EPJ Web of Conferences **68**, 00033 (2014)
DOI: 10.1051/epjconf/20146800033

Evaluation of Sub Query Performance in SQL Server

Tanty Oktavia[1], Surya Sujarwo[2]
toktavia@binus.edu, surya.ss@binus.edu

[1]School of Information System, Bina Nusantara University, Jl. K.H. Syahdan No. 9 Palmerah Jakarta 11480 Indonesia
[2]School of Computer Science, Bina Nusantara University, Jl. Kebon Jeruk Raya No. 27, Jakarta Barat 11530 Indonesia

Abstract. The paper explores several sub query methods used in a query and their impact on the query performance. The study uses experimental approach to evaluate the performance of each sub query methods combined with indexing strategy. The sub query methods consist of in, exists, relational operator and relational operator combined with top operator. The experimental shows that using relational operator combined with indexing strategy in sub query has greater performance compared with using same method without indexing strategy and also other methods. In summary, for application that emphasized on the performance of retrieving data from database, it better to use relational operator combined with indexing strategy. This study is done on Microsoft SQL Server 2012. **Keywords:** Sub query, indexing, performance, SQL Server

1 Introduction

Business application relies on a database management system (DBMS) for retrieving and storing data, which its overall performance depend on. For this matter, it must be administered and optimized for better performance according to business application needs, as well as quick in handling all kind of performance threats. Its performance influenced by several factors i.e.: growing database size proportional with the data, increasing user base, increasing user processes, improperly and un-tuned system [1]. These factors must be maintained according to business application needs to stabilizing the overall performance of each request to DBMS.

One of the major critical issues that often happened in a company is inadequate performance of queries used for the suitable output. Many factors causing this issue, one of them is query processing problem. Since then, a significant amount of research and observation has been done to find an efficient solution for processing queries. A query may be expensive in cost of execution if it is not optimized well [2]. This will give a negative impact on the performance of business application, hence reducing business performance. The degradation of performance can be detected by performing monitoring at a timely basis on system performance parameter.

In the first generation database management systems, the low level procedural query language is embedded in a high level programming language and the programmer's should select the most appropriate execution strategy. In contrast, with declarative languages such as Structured Query Language (SQL), the user specifies what data is required rather than how it is to retrieved [3]. This pattern transforms the user responsibility from determine method to support good execution strategy. The most important objectives to be considered in order to improve the performance of DBMS are: designing an efficient data schema, optimizing indexes, analyzing execution plans, monitoring access to data, and optimizing query [4].

The object of this study is to present the comparative performance of sub query methods using Microsoft SQL server 2012 by using large data according to experimental method.

2 Methodologies

This study follows experimental method i.e. generate model, simulate the model, record and then analyze the record, we use this method because the performance measurements is a significant issue [5].

It is based on operational data of student attendances in a laboratory that consists of 6 columns and 500,000 records. The table consists of record id, class id, student id; attend date time and place, status, and record date that save all students attendances with attendance updates as a new record. The queries for experiment must produce the latest status of all students from the attendances table. The experiment uses several query variation by using sub query method: in, exists, relation operator and relation operator with top operator, and the table used in experiment use two indexing scenario [6]. In the first scenario, the table uses indexing in record id column, the second one uses additional indexing in class id, student id include record date, and also in class id, student id, include status and record date. For the analysis, each query and indexing scenario is executed 10 times on Microsoft SQL Server 2012 and each execution's server processing time is recorded using client statistics feature provided by SQL Server Management Studio.

3 Query Execution

Business application retrieves and stores data from DBMS using query. The query operations used by the application consist of select, insert, update, and delete. Each operation is run by DBMS then the result is transfer to business application. All data in DBMS are stored in file on physical device such as disk device. Assume one file consists of many records and user want to retrieve a single records based on a particularly criteria [3], the disk device is able to going directly in the middle of a file to retrieve the record. To accomplish that, the system need time to move the disk device and retrieve the requested record using many procedures compilation in DBMS. To reduce the time needed by the system, the DBMS must be tuned according to the executed query.

The process of DBMS manages query are: after receiving query from external level, the query are checked semantically and syntactically by the query parser. If there are violation of structure, user right, or procedure; an error message will be return, otherwise query will be translated into internal level in relational algebra expression. After that, query optimizer selects appropriate optimal method to implement relational algebra and generate query execution plan then executed [7]. This DBMS's process can be seen at Figure 1.

The query execution plan is a compiled code that contains the ordered steps to carry out the query [8]. Identifying an appropriate plan for execution is very important because this will determine the effectiveness of the execution. By using statistics on tables and indexes, the optimizer predicts the cost of alternative access methods to resolve a particular query.

Queries in algebra are constructed using operators. Each relational query describes a step by step procedure to compute the desired output, based on the order in which operators are applied [9]. There are many variations of the operations included in relational algebra. The fundamental operations in relational algebra are selection, projection, Cartesian product, union, set difference, join, intersection, and division operations. The selection and projection operations are unary operations, which operate only on one relation. The other operations work on pairs of relations is therefore called binary operations [3].

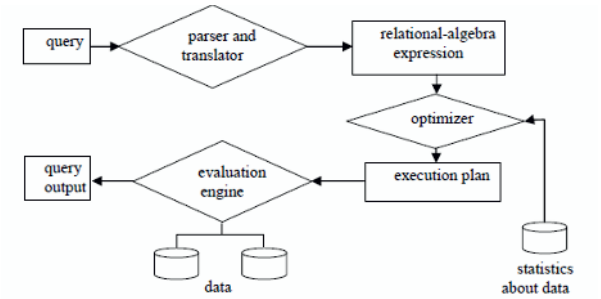

Figure 1 Process of Query Execution [7]

As one of the binary operations, a sub query is a query that nested somewhere inside a DML syntaxes, such as INSERT, UPDATE, DELETE, OR SELECT statement. Sub query can be applied with keyword IN, EXISTS and relation operator. By default, the sub query result only consists of a single column name or expression, except for sub query that use EXISTS keyword. Sub query can be combined with all function of SQL in DBMS. The DBMS will resolve the IN condition by accessing the index for number of times equal to the number of values which to search. There are three types of sub query [3]:

- A scalar sub query returns a single column and a single row.
- A row sub query returns multiple columns, but only a single row.
- A table sub query returns one or more column and multiple rows.

The types of sub query can be applied in accordance what data and how many value of data, a user wants to return.

According to the utilization requirements for the system, SQL Server provides indexing method. It

divided into clustered index and non clustered index. The clustered indexes are recommended only for tables that are frequently updated. Clustered type indexes are effective when operators like BETWEEN, >, >=, <, <=, <>, and !=. Non clustered indexes usage is recommended only for databases where updates are infrequent and gives the optimal solution for the "exact match" [10]. Many people wonders which attributes are suitable for indexing to be applied for getting better performance. Gilenson [6] states there are two sorts of possibilities: primary keys, and search attributes.

Indexes are powerful method for improving searching time, but we should keep in mind that when the record in an indexed table is modified, the system must take the time to update the table's indexes too. If user updates a lot of data, the time that it takes to execute the updates operation and update all the indexes could slow down the operations that are just trying to read the data for applications, and also degrading query response time. Hence that fact, user should beware when applied index in query. The placement of index must be precise with the necessity and procedures.

In transact SQL statements, there is usually no procedure that regulate when to apply a sub query or that does not, because it is not difference between them. User does not concern how to retrieve the data, but how many times the execution occurred. However, in some cases where the data to be returned numerous or the conditions from the query are very complicated, it caused the performance query is going to down. In case of query optimization, it is impractical to search evaluation plans exhaustively, when the optimization of query involves many relations [11].

4 RESULTS

The table used in the experiment is named *TransactionAttendances*. The *TransactionAttendances* table consists of *TransactionAttendanceId* as record id, *ClassTransactionDetailId* as class id, *BinusianId* as student id, *AttendDate* as attend date time, *AttendPlace* as attend place, *Status* as attend status, and *InsertedDate* as record date. The data type for each column in the table can be seen in table 1. The table is populated with 500,000 operational records. The four queries used in the experiments can be seen in table 2.

Table 1. *TransactionAttendance* table structure

Column Name	Data Type	Length	Key
TransactionAttendanceId	uniqueidentifier		Primary
ClassTransactionDetailId	uniqueidentifier		Foreign
BinusianId	uniqueidentifier		Foreign
AttendDate	datetime		
AttendPlace	nvarchar	25	
Status	varchar	256	
InsertedDate	datetime		

Table 2. List of Query Used For Experiments

No.	Methods	Query
1	IN	SELECT i.ClassTransactionDetailId,i.BinusianId,i.Status FROM Messier.dbo.TransactionAttendances i WHERE CONVERT(VARCHAR(50),i.ClassTransactionDetailId)+CONVERT(VARCHAR(50), i.BinusianId)+CONVERT(VARCHAR(50),i.InsertedDate,13) IN(SELECT CONVERT(VARCHAR(50),ClassTransactionDetailId)+CONVERT(VARCHAR(50), BinusianId)+CONVERT(VARCHAR(50),LastInsertedDate,13) AS [Key] FROM (SELECT ClassTransactionDetailId, BinusianId, MAX(InsertedDate) AS LastInsertedDate FROM TransactionAttendances GROUP BY ClassTransactionDetailId, BinusianId) x) GO
2	EXIST	SELECT i.ClassTransactionDetailId,i.BinusianId,i.Status FROM Messier.dbo.TransactionAttendances i WHERE EXISTS(SELECT 1 FROM (SELECT ClassTransactionDetailId, BinusianId, MAX(InsertedDate) AS LastInsertedDate FROM TransactionAttendances GROUP BY ClassTransactionDetailId, BinusianId) x WHERE CONVERT(VARCHAR(50), i.ClassTransactionDetailId)+CONVERT(VARCHAR(50), i.BinusianId)+CONVERT(VARCHAR(50),i.InsertedDate,13) = CONVERT(VARCHAR(50), ClassTransactionDetailId)+CONVERT(VARCHAR(50), BinusianId)+CONVERT(VARCHAR(50),LastInsertedDate,13)) GO
3	RELATIONAL OPERATOR (=)	SELECT i.ClassTransactionDetailId,i.BinusianId,i.Status FROM Messier.dbo.TransactionAttendances i WHERE CONVERT(VARCHAR(50), i.ClassTransactionDetailId)+CONVERT(VARCHAR(50),i.BinusianId)+CONVERT(VARCHAR(50),i.InsertedDate,13) =(SELECT CONVERT(VARCHAR(50), ClassTransactionDetailId)+CONVERT(VARCHAR(50), BinusianId)+CONVERT(VARCHAR(50),LastInsertedDate,13) AS [Key] FROM (SELECT ClassTransactionDetailId, BinusianId, MAX(InsertedDate) AS LastInsertedDate FROM TransactionAttendances GROUP BY ClassTransactionDetailId, BinusianId) x WHERE x.ClassTransactionDetailId = i.ClassTransactionDetailId AND x.BinusianId = i.BinusianId) GO
4	RELATIONAL OPERATOR (=) + TOP	SELECT i.ClassTransactionDetailId,i.BinusianId,i.Status FROM Messier.dbo.TransactionAttendances i WHERE i.TransactionAttendanceId =(SELECT TOP 1 TransactionAttendanceId FROM TransactionAttendances x WHERE i.ClassTransactionDetailId = x.ClassTransactionDetailId AND i.BinusianId = x.BinusianId ORDER BY x.InsertedDate desc) GO

All the query in table 2 is used to retrieve the last student attendance record that represented by inserted date of each data. The first to third query is using an aggregate function MAX to return the latest record of student attendance having max inserted date (last inserted). For the fourth query, the query change the MAX function with TOP with the

same purpose as the other query. The result of the experiments can be seen in table 3 and table 4.

Table 3 Time comparison of sub query method with no additional index

Method	No Additional Index (ms)										Average (ms)
	1	2	3	4	5	6	7	8	9	10	
IN	3,310	2,925	3,437	2,859	2,922	2,781	3,047	2,812	2,843	3,094	3,003
EXISTS	2,859	2,797	2,859	3,127	2,876	3,221	4,000	4,000	2,797	2,937	3,147
TOP	3,437	3,578	3,609	3,593	3,640	3,422	3,453	3,390	3,578	3,359	3,506
EQUAL	10,359	5,031	5,047	5,562	14,813	5,126	14,187	5,344	5,678	4,875	7,602

The results in table 3 show a significant time different in processing given sub query between using relational operator (=) which is around 7 seconds and the other three methods which are around 3 seconds. IN and EXISTS method does not show a significant difference in the processing time. From this result, we conclude for table that not use additional indexing we better use IN, or EXISTS method compared to use relational operator in this scenario.

Table 4 Time comparison of sub query method with additional index applied

Method	With Additional Index (ms)										Average (ms)
	1	2	3	4	5	6	7	8	9	10	
IN	1,921	1,969	1,876	1,796	1,796	1,828	1,828	1,968	1,750	2,203	1,894
EXISTS	2,093	1,796	1,750	2,000	1,875	1,796	1,828	2,047	1,703	1,875	1,876
TOP	78	31	15	33	15	15	15	15	31	15	26
EQUAL	203	78	78	62	78	79	49	78	62	62	83

To overcome the boundary of existing condition, this study proposed an improved method of query optimization to reduce execution time by applying indexing strategy. The indexing strategy is produced using SQL Server Database Engine Tuning Advisor to analyze the four queries in this study. After the indexing strategy is applied, there are significant improvements of processing time for every query experimented. The average processing time of IN and EXISTS inversely correlated with relational operator. IN and EXISTS need longer processing time around 1.8 seconds, compared with relational operator and TOP only need minimal processing time around 0.02 seconds and 0.08 seconds. From this result we conclude for table that use additional indexing we better use relational sub query combined with TOP to get the best processing time in this scenario.

For other comparison we compare each method from table with index and without index which can be seen at figure 2, figure 3, figure 4 and figure 5. From all the comparison we can see the effect of applying indexing in table can significantly reduce processing time of SELECT statement.

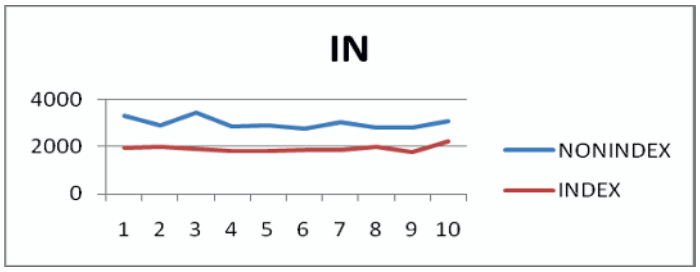

Figure 2 Time Comparison of Query Using IN

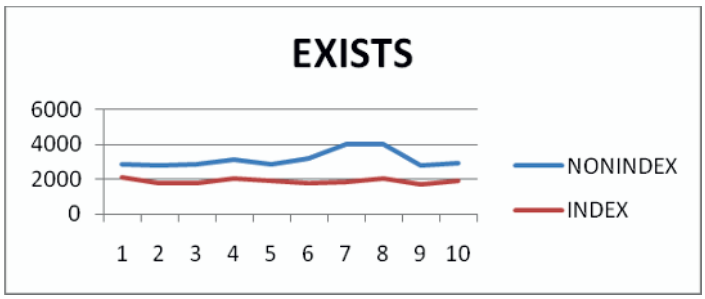

Figure 3 Time Comparison of Query Using EXISTS

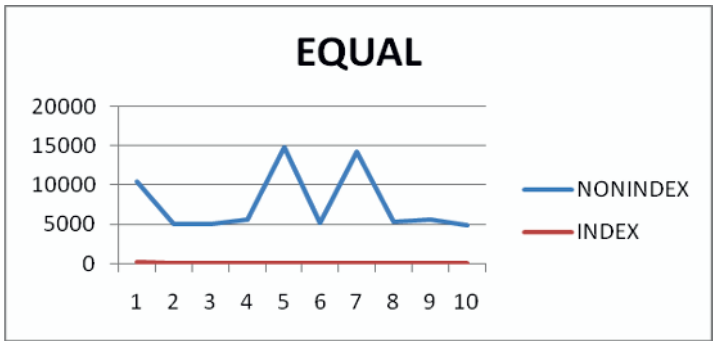

Figure 4 Time Comparison of Query Using Relational Operator

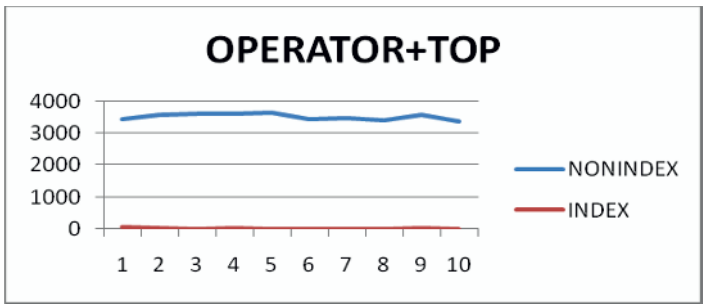

Figure 5 Time Comparison of Query Using Collaboration Operator and TOP

5 CONCLUSION

Optimization of sub query can be done by applying indexing strategy to the table according to condition used in queries. For business application that require fast processing time in retrieving data from database

than processing time in storing date, it's better to applied indexing in all table used by the application, and also all the sub query better use relational operator. Conversely, for business application that required fast processing time in storing data than fast processing time in retrieving data, it's better to not applied indexing and for sub query better use IN or EXISTS, since if indexing applied, the processing time to storing data will increased in each INSERT operation which also automatically update index in the related table.

References

[1] A. Verma, "Enhanced Performance of Database by," *IJCSMS International Journal of Computer Science & Management Studies,* 2011.

[2] M. K. Gupta and P. Chandra, "An Empirical Evaluation of LIKE Operator in Oracle," *BVICAM's International Journal of Information Technology,* 2011.

[3] T. M. Connolly and C. E. Begg, Database Systems : A Practical Approach to Design, Implementation, and Management, Boston: Pearson Education, 2010.

[4] N. Mercioiu and V. Vladucu, "Improving SQL Server Performance," *Informatica Economică,* 2010.

[5] D. Channon and D. Koch X, *Experimental Methodology: Issues and Practice,* 1997.

[6] M. L. Gillenson, Fundamentals of Database Management Systems, USA: Wiley, 2012.

[7] M. Alamery, A. Faraahi, H. H. S. Javadi, S. Nourossana and H. Erfani, "Application of Bees Algorithm in Multi-Join Query Optimization," *ACSIJ Advances in Computer Science: an International Journal,* 2012.

[8] P. Karthik, G. T. Reddy and E. K. Vanan, "Tuning the SQL Query in order to Reduce Time Consumption," *IJCSI International Journal of Computer Science,* 2012.

[9] S. Lasya and S. Tanuku, "A Study of Library Databases by Translating Those SQL Queries Into Relational Algebra and Generating Query Trees," *IJCSI International Journal of Computer Science,* 2011.

[10] I. Lungu, N. Mercioiu and V. Vladucu, "Optimizing Queries in SQL Server 2008," *Scientific Bulletin – Economic Sciences, Vol. 9 (15).*

[11] D. S. M. Mahajan and V. P. Jadhav, "GENERAL FRAMEWORK FOR OPTIMIZATION OF DISTRIBUTED QUERIES," *International Journal of Database Management Systems (IJDMS),* 2012.