ISSN 2250-3153

SQL Server Query Optimization Techniques - Tips for Writing Efficient and Faster Queries

Navita Kumari

Innovation Centre - Baba Farid Group of Institutions

Abstract- SQL statements can be used to retrieve data from the any database. To get same results we need to write different SQL queries. For better performance we need to use best, faster and efficient queries. So we need SQL query tuning based on the business and user requirements. This paper covers how these SQL queries can be optimized for better performance. Query optimization subject is very deep but we will try to cover the most important points. In this paper I am not focusing on, in-depth analysis of database but simple query tuning tips & tricks which can be applied to gain immediate performance gain.

I. INTRODUCTION

The best way to tune performance is to try to write your queries in a number of different ways and compare their reads and execution plans. Here are various techniques that you can use to try to optimize your database queries. Query optimization is an important skill for SQL developers and database administrators (DBAs). In order to improve the performance of SQL queries, developers and DBAs need to understand the query optimizer and the techniques it uses to select an access path and prepare a query execution plan. Query tuning involves knowledge of techniques such as cost-based and heuristic-based optimizers, plus the tools an SQL platform provides for explaining a query execution plan.

II. QUERY PERFORMANCE OVERVIEW USING STATISTICS IO

There are different ways to determine the best way to write queries. Two of common methods are looking at the number of logical reads produced by the query and looking at graphical execution plans provided by SQL Server Management Studio. For determining the number of logical reads, you can turn the STATISTICS IO option ON. Consider this query:

```
SET STATISTICS IO ON

SELECT * FROM tablename
```

The following is returned in the Messages window in SQL Server Management Studio:

Table ‘tablename’. Scan count 1, logical reads 33, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

There are several bits of data returned by STATISTICS IO, but we are concerned with the logical reads portion because it will tell us the number of pages read from the data cache. This is the most helpful because it will stay constant when I run the same query, which is important because there are sometimes external factors that might vary the execution time of my queries, such as locking by other queries. When tuning SQL queries, our goal should be to get the number of logical reads as low as possible. As fewer logical reads typically lead to faster execution times.

III. GENERAL TIPS FOR QUERY OPTIMIZATION

*Specific Column Names instead of * in SELECT Query*

The SQL query becomes faster if you use the actual columns names in SELECT statement instead of than '*'. So we need to restrict the queries result set by selecting only the particular columns from the table, rather than all columns from a particular table. This results in performance benefits, as SQL Server will return only particular columns to the client, not all columns of a table. This will help reduce the network traffic and also boost the overall performance of the query.

Example: Write the query as

```
SELECT col_1, col_2, col_3, col_4, subject FROM
table_name;
```

Instead of:

```
SELECT * FROM table_name;
```

Alternatives of COUNT (*) for returning total tables row count

If we need to return the table's row count, we can use alternative ways instead of the SELECT COUNT (*) statement. As SELECT COUNT (*) statement makes a full table scan to return the table's row count, it can take much time for the large tables. There is another way to determine the total row count of a table. We can use sysindexes system table. There is a ROWS column in the sysindexes table. This ROWS column contains the total row count for each table in a particular database. So, we can use the following select statement instead of “SELECT COUNT (*): SELECT rows FROM sysindexes WHERE id = OBJECT_ID ('table_name') AND indid < 2”. So we can improve the speed of such queries thus resulting in better performance.

Try to avoid HAVING Clause in Select statements

HAVING clause is used to filter the rows after all the rows are selected and is used like a filter. Try not to use HAVING clause for any other purposes.
For Example: Write the query as

```
SELECT Col_1, count (Col_1)
FROM table_name
WHERE col_1!= ‘testvalue1’
```

```
AND col_1!= 'testvalue1'
GROUP BY col_1;
```

Instead of:

```
SELECT Col_1, count (Col_1)
FROM table_name
GROUP BY Col_1
HAVING Col_1!= 'testvalue1' AND Col_1!= 'testvalue2';
```

Try to minimize number of sub query blocks within a query

Sometimes we may have more than one sub query in our main query. We should try to minimize the number of sub query block in our query.

For Example: Write the query as

```
SELECT col_1
FROM table_name1
WHERE (col_2, col_3) = (SELECT MAX (col_2), MAX
(col_3)
FROM table_name2)
AND col_4 = 'testvalue1';
```

Instead of:

```
SELECT col_1
FROM table_name1
WHERE col_2 = (SELECT MAX (col_2) FROM
table_name2)
AND col_3 = (SELECT MAX (col_3) FROM table_name2)
AND col_4 = 'testvalue1';
```

Try to use operators like EXISTS, IN and JOINS appropriately in your query

a) Usually IN has the slowest performance.

b) IN is efficient, only when most of the filter criteria for selection are placed in the sub-query of a SQL statement.

c) EXISTS is efficient when most of the filter criteria for selection is in the main query of a SQL statement.

For Example: Write the query as

```
SELECT * FROM table1 t1
WHERE EXISTS (SELECT * FROM table2 t2 WHERE
t2.col_id = t1.col_id)
```

Instead of:

```
SELECT * FROM table1 t1
WHERE t1.col_id IN (SELECT t2.col_id FROM table2 t2)
```

Use EXISTS instead of DISTINCT when using table joins that involves tables having one-to-many relationships.

For Example: Write the query as

```
SELECT d.col_id, d.col2
FROM table1 d
WHERE EXISTS (SELECT 'X' FROM table2 e WHERE
e.col2 = d.col2);
```

Instead of:

```
SELECT DISTINCT d.col_id, d.col2
FROM table1 d, table2 e
WHERE e.col2 = e.col2;
```

Try to use UNION ALL instead of UNION, whenever possible.

The UNION ALL statement is faster than UNION, because UNION ALL statement does not consider duplicate s, and UNION statement does look for duplicates in a table while selection of rows, whether or not they exist.

For Example: Write the query as

```
SELECT id, col1
FROM table1
UNION ALL
SELECT id, col1
FROM table2;
```

Instead of:

```
SELECT id, col1, col2
FROM table1
UNION
SELECT id, col1
FROM table2;
```

We should try to carefully use conditions in WHERE clause.

For Example: Write the query as

```
SELECT id, col1, col2 FROM table WHERE col2 > 10;
```

Instead of:

```
SELECT id, col1, col2 FROM table WHERE col2 != 10;
```

Write the query as

```
SELECT id, col1, col2
FROM table
WHERE col1 LIKE 'Nav%';
```

Instead of:

```
SELECT id, col1, col2
FROM table
WHERE SUBSTR(col1,1,3) = 'Nav';
```

Write the query as

```
SELECT Col1, Col2
FROM table
WHERE Col3 BETWEEN MAX (Col3) and MIN (Col3)
```

Instead of:

```
SELECT Col1, Col2
FROM table
WHERE Col3 >= MAX (Col3)
and Col3 <= MIN (Col3)
```

We should try to use NON-Column expression on one side of the SQL query as it will be processed before any other clause.

For Example: Write the query as

```
SELECT id, Col1, Col2
FROM table
WHERE Col2 < 25000;
```

Instead of:

```
SELECT id, Col1, Col2
FROM Table
WHERE Col2 + 10000 < 35000;
```

IV. SOME MORE TIPS FOR OPTIMIZATION OF QUERIES/ TABLES/ STORED PROCEDURES IN SQL SERVER

a. Table should have minimum of one clustered index and appropriate number of non clustered indexes, which should be created on columns of table based on query which is running following the priority order as WHERE clause, then JOIN clause, then ORDER BY clause and finally the SELECT clause.
b. Avoid using Triggers if possible; incorporate the logic of trigger in a stored procedure.
c. Table should have a primary key.
d. Try to use constraints for selection instead of using triggers, whenever possible. Constraints are efficient than triggers enhance performance. So, you should use constraints instead of triggers, whenever possible.
e. Try to use table variables instead of temporary tables as table variables require less locking resources as well as less logging resources than the temporary tables, so table variables should be used whenever possible.
f. Avoid the use of views or replace views with original tables.
g. Try to avoid the use of DISTINCT clause, where ever possible. As the DISTINCT clause will result in performance degradation, we should use this clause only when it is necessary or unavoidable.
h. Try to add SET NOCOUNT ON statement into your stored procedures as it stops the message indicating the number of rows affected by a SQL statement.
It also reduces network traffic, because our client will not receive any message indicating the number of rows affected by a SQL statement.
i. Try to use TOP keyword or the SET ROWCOUNT statement in the select statements, if we need to return only the first n rows. This can improve performance of our queries, as the smaller result set will be returned. It can also reduce the traffic between the server and the clients.
j. Try to use user-defined functions to keep the encapsulated code for reuse in future.
The user-defined functions (UDFs) contain one or more SQL statements that can be used to encapsulate code for reuse. Using UDFs can also reduce the network traffic.
k. If possible move the logic of UDF to SP as well.
l. If you need to delete all rows of a table, try to use TRUNCATE TABLE command instead of DELETE command. Using the TRUNCATE TABLE is a faster way to delete all rows of a tables, because it removes all rows from a table without logging each row delete.
m. Remove any unnecessary joins from tables.
n. If there is cursor used in query, see if there is any other way to avoid the usage of this (either by SELECT … INTO or INSERT … INTO, etc).
Try to avoid using cursors whenever possible. As SQL Server cursors can result in some performance degradation as compared to select statements. Try to use correlated sub-queries or derived tables for row-by-row operations on tables.
o. When writing a sub-query (a SELECT statement within the WHERE or HAVING clause of another SQL statement):
    1. Try to use a correlated (refers to at least one value from the outer query) sub-query when the return is relatively small and/or other criteria are efficient i.e. if the tables within the sub-query have efficient indexes.
    2. Try to use a non-correlated (does not refer to the outer query) sub-query when dealing with large tables from which you expect a large return (many rows) and/or if the tables within the sub-query do not have efficient indexes.
    3. Ensure that multiple sub-queries are in the most efficient order.
    4. Remember that rewriting a sub-query as a join can sometimes increase efficiency.
p. Use char/varchar columns data type, instead of nchar/nvarchar data type if we do not need to store Unicode data. The char/varchar data value uses only one byte to store one character; whereas the nchar/nvarchar value uses two bytes to store one character, so the char/varchar columns use two times less space to store data as compared to nchar/nvarchar data columns.

q. Try to use stored procedures instead of heavy queries as they can reduce network traffic, because our client will send to server only stored procedure name (along with some parameters) instead of heavy and lengthy queries text. Stored procedures can be used to enhance security. For example, we can give different users, different set of permissions to execute the stored procedure to work with the restricted set of the columns and data.

r. We should try to return an integer value from a RETURN statement instead of returning an integer value as a part of a record set. As the RETURN statement exits from a stored procedure unconditionally, so the statements following the RETURN statement are not executed. The RETURN statement is generally used for error checking, but we can also use this statement to return an integer value for any other reason. Using RETURN statement can improve performance as SQL Server will not create a record set.

s. Try to drop indexes that are not being used. Because each index takes up disk space and slow the DML operations, we should drop indexes that are not used. We can use Index Wizard to identify indexes that are not being used in our SQL queries.

t. We should try to create indexes on columns that have integer values rather than character values. Because the integer values have less size than the size of characters values, so we can reduce the number of index pages which are being used to store the index keys. This finally reduces the number of reads required to read the index and enhances the overall index performance.

u. If we need to join several tables very frequently, then we should consider creating index on the joined columns which can significantly improve performance of the queries against the joined tables.

v. Try to avoid any operations on the fields, where ever possible. Some operations will prevent the use of index on a field even if it exists—for example, ltrim(rtrim(FieldColumnName)) as such operations will degrade the performance. For example, instead of using the condition cast(DateColumn as varchar(20)) = @dateStr, we should try to convert @dateStr to an expression of datetime type and then compare it to DateColumn value.

V. CONCLUSION

Query optimization has a very big impact on the performance of a DBMS and it continuously evolves with new, more sophisticated optimization strategies. Query optimization is a common task performed by database administrators and application designers in order to tune the overall performance of the database system. Even if you have a powerful infrastructure, the performance can be significantly degraded by inefficient queries. So, we should try to follow the general tips as mentioned above to get a better performance of queries. Optimization can be achieved with some efforts if we make it a general practice to follow the rules. The techniques described in this paper allow basic optimization of queries, tables, indexes and stored procedures for performance gains. The main focus was on query optimizations.

REFERENCES

[1] Sunderi Dejan, Microsoft SQL Server 2005 Stored Procedure Programming in T-SQL & .NET, Third Edition, McGraw-Hill, 2006, ISBN:0072262281.

[2] Dave Pinal, SQL SERVER – Optimization Rules of Thumb – Best Practices – Reader's Article. April 26, 2008. http://blog.sqlauthority.com/2008/04/26/sql-server-optimization-rules-of-thumb-best-practices-readers-article/

[3] Andrei Volkov, SQL Server Optimization, http://msdn.microsoft.com/en-us/library/aa964133(v=sql.90).aspx.

[4] SQL Server 2005 Books Online, Introducing SQL Trace. http://technet.microsoft.com/en-us/library/ms191006.aspx.

[5] SQL Server 2005 Books Online, Understanding plan forcing. http://msdn2.microsoft.com/en-us/library/ms186343.aspx.

[6] SQL Tuning or SQL Optimization. http://beginner-sql-tutorial.com/sql-query-tuning.htm

[7] SQL Server Optimization Tips. http://santhoshgudise.weebly.com/uploads/8/5/4/7/8547208/sql_server_optimization_tips-1.doc

[8] Optimize MS SQL Server. http://omarabid.hubpages.com/hub/Optimize-MS-SQL-Server

[9] SQL Spark. http://sqlspark.blogspot.in/2011/07/best-ways-to-write-sql-query.html

[10] MySQL Load Balancing PHP script. http://www.f1teksolutions.com/

[11] What are the most common SQL Optimizations. http://stackoverflow.com/questions/1332778/what-are-your-most-common-sql-optimizations

[12] Index Optimization Tips. http://www.mssqlcity.com/Tips/tipInd.htm

[13] Review, Change, or Delete Transferred or Imported data. http://turbotax.intuit.com/support/iq/Import/Review--Change--or-Delete-Transferred-or-Imported-Data/GEN12079.html.

AUTHORS

Navita Kumari, M.C.A., Innovation Centre - Baba Farid Group of Institutions and navitakamra@gmail.com