Optimization SQL Server 2005 Query Using Cost Model and Statistic

Ibnu Gunawan
Informatic Engineering Department – Petra Christian University
Siwalankerto 121 – 131 Surabaya 60236 Indonesia
Telp. (031) – 2983456, Fax (031) – 8417658
ibnu@peter.petra.ac.id

ABSTRACT
MS SQL Server Query Optimizer[1] is an optimization tools that based on a cost model, the database metadata, database statistics, system resources (memory, IO, CPU) and the query itself . If we want to optimize the Query in MS SQL Server, we must optimize the factor that MS SQL Server Query Optimizer rely on. Three of that factor is cost model, statistics and the query itself. In SQL Server 2005[2] one tools that can do optimization based on statistic and query is known as DTA . These Paper will explain how the query optimizer work, then it will explain how the DTA work side by side with query optimizer and it will explain how to use the DTA to auto optimize the query.

Keywords :
MS SQL Server, Database, Statistic, Query, Optimization**.**

1. INTRODUCTION
The query optimizer[3] in MS SQL Server 2005 selects a plan for a given query based on cost model, the database metadata, database statistics, system resources, and the query itself.

The query optimizer is the component in a database system that transform a parsed representation of an MS SQL Server 2005 query into efficient execution plan for evaluating it. Optimizer usually examine a large number of possible query plans and choose the best one in a cost-based manner.

To efficiently choose among alternative query execution plans[4], query optimizers estimate the cost of each evaluation strategy. This cost estimation needs to be accurate (since the quality of the optimizer is correlated to the quality of its cost estimations), and efficient (since it is invoked repeatedly during query optimization).

Poor performance can result due to modeling assumptions[1], outdated statistics, system resource assumptions etc. It is useful to have a tool for database engine users (DBA, application writers, and architects) and designers (developers, customer support) to understand the source of plan inefficiencies.

For example, engine designers can use it to understand why a particular join strategy resulted in poor performance whereas another known join order has better performance. Customer support can use such a tool to help narrow down the possible causes of poor performance to particular characteristics of a plan e.g. wrong join order.

In next section we describe the components of a generic query optimizer and show how statistical information can be used to improve the accuracy of cost estimations, which in turn impacts the whole optimization process.

2. BACKGROUND
The query optimizer is the component in a database system that transforms a parsed representation of an SQL query into an efficient execution plan for evaluating it. Optimizers usually examine a large number of possible query plans and choose the best one in a cost-based manner. To efficiently choose among alternative query execution plans, query optimizers *estimate* the cost of each evaluation Strategy . This cost estimation needs to be accurate (since the quality of the optimizer is correlated to the quality of its cost estimations), and efficient (since it is invoked repeatedly during query optimization). In this section we describe the components of a generic query optimizer and show how statistical information can be used to improve the accuracy of cost estimations, which in turn impacts the whole optimization process. After that we show that optimizer is part of DTA

2.1 Query Optimizer Architecture
There are several optimization frameworks in the literature [5, 6, 7, 8, 9] and most modern optimizers rely on the concepts introduced by those references. Although the implementation details
vary among different systems[4], all optimizers share the same basic structure [10], shown in Figure 1.

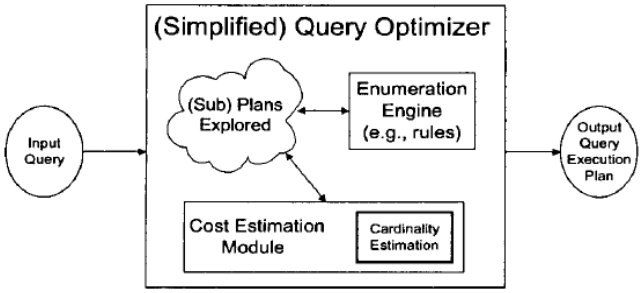

Figure 1. Simplified Optimizer's Architecture.

For each incoming query, the optimizer maintains a set of sub-plans already explored, taken from an implicit search space. An enumeration engine navigates through the search space by applying rules to the set of explored plans. Some optimizers have a fixed set of rules to enumerate all interesting plans (e.g., System-R) while others implement extensible transformational rules to navigate through the search space (e.g., Starburst, Cascades). All systems use dynamic programming or memorization to avoid recomputing the same information during query optimization. For each discovered query plan, a component derives different properties if possible, or estimates them otherwise. Some properties (e.g., cardinality and schema information) are shared among all plans in the same equivalence class, while others (e.g., estimated execution cost and output order) are tied to a specific physical plan. Finally, once the optimizer has explored all interesting plans, it extracts the most efficient plan, which serves as the input for the execution engine.
A useful property of a query plan from an optimization perspective is the estimated execution cost, which ultimately decides which is the most efficient plan. The estimated execution cost of a plan, in turn, depends heavily on the cardinality estimates of its sub-plans. Therefore, it is fundamental for a query optimizer to rely on accurate and efficient cardinality estimation algorithms.

EXAMPLE 1. Consider the query in Figure 2(a) and suppose that IRI ~ ISl ~ ITI. If the query optimizer has knowledge that R.a < 10 is much more selective than T.b > 20 (i.e.,just a few tuples in R verofy R.a < 10 and most of the tuples in T verify T.b > 20), it should determine that plan P1 in Figure 2(b) is more eficient than theplan P2 in Figure 2(c) 3. The reason is that Pl first joins R and S producing a (hopefully) small intermediate result that is in turn joined with T. In contrast, P2 produces a large intermediate result by first joining S and T.

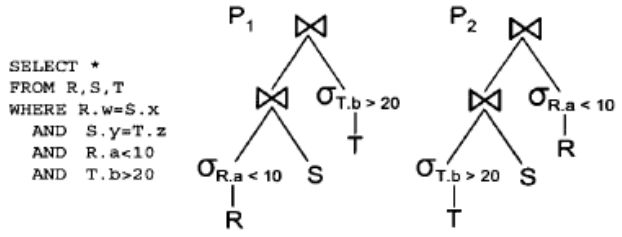

Figure 2. Query plans chosen by query optimizers depending on the cardinality of intermediate results.

Cardinality estimation uses statistical information about the data that is stored in the database system to provide estimates to the query optimizer. Histograms are the most common statistical information used in commercial database systems.

2.2 Database Tuning Advisor Architecture

DTA takes as input a workload consisting of T-SQL (SQL Server's flavor of the SQL language) statements such SELECT, INSERT, UPDATE, DELETE, stored procedure calls, dynamic SQL, and DDL statements and produces as output a T-SQL script that consists of recommendations for indexes, materialized views (called indexed views in Microsoft SQL Server), and horizontal partitioning[2].

The architecture of DTA appears in Figure 3. In Figure 3 we see that the query optimizer is a part of DTA. The details are presented in [11], and have been omitted. For clarity, we will summarize the input/output of DTA.

DTA recommends a set of indexes, materialized views, indexes on materialized views and their respective horizontal partitioning that is appropriate for the given workload. The tool relies on interfaces provided by Microsoft SQL Server's query optimizer [12, 13]. First it needs to be able to simulate partitioned tables/indexes and materialized views (referred to as hypothetical indexes and materialized views) that do not exist in the current database. Second, it needs to tell the query optimizer to optimize a query for a given hypothetical configuration (a set of partitioned tables, indexes, materialized views and indexes on materialized views).

DTA [2] searches the space of (partitioned) tables, indexes and materialized views to arrive at the best configuration for the given workload. In general, the above search problem can be prohibitively expensive. Column Group Restriction, Candidate Selection, Merging, and Enumeration (Figure 2) are individual steps in the search algorithm that allow DTA to search the space effectively [14].

3. DISCUSSION AND ANALYSIS

In these section we describe how to use DTA to optimize the statistic for SQL Server 2005 database, then we show the effect on that database. And it will ended by show the analysis of these process.

3.1 Optimization of SQL Server 2005 Query

For this experiment, we used the database and query loading from Microsoft SQL Server 2005 official training course number 2784A which titled Tuning and Optimizing Queries Using Microsoft SQL Server 2005.

The scenario is based on Baldwin Museum case study on chapter 3. Baldwin Museum of Science is a hands-on science center located in a medium-sized city in the eastern United States. Its mission is to educate the public and enrich local schools with knowledge of science and nature.

The museum recently upgraded its database to SQL Server 2005, and the database has now become an integral part of the day-to-day operation of the museum. Although the past two years have seen zgood museum growth, users have started to complain about the performance of this application.

The database named Baldwin2, will be tested with script. When the script is running without optimization from DTA, it consume about 3.35 min in execution time, we can see the detail of it in figure 4. And then we start the DTA, the user interface is in like figure 5. at the first DTA running, it will asked about the database and the load file. After that, we just click start analysis button like figure 6.

Then The DTA will show the result of its analysis based on SQL workload and database, like figure 7. Beside that, the DTA also give a recommendation based on SQL workload and database. These recommendation can be see in figure 8. in Figure 8 we can see that DTA is not only given recommendation for indexing the SQL Server database based on sql workload, but it can give

recommendation for creating statistic too. There is 5 statistic that will be created for us if we apply these recommendation. Start from _dta_stat_197575742_7_8_5_6 until _dta_stat_2089058478_3.

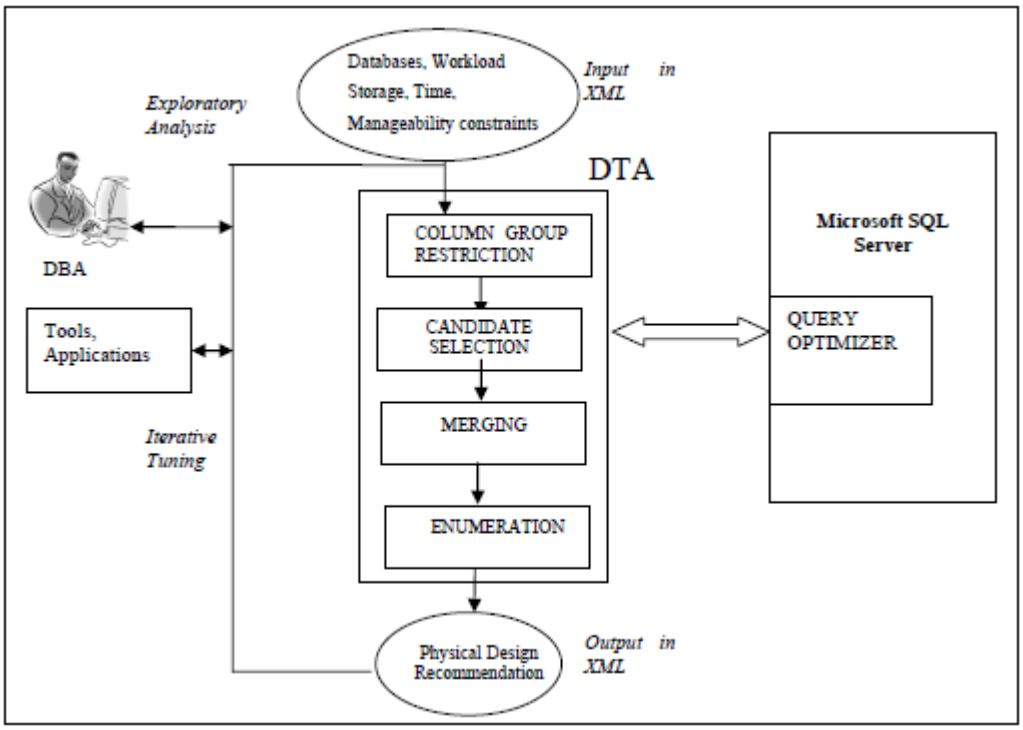

Figure 3. DTA Architecture

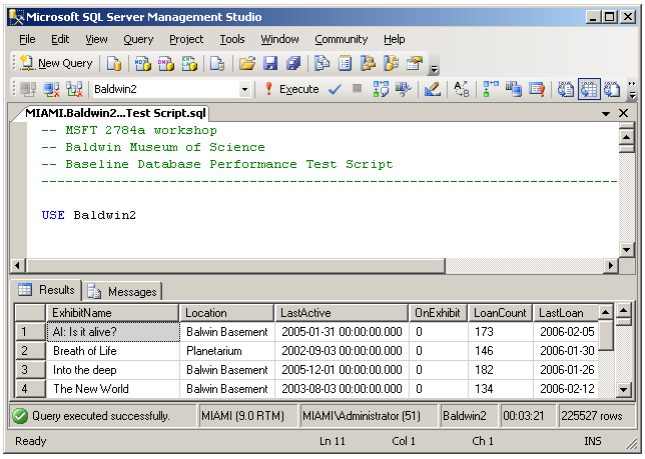

Figure 4. Load stress result without DTA

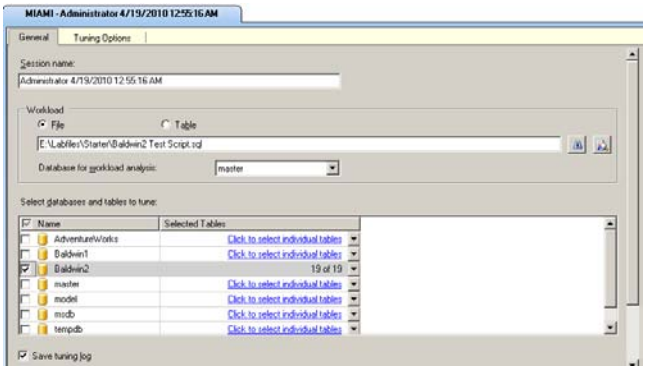

Figure 5. Load stress test file to test the database

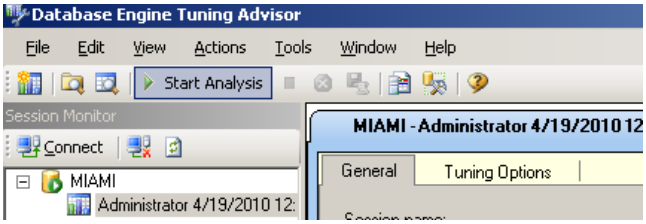

Figure 6. begin analysis

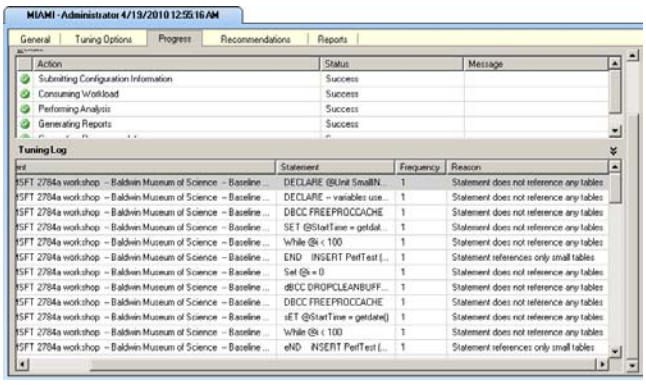

Figure 7. analysis result

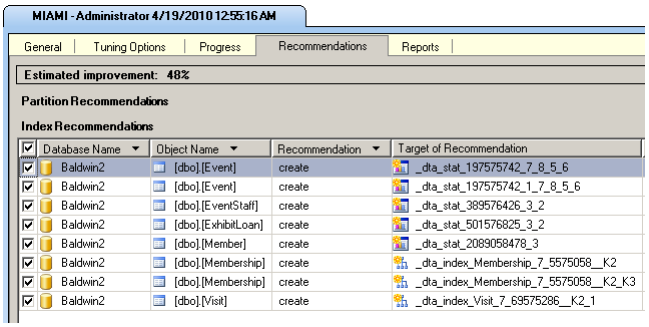

Figure 8. recommendation

After we applying the recommendation, figure 9 will showed, how many recommendation have been success in implementation.

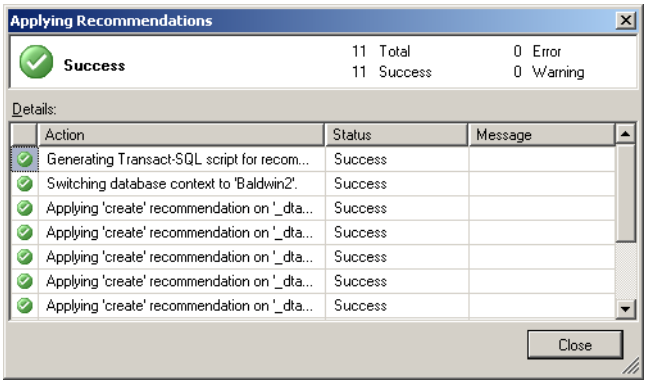

Figure 9. applying recommendation

If we success in applying recommendation, then the performance of baldwin 2 should be more better than before. Because of that, let try it by running the sql load script once more

It is shown in figure 10. it shown that the execution time is down to 1 minute 52 second, is far more efficient than the first time query run without optimization from DTA, which taken time 3 minutes 21 seconds. Based on these case study we can make a conclucion that using DTA the execution time can be sliced up to 50%

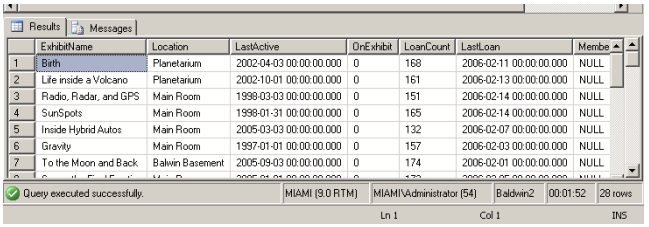

Figure 10. result from DTA

4. CONCLUSION

The purpose of this research is to prove that statistic has important role to optimize the SQL Server Database via Query Optimizer. Then we see that query optimizer is a part of DTA. So because DTA is a part of SQL Server 2005 Database tools , then we can make a conclusion that DTA is based on statistic too. It proven by DTA recommendation. It is no give

recommendation just based on index, but it can give a recommendation based on statistic too..

5. REFERENCES

[1] Zhang.XS and Gosalia. A. 2008. Automatic Plan Choice Validation Using Performance Statistics. DB Test'08. (June , 2008)

[2] Chaudhuri. S, Agrawal.S, Kollar.L, Marathe.A, Narasayya.V, Syamala M. 2005. Database Tuning Advisor for Microsoft SQL Server 2005: Demo. ACM SIGMOD. (June. 2005), 930-932.

[3] L. Giakoumakis and C. Galindo-Legaria. Testing SQL Server's Query Optimizer: Challenges, Techniques and Experiences. *IEEE Bulletin of the Technical Comitee on Data Engineering*, 31, 1(March 2008), 37-44

[4] Chaudhuri. S, Bruno. N. 2002. Exploiting Statistics on Query Expression for Optimization. ACM SIGMOD. (June. 2002), 263-274.

[5] G. Graefe. The cascades framework for query optimization. *Data Engineering Bulletin*, 18(3), 1995.

[6] G. Graefe and D.J.DeWitt. The exodus optimizer generator. In *Proceedings of the 1987 ACM International Conference on Management of Data (SIGMOD'87)*, 1987.

[7] G. Graefe and W.J.McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth International Conference on Data Engineering*, 1993.

[8] L.M. Haas, J. C. Freytag, G.M. Lohman, and H. Pirahesh. Extensible query processing in starburst. In *Proceedings of the 2000 ACM International Conference on Management of Data (SIGMOD'89)*, 1989.

[9] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, and management system. In *Proceedings of the 1979 ACM International Conference on Management of Data (SIGMOD'79)*, 1979.

[10] S. Chauduri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 1998.

[11] Agrawal, S., Chaudhuri, S., Kollar L., Marathe A., Narasayya V. and Syamala, M. Database Tuning Advisor For Microsoft SQL Server 2005. In *Proceedings of the VLDB 2004 Conference*, Toronto, 2004.

[12] Agrawal, S., Chaudhuri, S. and Narasayya V. Automated Selection of Materialized Views and Indexes for SQL Databases. In *Proceedings of the 26th VLDB Conference*, Cairo, 2000.

[13] Chaudhuri, S. and Narasayya V. An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the 23rd VLDB Conference*, Athens, 1997.

[14] Agrawal, S., Narasayya V. and Yang, B. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database. In *Proceedings of the ACM SIGMOD*, Paris, 2004