Query Optimization in SQL

CSCI 21062 Advanced database management systems

Assignment 01

Tharushi Udawaththa – CS/2019/050

Query Optimization in SQL

1.0 Definition

“SQL Query optimization is defined as the iterative process of enhancing the performance of a query in terms of execution time, the number of disk accesses, and many more cost measuring criteria” [1].

2.0 The need of query optimization in SQL

Modern databases are intricate beings with numerous potentials for setup or hardware provisioning errors to result in poor performance. Despite various efforts to increase performance, poorly constructed SQL queries can slow down database response time. SQL queries must be optimized as a result [2].

3.0 Introduction to SQL

SQL (Structured Query Language) is a standard language for all data-related operations, including accessing, storing, creating, and deleting databases [1]. In SQL, there are four different language types:

3.1 DDL (Data Definition Language)

This offers instructions for establishing and changing the database's schema. For instance, CREATE, DROP, ALTER, and TRUNCATE [1].

3.2 DML (Data Manipulation Language)

This offers instructions for modifying database data. It modifies the relation's instance. SELECT, UPDATE, INSERT, and DELETE are a few examples [1].

3.3 DCL (Data Control Language)

This includes instructions for database access. For instance, GRANT and REVOKE [1].

3.4 TCL (Transaction Control Language)

This covers the commands used to control database transactions. As an illustration, COMMIT, ROLLBACK [1].

4.0 Query optimization techniques

There are some effective cost-cutting techniques. However, optimization is an iterative process. The query must first be written, its performance evaluated using IO statistics or an execution plan, and then it must be optimized. Iteratively following this cycle is necessary for query optimization. Additionally, the SQL Server determines the best and simplest plan to carry out the query [1].

4.1 Indexing

A data structure called an index is utilized to give users easy access to a table using a search key. It aids in reducing the amount of disk access required to get the database records. A scan or a search can be used as an indexing operation. While an index seeks filters rows using a matching filter, an index scan searches the entire index for matching criteria [1].

4.2 Selection

Instead of picking all the rows, only the necessary rows should be chosen. Because SELECT * examines the entire database, it is quite inefficient [1].

4.3 Avoid using SELECT DISTINCT

In SQL, the SELECT DISTINCT command is used to retrieve distinct results and eliminate duplicate rows from a relation. It essentially joins together comparable rows in order to accomplish this objective, then deletes them. The GROUP BY procedure is expensive. Therefore, one may add extra properties to the SELECT process to retrieve separate results and eliminate duplicate data [1].

4.4 Inner joins vs WHERE clause

Instead of utilizing the WHERE clause to combine two or more tables, we should utilize inner join. When merging tables, the WHERE clause generates the CROSS join/CARTESIAN product. The CARTESIAN product of two tables is time-consuming [1].

4.5 LIMIT command

The limit command is used to limit how many rows from the result set are displayed. Only the necessary rows must be shown in the result set. In order to give an on-demand computation of rows for the production purpose, one must use limit with the production dataset [1].

4.6 IN versus EXISTS

In terms of scan costs, the IN operator is more expensive than EXISTS, particularly when the output of the subquery is a huge dataset. So when retrieving results with a subquery, we should try to use EXISTS rather than IN [1].

4.7 Loops versus Bulk insert/update

Due to the need to repeatedly run the same query, loops must be avoided. We should choose bulk updates and inserts instead [1].

5.0 SQL Query Optimization Tips with Examples

5.1 **Add missing indexes**

Database table indexes facilitate quicker and more effective information retrieval.When you run a query in SQL Server, the optimizer creates an execution plan. The execution plan suggests this in the warning section if it discovers a missing index that might be established to improve performance. This recommendation tells you which columns in the current SQL should be indexed and how performance will be enhanced after that [3].

5.2 **Check for unused indexes**

There could be instances where indexes are there but not being used. Implicit data type conversion can be one of the causes. The CAST() function, which changes a value of any type into a given datatype, is advised. See the question below [3].

SELECT * FROM TestTable WHERE IntColumn = CAST(@char AS INT);

5.3 **Avoid using multiple OR in the FILTER predicate**

It is advised to avoid using the OR operator or to divide the query into sections that are separated by search expressions when you need to combine two or more conditions. OR cannot be processed by SQL Server in a single operation. Instead, it assesses each OR component, which could result in subpar performance. The query will be optimized and SQL Server will be able to use the indexes if we divide it into two SELECT queries and join them using the UNION operator [3].

SELECT * FROM USER WHERE Name = @P UNION SELECT * FROM USER WHERE login = @P;

5.4 **Use wildcards at the end of a phrase only**

Wildcards can be used at the start or end of words and phrases to act as placeholders. You can make use of wildcards in the SELECT statement at the conclusion of a phrase to speed up and improve the efficiency of data retrieval [3]. For instance:

SELECT p.BusinessEntityID ,p.FirstName ,p.LastName ,p.Title FROM Person.Person p WHERE p.FirstName LIKE 'And%';

5.5 **Avoid too many JOINs**

A query could become overloaded if you join numerous tables to it. Additionally, a large number of tables from which to retrieve data may lead to an ineffective execution strategy. The SQL query optimizer must specify how the tables are joined, in which sequence, how and when to apply filters, and when to use aggregation when creating a plan [3].

One of the various methods for creating effective query plans is JOIN removal. By dividing a single query into multiple independent ones that can subsequently be linked, you can eliminate extra joins, subqueries, tables, etc [3].

5.6 **Avoid using SELECT DISTINCT**

This might necessitate the tool processing enormous amounts of data, which might slow down the query. In general, it is advised to just execute the SELECT statement while specifying columns rather than utilizing the SELECT DISTINCT option [3].

5.7 **Use SELECT fields instead of SELECT ***

Data from the database is retrieved using the SELECT query. It is not advised to obtain all data from large databases because doing so would require more resources to query such a massive amount of data. Instead, you can narrow down which precise columns you require data from in order to conserve database resources. In this scenario, SQL Server will just retrieve the necessary data, resulting in a less expensive query [3].

```
SELECT FirstName ,LastName ,Email ,Login FROM Users;
```

5.8 **Use TOP to sample query results**

Setting a limit on the number of records the database can return can be done with the SELECT TOP command. You can use this command to retrieve a sample set of few rows to ensure that your query will produce the desired result [3].

```
SELECT TOP 5 p.BusinessEntityID ,p.FirstName ,p.LastName ,p.Title FROM Person.Person p WHERE
p.FirstName LIKE 'And%';
```

6.0 Conclusion

Some other things to keep in mind:

1. Avoid using correlated nested queries.
2. Avoid inner joins with Equality or OR conditions.
3. Check whether records exist before fetching them.
4. Use indexing properly, try to create more indexes for fetching composite columns.
5. Use Wildcards wisely.
6. Try to use WHERE rather than HAVING. Only use HAVING for aggregated values.

Consequently, we discovered how even little adjustments to a query can significantly boost its performance. Applications' performance will be improved, improving user experience. When drafting enquiries, remember to follow all the rules [1].

7.0 References

[1] S. Sachdeva, "A Detailed Guide on SQL Query Optimization," Analytics Vidhya, 5 Oct 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/a-detailed-guide-on-sql-query-optimization/. [Accessed 21 Dec 2022].

[2] R. Agar, "Why It’s Important to Optimize SQL Queries," IDERA, 5 Mar 2021. [Online]. Available: https://blog.idera.com/database-tools/why-it-s-important-to-optimize-sql-queries/. [Accessed 23 Dec 2022].

[3] d. Team, "SQL Query Optimization: How to Tune Performance of SQL Queries," devart, 23 Dec 2021. [Online]. Available: https://blog.devart.com/how-to-optimize-sql-query.html. [Accessed 23 Dec 2022].