The SQL Server Query Optimizer is a cost-based optimizer. Each possible execution plan has an associated cost in terms of the amount of computing resources used. The Query Optimizer must analyze the possible plans and choose the one with the lowest estimated cost. Some complex SELECT statements have thousands of possible execution plans. In these cases, the Query Optimizer does not analyze all possible combinations. Instead, it uses complex algorithms to find an execution plan that has a cost reasonably close to the minimum possible cost.

The SQL Server Query Optimizer does not choose only the execution plan with the lowest resource cost; it chooses the plan that returns results to the user with a reasonable cost in resources and that returns the results the fastest. For example, processing a query in parallel typically uses more resources than processing it serially, but completes the query faster. The SQL Server Query Optimizer will use a parallel execution plan to return results if the load on the server will not be adversely affected.

The SQL Server Query Optimizer relies on distribution statistics when it estimates the resource costs of different methods for extracting information from a table or index.

The SQL Server Query Optimizer is important because it enables the database server to adjust dynamically to changing conditions in the database without requiring input from a programmer or database administrator. This enables programmers to focus on describing the final result of the query. They can trust that the SQL Server Query Optimizer will build an efficient execution plan for the state of the database every time the statement is run.

**Trivial plan optimization**

If SQL Server knows that there is only one viable plan for a query, a lot of work can be avoided. For example, where there is only one possible plan:

A query that consists of an INSERT with a VALUES clause.

A SELECT statement for which all the columns are among the keys of a unique composite index, and that index is the only one that is relevant. No other index has that set of columns in it.

The trivial plan query optimizer finds the really obvious plans that are typically very inexpensive. This saves the query optimizer from having to consider every possible plan, which can be costly and can outweigh any benefit provided by well-optimized queries.

**Simplification**

If there is no trivial plan, SQL Server will perform a simplification. These are usually syntactic transformations of the query itself, to look for commutative properties and operations that can be rearranged. There is no cost or available index analysis. SQL Server then loads up the metadata, including the statistical information on the indexes.

**Cost-based optimization**

A set of transformation rules is followed that apply various permutations of data access strategies, join orders, aggregation placement, subquery transformations, and other rules that guarantee that a correct result is returned.

Optimization is broken up into three search phases to prevent the process from taking much longer than the execution of the query.

SQL Server evaluates the cost of the cheapest plan after each phase and executes that plan, if the plan is cheap enough. Otherwise, the query optimizer runs the next phase, which involves another set of usually more complex rules.

* + **Phase 0 – Transactional Processing Phase**: Contains a very limited set of rules and is applied only to simple queries with three or fewer tables. A limited number of join orders are evaluated, which may generate many potential plan candidates. Hash and nested loop join strategies are considered. If a plan with an estimated cost below 0.2 is found, the optimization ends and the query is executed. This phase is also known as the Transaction Processing phase, because query plans produced by this phase are typically found for queries in online transaction processing (OLTP) applications.
  + **Phase 1 – Quick Plan Optimization Phase**: More transformation rules and different join orders are evaluated. If the best plan costs less than 1.0, the optimization ends.

The query optimizer has considered only non-parallel execution plans so far. Phase 1 is repeated with the goal of finding the best parallel plan if:

* + - * + More than one logical CPU is available to SQL Server.
        + The least expensive plan produced by Phase 1 costs more than the Cost Threshold for Parallelism.
        + The cost is >= 1.
  + **Phase 2 – Full Optimization Phase**: The cost of the serial and parallel plans are compared and Phase 2 is executed for the cheaper plan. Outer join reordering and automatic indexed view substitution for multi-table views are also considered.

An internal timeout can occur and the Query Optimizer stops the optimization process and returns the least expensive plan it has found so far. This timeout event is also shown on the properties of a graphical plan as **Reason For Early Termination of Statement Optimization** or on an XML plan as **StatementOptmEarlyAbortReason**.

We should be aware that if the full optimization phase produced a parallel plan for this query, it does not necessarily mean that the plan will be executed on multiple processors. If the machine is very busy, and cannot support running a single query on multiple CPUs, the plan