International Journal of Psychosocial Rehabilitation, Vol. 24, Issue 05, 2020
ISSN: 1475-7192

Tuning SQL Queries for Better Performance

S.N. Kavitha

Abstract--- *Structured Query Language (SQL) Statements are used to retrieve data from the database. We can get same results by writing SQL queries in a different way. But use of the best query is important when performance is considered. So we needSQL query tuning based on the business and user requirements. This paper is focusing on, simple query tuning tips & tricks which can be applied to gain better performance.*

Keywords--- *Structured Query Language, Tuning, Performance.*

I. INTRODUCTION

Database performance is one of the most challenging aspects of an organization's database operations. A well designed application may still experience performance problems if the SQL it uses is poorly constructed. It is much harder to write efficient SQL than it is to write functionally correct SQL. As such, SQL tuning can help significantly improve a system's health and performance. The key to tuning SQL is to minimize the search path that the database uses to find the data. The target audience of this paper includes developers and database administrators who want to improve the performance of their SQL queries.

II. LITERATURE REVIEW

In Navita Kumari (2012) summary article focusing on few techniques which is required to improve the performance of SQL queries. 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.

General SQL Tuning Guidelines

The goals of writing any SQL statement include delivering quick response times, using the least CPU resources, and achieving the fewest number of I/O operations. The following content provides best practices for optimizing SQL performance.

Tuning Sub Queries

If the SQL contains subqueries, tune them. In fact, tune them fist. The main query will not perform well if the subqueries can't perform well themselves. If a join will provide you with the functionality of the subquery, try the join method fist before trying the subquery method. Pay attention to correlated subqueries, as they tend to bevery costly and CPU- incentive.

Alias Usage

If an alias is not present, the engine must resolve which tables own the specified columns. A short alias is parsed

S.N. Kavitha, Assistant Professor, New Horizon College of Engineering, Bangalore, Karnataka, India.
E-mail: kavijadhav86@gmail.com

DOI: 10.37200/IJPR/V24I5/PR2020703
Received: 16 Apr 2020 | Revised: 22 Apr 2020 | Accepted: 12 May 2020

more quickly than a long table name or alias. If possible, reduce the alias to a single letter. The following is an example:

BadStatement

SELECT first_name, last_name, country FROM employee, countries WHERE country_id = id AND last_name = 'HALL';

Good Statement

SELECT e.first_name, e.last_name, c.country FROM employee e, countries c WHERE e.country_id = c.idAND e.last_name = 'HALL';

Avoid Using Select * Clauses

The dynamic SQL column reference (*) gives you a way to refer to all of the columns of a table. Do not use the* feature because it is very inefficient -- the * has to be converted to each column in turn. The SQL parser handles all the field references by obtaining the names of valid columns from the data dictionary and substitute them on the command line, which is time consuming.

Exists vs. In

The EXISTS function searches for the presence of a single row that meets the stated criteria, as opposed to the IN statement that looks for all occurrences.

For example:

PRODUCT - 1000 rows ITEMS - 1000 rows

(A) SELECT p.product_id FROM products p WHERE p.item_no IN (SELECT i.item_no FROM items i);

(B) SELECT p.product_id FROM products p WHERE EXISTS (SELECT '1' FROM items I WHERE i.item_no = p.item_no);

For query A, all rows in ITEMS will be read for every row in PRODUCTS. The effect will be 1,000,000 rows read from ITEMS. In the case of query B, a maximum of 1 row from ITEMS will be read for each row of PRODUCTS, thus reducing the processing overhead of the statement.

Not Exists vs. Not In

In subquery statements such as the following, the NOT IN clause causes an internal sort/ merge.

SELECT * FROM student WHERE student_num NOT IN (SELECT student_num FROM class);

Instead, use:

SELECT * FROM student c WHERE NOT EXISTS (SELECT 1 FROM class a WHERE
a.student_num =c.student_num);

Using Union in Place of Or

In general, always consider the UNION verb instead of OR verb in the WHERE clauses. Using OR on an indexed column causes the optimizer to perform a full-table scan rather than an indexed retrieval.

Union All Instead of Union

The SORT operation is very expensive in terms of CPU consumption. The UNION operation sorts the result setto eliminate any rows that are within the sub-queries. UNION ALL includes duplicate rows and does not require a sort. Unless you require that these duplicate rows be eliminated, use UNION ALL.

Using Indexes to Improve Performance

Indexes primarily exist to enhance performance. But they do not come without a cost. Indexes must be updated during INSERT, UPDATE and DELETE operations, which may slow down performance. Some factors to consider when using indexes include:

- Choose and use indexes appropriately. Indexes should have high selectivity. Bitmapped indexes improve performance when the index has fewer distinct values like Male or Female.
- Avoid using functions like “UPPER” or “LOWER” on the column that has an index. In case there is no way that the function can be avoided, use Functional Indexes.
- Index partitioning should be considered if the table on which the index is based is partitioned. Furthermore, all foreign keys must have indexes or should form the leading part of Primary Key.
- Occasionally you may want to use a concatenated index with the SELECT column. This is the most favored solution when the index not only has all the columns of the WHERE clause, but also the columns of the SELECT clause. In this case there is no need to access the table. You may also want to use a concatenated index when all the columns of the WHERE clause form the leading columns of the index.
- When using 9i, you can take advantage of skip scans. Index skip scans remove the limitation posed by column positioning, as column order does not restrict the use of the index.
- Large indexes should be rebuilt at regular intervals to avoid data fragmentation. The frequency of rebuilding depends on the extents of table inserts

III. CONCLUSION

Eighty percent of your database performance problems arise from bad SQL. Designing and developing optimal SQL is quintessential to achieving scalable system performance and consistent response times. The key to tuning often comes down to how effectively you can tune those single problem queries.

REFERENCES

[1] Shalabh Mehrotra-Senior Solutions Architect- SQL Performance Tuning

[2] Navita Kumari-SQL Server Query Optimization Techniques - Tips for Writing Efficient and Faster Queries, *International Journal of Scientific and Research Publications,* Volume 2, Issue 6, June 2012.

[3] Oracle Performance Tuning 101 by Gaja Krishna Vaidyanatha, Kirtikumar Deshpande and John Kostelac

[4] Oracle 9I Documentation

[5] SQL Server 7.0 Query Processor

[6] Dean Richards, Manager Sales Engineering, Senior DBA- SQL Query Tuning for SQL Server Index Tuning Wizard for SQL Server 7.0

[7] http://docs.oracle.com/html/A86647_01/vmqtune.htm

DOI: 10.37200/IJPR/V24I5/PR2020703
Received: 16 Apr 2020 | Revised: 22 Apr 2020 | Accepted: 12 May 2020

[8] http://www.dba-oracle.com/art_sql_tune.htm

[9] https://technet.microsoft.com/en-us/library/ms172984(v=sql.110).aspx

[10] http://www.dba-village.com/dba/village/dvp_papers.Main?CatA=45

[11] http://sqlmag.com/database-performance-tuning/sql-server-white-papers