**Understanding a SQL Server Query Execution Plan**

## What is a Query Execution Plan?

To communicate with any RDBMS we use Structured Query Language (SQL), which is a declarative querying language and supported by the American National Standards Institute (ANSI). Microsoft extended ANSI SQL capabilities known as Transact-SQL (T-SQL). It is a SQL standard and adds improvements and capabilities, making T-SQL an efficient, robust, and secure language for data access and manipulation from Microsoft SQL Server.

A T-SQL statement (query) tells SQL Server what you want but does not tell it how to do. When SQL Server receives a query it applies a number of processes, which work on that query. The first two things the server does on any submitted query is parsing and optimization.

We will discuss query logical and physical processing stages in detail later in this article, to make sure we have a good understanding of a query execution plan but before that I want to highlight that once a query is parsed and then processed by the optimizer, the optimizer generates an execution plan.

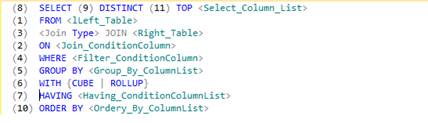
The generated execution plan, known as a Query Execution Plan is input for the SQL Server storage engine to tell it how to execute the query

An execution plan is the real work plan generated by the query optimizer to determine how to process or execute a submitted query. The plan is the primary method for any developer or DBA to troubleshoot any performance issue by a query; it can even identify which part of query is causing the issue.

The query has logical and physical processing stages, which helps the developer or DBA to understand what outcome a query execution result should produce and how SQL Server attains any query outcome.

## Logical Processing Stages

Logical processing stages decide the order in which the different clauses are logically processed. Below is the basic query syntax along with the logical order number in which it will be executed:

  
Basic Query Syntax

In this article, I will not be discussing the logical processing stages for table operators (APPLY, PIVOT and UNPIVOT), set operators (EXCEPT and INTERSECTION) and OVER clause as these are not ANSI operators.

In the above query you can observe that the first clause that is processed is the FROM clause while the clauses written in first row SELECT, DISTINCT and TOP are processed almost last. Each step generates a temporary virtual table (accessible only to SQL Server) that is used as the input to the following step. None of the intermediate virtual tables are available to the outer query, only the virtual table generated by the final step is returned to the caller.

## Physical Processing Stages

Understanding the way a database engine physically processes a query helps developers, DBAs and database architects to make good choices when designing the database schema and writing queries.

Whenever SQL Server gets a query to execute it performs two major steps to return the query output. The first step is query compilation, which generates a query execution plan by the SQL Server relational engine and the second step is execution of the query execution plan by the SQL Server storage engine.

Here, we can observe that during physical processing of a query, SQL Server gets help from the two component below:

* QL Server relational engine - responsible for generating the query execution plan.
* SQL Server storage engine - gets query execution plan as input and takes action upon it to return the desired result.

Further, we will learn how both engines above (relational and storage) help SQL Server to execute any T-SQL query.

## Query Execution Plan Generation

In the relational engine, a query is parsed and then processed by the query optimizer, which generates an execution plan. When any query reaches SQL Server, the first place it goes to is the relational engine. Here, the query compilation process happens in three phases; Parsing, Binding and Optimization.

Parsing, is a process to check the syntax--whether a query is written correctly or not. It doesn’t check whether a column used in a WHERE clause exists in any specified table in the FROM clause. The output of this process is a parse tree or sequence tree. Parse tree works as input for the next process and contains logical steps to execute the query.

Binding, is a process done by algebrizer. It checks weather query semantics are correct or not; for example, whether the specified two tables joined in the FROM clause are really tables or not. The algebrizer produces a query processor tree, which works as input for query optimizer.

Optimization is the last step in the compilation process and generates the query execution plan. The optimizer attempts to determine the most efficient way to execute a given query. The optimizer always tries to find the way to process/execute a query in the minimum amount of time; it compares several options, with putting all possible permutations and combinations to generate the most effective query execution plan in a reasonable time, which does  not deviate much from the best possible query execution plan.

A generated query execution plan by the optimizer is also known as a cost-based plan, because the optimizer makes the decision by calculating the cost of the execution plan for the query, the cost in terms of the required CPU processing and I/O and how fast it will execute.

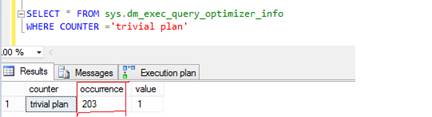
The query optimizer evaluates the resource consumed by various query plans and usees this as the basis for plan selection. It assigns an estimated "cost" to each possible query plan, and chooses the plan with the smallest cost, i.e. the plan the optimizer thinks will execute the query as fast as possible and use the least amount of resources, CPU and I/O. Sometimes, the optimizer trades off between the time taken and the best query plan; it will select a less efficient plan if it thinks it will take more time to evaluate many plans than to run a less efficient plan.

The optimizer can generate a trivial plan or non-trivial plan based on the complexity of submitted query.

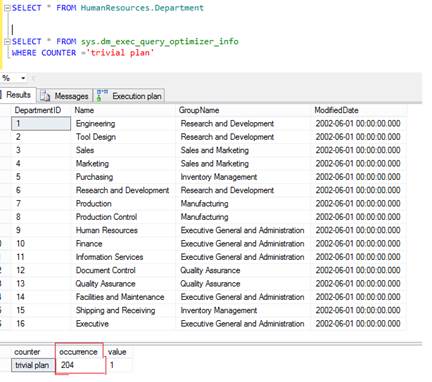
If the submitted query is simple, like SELECT \* FROM <<table\_name>>, the optimizer then will not invest time in optimizing the plan and the actual cost based optimization need not run in such cases. Here the optimizer knows that these queries can only produce a single plan, known as a trivial plan.

You can query the sys.dm\_exec\_query\_optimizer\_info dynamic management view passing COUNTER ='trivial plan' in the WHERE clause and monitor the value of Occurrence column. It increases after each run of a query that belongs to the trivial plan.

Before executing simple SQL query:

  
Before SQL Query

After running simple SQL query:

  
After SQL Query

If you make a small change in the above query, SELECT \* FROM <<table\_name>>, and introduce one more table with a join, it will make the plan non-trivial; then optimizer will perform a cost based calculation to select a plan. Optimizer always refers [statistics](http://technet.microsoft.com/en-us/library/cc966419.aspx) maintained by SQL Server to achieve a cost based calculation to generate the query plan.

Optimizer considers available [statistics](http://technet.microsoft.com/en-us/library/cc966419.aspx) with the query processor tree (the tree generated by the binding process) and heuristically decides on the best plan. This is decided among a number of generated plans trying with different possible joins, reconsidering indexes and applying different join orders until the optimizer evaluates the fastest plan. During the whole processes, the optimizer assigns a number to each step as optimizer’s estimation of the time it thinks that step will take. This is called the estimated cost for that step and the addition of the costs for each step is the cost for the execution plan.

The optimizer selects best plan, which it considers as the actual plan, and stores it in a memory space known as plan caches unless an identical plan already exists. Every time a newly generated plan is compared with plans already available in the plan cache and if there is a match, the previous plan is used.

Now, you know the process of how SQL Server engine generates a best (less expensive) query execution plan using SQL Server relational engine.

Once the query execution plan is ready and available to execute, SQL Server storage engines gets the query plan and executes it based on the actual query plan. After executing the query, the desired output is returned to you.

There are some more facts of which you should aware:

* Each query has two plans; estimated execution plan and actual execution plan.
* In some situations, Estimated and Actual execution plans may differ.
* SQL Server reuses query plans whenever possible.

In this article I’m not covering the details of these facts but if you are curious to learn about them, then visit [MSDN](http://msdn.microsoft.com/en-us/library/ms191227%28v=sql.105%29.aspx).

## Query Execution Plan Type

SQL Server provides a variety of query execution plan formats to fulfill different needs for developers and DBAs. These formats provide a different level of detailed information to understand the query execution.

* Graphical plans – Most preferred format to understand the query execution and troubleshoot the query performance issue.
* Text plans – A little tough to understand but detailed information is available, consisting of three different text plan formats (SHOWPLAN\_ALL, SHOWPLAN\_TEXT and STATISTICS PROFILE).
* XML plans – Most complete set of information available in the structured XML format, consisting two different XML plan formats (SHOWPLAN\_XML and STATISTICS\_XML).

## Summary

In today's article, we have explored how SQL Server generates the query execution plan and learned the processes involved to achieve the task. SQL Server has different components, which work internally and provide you the desired result set in the best possible time. Physical query processing consists of two basic steps, query compilation and execution. The main connection between the two steps is the query execution plan. Understanding how SQL Server generates the query plans and how it uses the query plan to return the query result is very helpful. Query execution plan enables database developers, DBAs and architects to understand the processing of a query on SQL Server and helps to troubleshoot any performance issue that occurs during its execution.

Now we understand fundamentals of the SQL Server query execution plan and how it generates. In the next article we’ll explore how we can interpret an execution plan and its operators.

## References

[http://msdn.microsoft.com/en-us/library/ms191227(v=sql.105).aspx](http://msdn.microsoft.com/en-us/library/ms191227%28v=sql.105%29.aspx)

<http://en.wikipedia.org/wiki/Query_optimization>

<http://my.safaribooksonline.com/book/databases/sql/9781906434939/chapter-1-execution-plan-basics/chap01_sub1_xhtml>