**Activity-5**

**Study and present the working of SQL optimizer**

# The SQL Server Query Optimizer

To understand how to write SQL code for SQL Server that performs well, it is important to appreciate how the query optimizer works. Ben Nevarez explains the essentials, in a broad sweep through a complex subject, in an article taken from his new book 'Inside the SQL Server Query Optimizer'.

The SQL Server Query Optimizer is a **cost-based optimizer**. It analyzes a number of candidate execution plans for a given query, estimates the cost of each of these plans and selects the plan with the lowest cost of the choices considered. Indeed, given that the Query Optimizer cannot consider every possible plan for every query, it actually has to do a cost-based balancing act, considering both the cost of finding potential plans and the costs of plans themselves.

Therefore, it is the SQL Server component that has the biggest impact on the performance of your databases. After all, selecting the right (or wrong) execution plan could mean the difference between a query execution time of milliseconds, and one of minutes or even hours. Naturally, a better understanding of how the Query Optimizer works can help both database administrators and developers to write better queries and to provide the Query Optimizer with the information it needs to produce efficient execution plans.

# How the Query Optimizer Works

At the core of the SQL Server Database Engine are two major components:

the **Storage Engine** and the **Query Processor**, also called the Relational Engine. The Storage Engine is responsible for reading data between the disk and memory in a manner that optimizes concurrency while maintaining data integrity. The Query Processor, as the name suggests, accepts all queries submitted to SQL Server, devises a plan for their optimal execution, and then executes the plan and delivers the required results.

Queries are submitted to SQL Server using the SQL language (or T-SQL, the Microsoft SQL Server extension to SQL). Since SQL is a high-level declarative language, it only defines what data to get from the database, not the steps required to retrieve that data, or any of the algorithms for processing the request. Thus, for each query it receives, the first job of the query processor is to devise a plan, as quickly as possible, which describes the best possible way to execute said query (or, at the very least, an efficient way). Its second job is to execute the query according to that plan.

Each of these tasks is delegated to a separate component within the query processor; the **Query Optimizer** devises the plan and then passes it along to the **Execution Engine**, which will actually execute the plan and get the results from the database.

## SQL Statement

* **Parsing**
* **Binding**
* **Query optimization**
* **Query execution**
* **Query results**

**Parsing and binding –**the query is parsed and bound. Assuming the query is valid, the output of this phase is a logical tree, with each node in the tree representing a logical operation that the query must perform, such as reading a particular table, or performing an inner join. This logical tree is then used to run the query optimization process, which roughly consists of the following two steps;

**Generate possible execution plans –** using the logical tree, the Query Optimizer devises a number of possible ways to execute the query i.e. a number of possible execution plans. An execution plan is, in essence, a set of physical operations (an index seek, a nested loop join, and so on), that can be performed to produce the required result, as described by the logical tree;

* **Cost-assessment of each plan –**While the Query Optimizer does not generate every possible execution plan, it assesses the resource and time cost of each plan it does generate. The plan that the Query Optimizer deems to have the lowest cost of those it’s assessed is selected, and passed along to the Execution Engine;

**Query execution, plan caching –** the query is executed by the Execution Engine, according to the selected plan. The plan may be stored in memory, in the plan cache.

* Parsing and binding are the first operations performed when a query is submitted to a SQL Server instance. Parsing makes sure that the T-SQL query has a valid syntax, and translates the SQL query into an initial tree representation: specifically, a tree of logical operators representing the highlevel steps required to execute the query in question. Initially, these logical operators will be closely related to the original syntax of the query, and will include such logical operations as “get data from the Customer table”, “get data from the Contact table”, “perform an inner join”, and so on. Different tree representations of the query will be used throughout the optimization process, qand this logical tree will received
* different names until it is finally used to initialize the Memo structure, as will be discussed later.
* Binding is mostly concerned with name resolution. During the binding operation, SQL Server makes sure that all the object names do exist, and associates every table and column name on the parse tree with their corresponding object in the system catalog. The output of this second process is called an algebrized tree, which is then sent to the Query Optimizer.

The next step is the optimization process, which is basically the generation of candidate execution plans and the selection of the best of these plans according to their cost. As has already been mentioned, SQL Server uses a cost-based optimizer, and uses a cost estimation model to estimate the cost of each of the candidate plans.

In essence, query optimization is the process of mapping the logical query operations expressed in the tree representation to physical operations, which can be carried out by the execution engine. So it’s actually the functionality of the execution engine that is being implemented in the execution plans being created by the Query Optimizer, that is, the execution engine implements a certain number of different algorithms and it is from these algorithms that the Query Optimizer must choose, when formulating its execution plans. It does this by translating the original logical operations into the physical operations that the execution engine is capable of performing, and execution plans show both the logical and physical operations. Some logical operations, such as a Sort, translate to the same physical operation, whereas other logical operations map to several possible physical operations. For example, a logical join can be mapped to a Nested Loops Join, Merge Join, or Hash Join physical operator.

* Thus, the end product of the query optimization process is an execution plan:

a tree consisting of a number of physical operators, which contain the algorithms to be performed by the execution engine in order to obtain the desired results from the database.

**2 Learn and report optimization techniques**

## 1 Define the requirements

Frame the optimal requirements before starting to write the query. This will help refine the query to avoid fetching unwanted data from the table.

## 2 SELECT fields, rather than using SELECT \*

Use the SELECT statement optimally, instead of always fetching all data from the table. Fetch only the necessary data from the table, thereby avoiding the costs of transferring unwanted data and processing it.

This query is much simpler, and only pulls the required details from the table.

## 3 Avoid DISTINCT in SELECT query

SELECT **DISTINCT** is a simple way of removing duplicates from a database.

SELECT **DISTINCT** works to generate distinct outcomes by using the **GROUP BY** clause, which groups all the fields in the query. However, a large amount of processing power is required to do this. So, avoid DISTINCT in SELECT queries.

Unduplicated records are returned without using SELECT DISTINCT by adding more fields.

## 4 Indexing

Indexing in SQL Server helps retrieve data more quickly from a table, thereby giving a tremendous boost to SQL query performance. Allow effective use of clustered and nonclustered indexes. Understand the query’s intent and choose the right form for your scenario.

Use a covering index to reduce the time needed for the execution of commonly used statements. Indexes occupy disk space. The more indexes you have, the greater the space used on the disk. In SQL Server, a clustered index requires no additional disk space, but any non-clustered index needs additional disk space as it is stored separately from the list

**5 To check the existence of records, use EXISTS() rather than COUNT()** Both **EXISTS()** and **COUNT()** methods can be used to check the existence of a record entry in the table. The EXISTS() method is more effective as it exits processing as soon as it finds the first entry of the record in the table. The COUNT() method would scan the entire table to return the number of records in the table that match the provided constraint.

## 6 Limit your working data set size

The less data retrieved, the faster the query will run. Instead of adding too many client-side filters, filter the data at the server as much as possible. This limits the data sent on the wire, and you will be able to see the results much more quickly.

## 7 Use WHERE instead of HAVING

The **HAVING** clause filters the rows after all the rows are selected. It works just like a filter.

Do not apply the **HAVING** clause for any other purpose.

**HAVING** statements are determined in the SQL operating order after **WHERE** statements. Therefore, it is quicker to execute the **WHERE** query.

## 8 Ignore linked subqueries

A linked subquery depends on the query from the parent or from an external source. It runs row by row, so the average cycle speed is greatly affected.

## 9 Use of temp table

This is yet another issue that is very difficult to solve. In many cases, we use a temp table to stop double-dipping into large tables. A temp table can also be used to significantly reduce the mandatory computing power when dealing with large volumes of data.

When linking data from a table to a large table, add a large subset to reduce the efficiency hindrance.L

## 10 Don’t run queries in a loop

Coding SQL queries in loops slows the entire sequence. Instead of writing a question and running it in a loop, bulk insert and update can be used depending on the situation.