of the most expensive operations performed in SQL Server. If at all possible, sorting should be offloaded to the execution layer.
* NoJoinPredicate: This is an indicator that a cross join was performed. Possibly from a malformed query or on purpose. If on purpose this can be ignored.
* CompilationTimeout, CompilationMemoryLimitExceeded: Both of these are indicators that the execution plan is way too complex and should be simplified if possible. Possible causes:
  + High number of joins
  + Nested views
  + Overly complex query
* HighNumberOfJoins: A high number of joins can cause SQL Server to have issues processing the query. One way to reduce this is to do a smaller set of joins into a temp table, then add more joins to the temp table as needed.
* JoinToTableValueFunction: Joining to a table valued function has several ramifications. Firstly, the function block parallelism. Secondly, functions always report bad row estimates which wreaks havoc with the optimizer causing a suboptimal plan.
* ColumnsWithNoStatistics: This warning is an indicator that a column was used in a clause and could benefit from having statistics but does not have any. This can occur if Auto Create Statistics is turned off or on a temp table.
* SpillToTempDb: Spills occur when SQL Server underestimates the number of rows that are being returned by an operator. This is typically an indicator of poorly maintained statistics.
* UnmatchedIndexes: This warning occurs when a filtered index is present, but SQL Server could not use the index because of parameterization.
* PossibleNonparameterizedQuery: This warning occurs when the query uses string or numeric literals in the where clause. The use of a lot of dynamic sql causes plan cache bloat and memory pressure leaving less memory for the buffer(data) pool. All query variables should be parameterized to avoid SQL Injection as well as performance issues for the SQL Server. More info:
  + [How to: Execute a Parameterized Query](https://msdn.microsoft.com/en-us/library/bb738521(v=vs.100).aspx)
  + [Parameterized IN clauses with ADO.NET and LINQ](http://www.mikesdotnetting.com/article/116/parameterized-in-clauses-with-ado-net-and-linq)

# Check (Possible) Fixes

* ImplicitConversion:
* MissingIndex:
* KeyLookup, RIDLookup:
* ClusteredIndexScan, IndexScan:
* FatPipes:
* MediumCostOperator:
* HighCostOperator:
* IndexSpool, TableSpool:
* HighDesiredMemory:
* NoLock:
* Cursor:
* HighCostSort:
* NoJoinPredicate:
* CompilationTimeout, CompilationMemoryLimitExceeded:
* HighNumberOfJoins:
* JoinToTableValueFunction:
* ColumnsWithNoStatistics:
* SpillToTempDb:
* UnmatchedIndexes:
* PossibleNonparameterizedQuery: