Analyze Database Optimization Techniques

Syedur Rahman[1] , A. M. Ahsan Feroz[2], Md. Kamruzzaman[3] and Meherun Nesa Faruque[4]

[1]Jaxara IT Limited, Dhaka, Bangladesh
[2]M&H Informatics, Dhaka, Bangladesh
[3]Jaxara IT Limited, Dhaka, Bangladesh
[4]M&H Informatics, Dhaka, Bangladesh

Summary
Data Management has emerged as the most significant factor in today's world of computing. Applications as diverse as weather satellite feedback to military operation details employ huge databases that store graphics images, texts and other formats of data. The primary challenge in maintaining this information is to access them in an efficient manner. Database optimization techniques have been derived to address this issue that may otherwise limit the performance of a database to an extent of vulnerability. In this paper we therefore discuss the aspects of performance optimization related to data access in transactional databases. Furthermore, we analyze the effect of these optimization techniques.

Key words:
Database, Optimization, Database performance

1. Introduction

Due to modern information technology, which produces ever more powerful computers every year, it is possible today to collect, store, transfer, and combine huge amounts of data at very low costs. Thus an ever increasing number of companies and scientific institutions can afford to build up large archives of documents and other data like numbers, tables, images, and sounds. However, exploiting the information contained in these archives in an intelligent way turns out to be fairly difficult and it becomes harder to work with these data when it starts to enhance. It will be almost impossible to handle or access this large amount of data if we don't do all our database operations in optimized way. When a database based application performs slowly, there is a 90% probability that, the data access routines of that application are not optimized, or, not written in the best possible way. In this paper we will discuss Data access performance optimization in transactional (OLTP) SQL Server databases and will also analyze the performance of a very large database with and without our suggested optimization. Though the optimization techniques are suggested for transactional (OLTP) SQL Server databases but most of the techniques are roughly the same for other database platforms.

Manuscript received August 5, 2010
Manuscript revised August 20, 2010

2. Optimization Techniques

2.1 Indexing in the table column in the database

We need to create primary key in every table of the database. When we create a primary key in a table, a clustered index tree is created and all data pages containing the table rows are physically sorted in the file system according to their primary key values. Each data page contains rows which are also sorted within the data page according to their primary key values. So, each time we ask any row from the table, the database server finds the corresponding data page first using the clustered index tree and then finds the desired row within the data page that contains the primary key value. Following is how an index tree looks like:

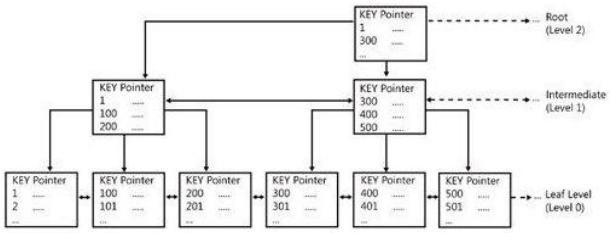

Figure.1: Index tree structure

This is called a B+ Tree (Balanced tree). The intermediate nodes contain range of values and direct the SQL engine where to go while searching for a specific index value in the tree starting from the root node. The leaf nodes are the nodes which contain the actual index values. If this is a clustered index tree, the leaf nodes are the physical data pages. If this is a non-clustered index tree, the leaf nodes contain index values along with clustered index keys (Which the database engine uses to find the corresponding row in the clustered index tree). Usually, finding a desired value in the index tree and jumping to the actual row from there takes an extremely small amount of time for the database engine. So, indexing generally improves the data retrieval operations.

Here we worked on millions of data with some complex query and got the results in seconds.

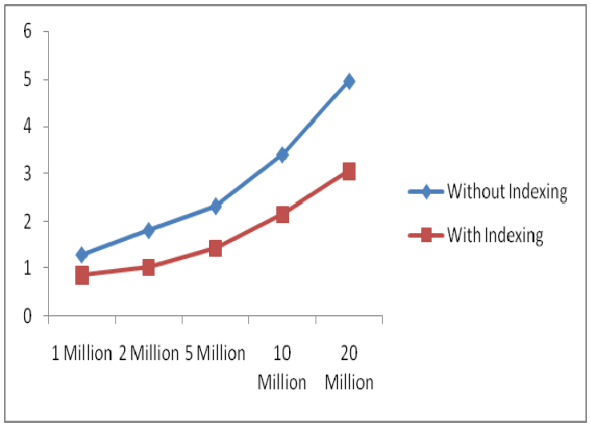

Figure.2: Comparison graph between "With Indexing" and "Without Indexing".

2.2 Appropriate covering indexes

If we know that our application will be performing the same query over and over on the same table, we should consider creating a covering index on the table. A covering index, which is a form of a composite index, includes all of the columns referenced in SELECT, JOIN, and WHERE clauses of a query. Because of this, the index contains the data we are looking for and SQL Server doesn't have to look up the actual data in the table, reducing logical and/or physical I/O, and boosting performance. On the other hand, if the covering index gets too big (has too many columns), this could actually increase I/O and degrade performance. So we have to be very careful to create the covering indexes.
Here we worked on millions of data with some complex query and got the results in seconds.

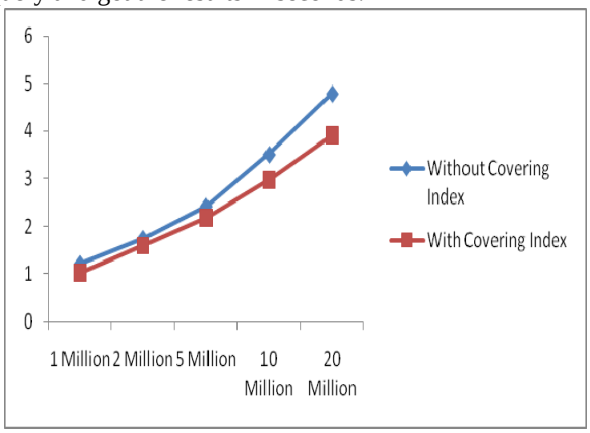

Figure.3: Comparison graph between "With Covering Index" and "Without Covering Index".

2.3 Move TSQL codes from application into the database server

Moving the SQLs from application and implementing these using stored procedures/ Views/ Functions/ Triggers will enable us to eliminate any duplicate SQLs in our application. This will also ensure reusability of our TSQL codes. Implementing all TSQLs using the database objects will enable us to analyze the TSQLs more easily to find possible inefficient codes that are responsible for slow performance. Also, this will let us manage our TSQL codes from a central point. Doing this will also enable us to re-factor our TSQL codes to take advantage of some advanced indexing techniques. Also, this will help us to write more "Set based" SQLs along with eliminating any "Procedural" SQLs that we might have already written in our application. Despite the fact that indexing will let us troubleshoot the performance problems in our application in a quick time, following this step might not give us a real performance boost instantly. But, this will mainly enable us to perform other subsequent optimization steps and apply different other techniques easily to further optimize our data access routines.
Here we worked on millions of data with some complex query and got the results in seconds.

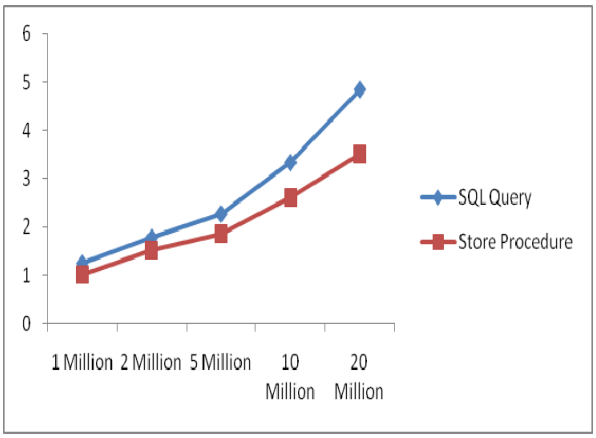

Figure.4: Comparison graph between "SQL Query" and "Store Procedure".

2.4 Organize the file groups and files in the database

When an SQL Server database is created, the database server internally creates a number of files in the file system. Every database related object that gets created later in the database are actually being stored inside these files. An SQL Server database has following three kinds of files:
•.mdf file: This is the primary data file. There could be only one primary data file for each database. All system objects resides in the primary data file and if a secondary

data file is not created, all user objects (User created database objects) also takes place in the primary data file.
•.ndf file: These are the secondary data files, which are optional. These files also contain user created objects.
•.ldf file: These are the Transaction log files. These files could be one or many in number. It contains transaction logs.
Database files are logically grouped for better performance and improvement of administration on large databases. When a new SQL Server database is created, the primary file group is created and the primary data file is included in the primary file group. Also, the primary group is marked as the default group. As a result, every newly created user objects are automatically placed inside the primary file group (More specifically, inside the files in the primary file group). If our database has a tendency to grow larger (Say, over 1000 MB) in size, we can (and should) do a little tweaking in the file/file group organizations in the database to enhance the database performance. Here are some of the best practices we can follow:
•The primary file group must be totally separate and should be left to have only system objects and no user defined object should be created on this primary file group. Also, the primary file group should not be set as the default file group. Separating the system objects from other user objects will increase performance and enhance ability to access tables in the case of serious data failures.
•If there are N physical disk drives available in the system, then we should try to create N files per file group and put each one in a separate disk. This will allow Distributing disk I/O loads over multiple disks and will increase performance.
•For frequently accessed tables containing indexes we should put the tables and the indexes in separate file groups. This would enable to read the index and table data faster.
•We should put the transaction log file on a different physical disk that is not used by the data files. The logging operation (Transaction log writing operation) is more write-intensive, and hence, it is important to have the log on the disk that has good I/O performance.
Here we worked on millions of data with some complex query and got the results in seconds.

2.5 Apply partitioning in the big fat tables

Table partitioning means nothing but splitting a large table into multiple smaller tables so that, queries has to scan less amount data while retrieving. That is “Divide and conquer”. When we have a large (In fact, very large, possibly having more than millions of rows) table in our database we should consider portioning this table to improve performance. Suppose we have a table containing 10 millions of rows.

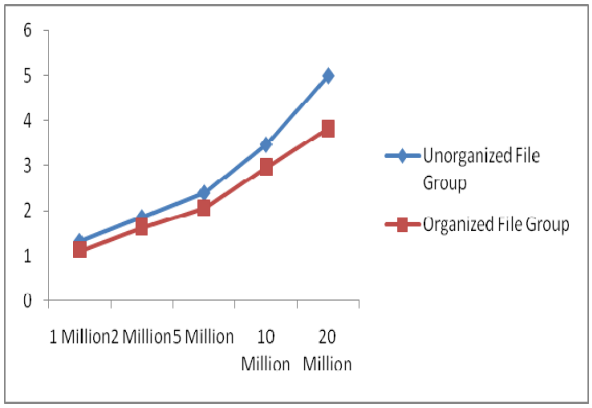

Figure.5: Comparison graph between “Organized File Group” and “Unorganized File Group”.

For easy understandability, let’s assume that, the table has an auto-increment primary key field (Say, ID). So, we can divide the table’s data into 10 separate portioning tables where each partition will contain 1 million rows and the partition will be based upon the value of the ID field. That is, First partition will contain those rows which have a primary key value in the range 1-1000000, and, Second partition will contain those rows which have a primary key value in the range 1000001-2000000 and so on.
Here we worked on millions of data with some complex query and got the results in seconds.

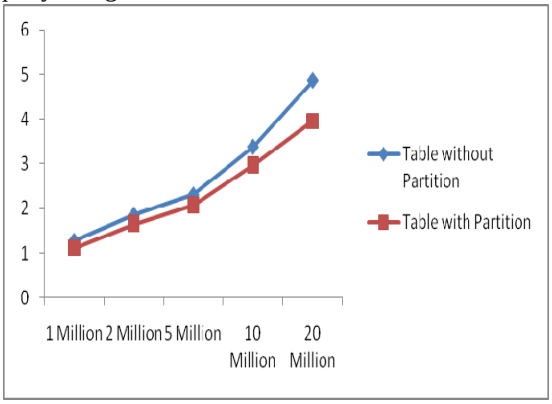

Figure.6: Comparison graph between “Table with Partition” and “Table without Partition”.

2.6 Identify inefficient TSQLs, re-factor and apply best practices

Knowing the best practices is not enough at all. The most important part is we have to make sure that we follow the best practices while writing TSQLs. Some TSQL Best practices are described here:
•We should not use “SELECT *” in SQL Query because then unnecessary columns may get fetched that adds expense to the data retrieval time and the Database engine

cannot utilize the benefit of "Covered Index" hence, query performs slowly.

•We should not use COUNT() aggregate in a subquery to do an existence check because when we use COUNT(), SQL Server does not know that we are doing an existence check. It counts all matching values, either by doing a table scan or by scanning the smallest nonclustered index. But if we use EXISTS, SQL Server knows you are doing an existence check. When it finds the first matching value, it returns TRUE and stops looking.

•We should try to avoid joining between two types of columns because when joining between two columns of different data types, one of the columns must be converted to the type of the other. The column whose type is lower is the one that is converted. If we are joining tables with incompatible types, one of them can use an index, but the query optimizer cannot choose an index on the column that it converts.

•We should try to avoid the use of Temporary Tables unless really required. Rather, try to use Table variables. Almost in 99% case, Table variables reside in memory; hence, it is a lot faster. But, Temporary tables reside in "TempDb" database. So, operating on Temporary table requires inter db communication and hence, slower.

•We should try to avoid deadlock. We should always access tables in the same order in all our stored procedures and triggers consistently and keep our transactions as short as possible. Also should touch as few data as possible during a transaction and should never, ever wait for user input in the middle of a transaction.

•We should write TSQLs using "Set based approach" rather than using "Procedural approach". The database engine is optimized for set based SQLs. Hence, procedural approach (Use of Cursor, or, UDF to process rows in a result set) should be avoided when large result set has to be processed. By using inline sub queries to replace User Defined Functions and by using correlated sub queries to replace Cursor based codes we can get rid of "Procedural SQLs"

•We should use Full Text Search for searching textual data instead of LIKE search as Full text search always outperforms the LIKE search. Full text search will enable us to implement complex search criteria that can't be implemented using the LIKE search such as searching on a single word or phrase, searching on a word or phrase close to another word or phrase, or searching on synonymous forms of a specific word.

•We should try to use "UNION" instead of "OR" in the query. If distinguished result is not required we better use "UNION ALL" because "UNION ALL" is faster than "UNION" as it does not have to sort the result set to find out the distinguished values.

Here we worked on millions of data with some complex query and got the results in seconds.

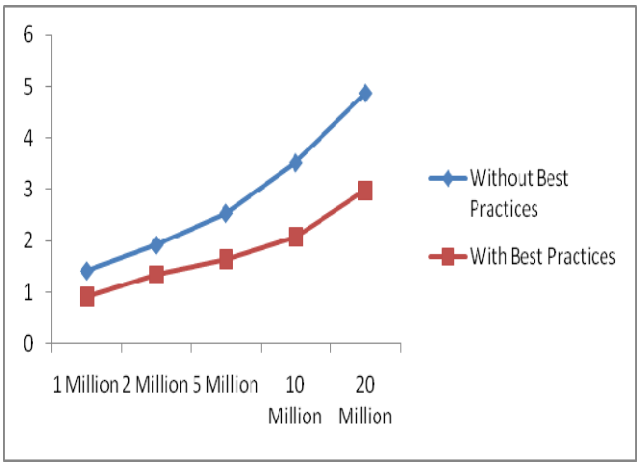

Figure7.: Comparison graph between "With Best Practices" and "Without Best Practices".

3. Experimental evaluation and performance study

In this section we present a performance comparison of a really huge database with millions of data. We did these experiments both ways, without any kind of optimization and with all our suggested optimizations.

Finally we worked on millions of data with few very complex query and store procedures. At first we didn't use any kind of optimization and then later we used all the suggested optimization techniques discussed above in our query and store procedure. Here X – axis indicates the number of data and Y – axis indicates the execution time of the query and store procedure in seconds.

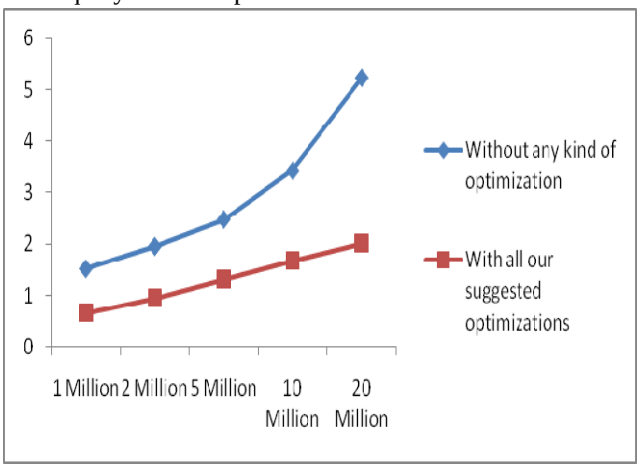

Figure.8: Comparison graph between "With all our suggested optimizations" and "Without any kind of optimization".

4. Conclusion

Here in this paper we have suggested very few performance optimization techniques in transactional (OLTP) SQL Server databases. Optimization is a "Mindset", rather than an automatic occurrence. In order to optimize our database performance, first we have to believe that, optimization is possible. Then we need to give our best effort and apply knowledge and best practices to optimize. The most important part is, we have to try to prevent any possible performance issue that may take place later, by applying our knowledge before or along with our development activity, rather than trying to recover after the problem occurs.

References

[1] Michael J. Hernandez, "Database Design for Mere Mortals", A Hands-On Guide to Relational Database Design, Addison-Wesley Professional, ISBN-10: 0201694719, ISBN-13: 978-0201694710, December 19, 1996.
[2] Clare Churcher, "Beginning Database Design: From Novice to Professional", Apress, ISBN-10: 1590597699, ISBN-13: 978-1590597699, January 15, 2007.
[3] C.J. Date, "An Introduction to Database Systems", Addison Wesley, ISBN-10: 0321197844, ISBN-13: 978-0321197849, August 1, 2003.
[4] E. F. Codd, "The relational model for database management: version 2", Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, ISBN:0-201-14192-2, 1990.
[5] C. J. Date, "Foundation for Future Database Systems: The Third Manifesto", Addison-Wesley Professional, ISBN-10: 0201709287, ISBN-13: 978-0201709285, 2000.
[6] http://www.websitedatabases.com/database-index.html
[7] http://www.simple-talk.com/sql/learn-sql-server/using-covering-indexes-to-improve-query-performance/

Syedur Rahman received the B.S. degree in Computer Engineering from North South University in 2007. During 2007-2008, he stayed in North South University (NSU) as a Teacher Assistant and as a Lab Instructor. From 2009 he is working as a Software Engineer in Jaxara It Limited (An USA based Software Company), Bangladesh.

A.M. Ahsan Feroz received the B.S. degree in Computer Science and Information Technology from Islamic University of Technology (IUT) in 2007. During 2007-2008, he worked as a research assistant on a research based firm. From 2009 he is working as a Software Engineer in M&H Informatics (An IMS Health Company), Bangladesh.

Md. Kamruzzaman received the B.Sc. degree in Computer Science and Engineering from Khulna University in 2005. After graduation, he joined EVOKNOW Bangladesh Ltd. and worked there as a Software Engineer for one year. Afterwards he joined United IT Global Net as a Software Engineer and stayed for about 8 months. Currently he is working as a Senior Software Engineer in Jaxara IT Limited (An USA based Software Company), Bangladesh.

Meherun Nesa Faruque accomplished the degree of Bachelor of Computer science in 2007 from BRAC University. She has worked as Junior Systems Analyst in a Project named RIRA under Maxwell Stamp (PLC) from 2007-2009. She is working as Software engineer at IMS health (Dhaka office) from 2009.