# School of Computer Science and Information Systems - Northwest Missouri State University
# Advanced Optimization in SQL Query using Tuning to Reduce Time Consumption

Greeshma Jale – S555082, Sai Krishna Koruprolu – S559218
Tejaswini Kotha – S560283, Niharika Polu – S559357
Keerthi Sannapareddy – S559469

**Abstract**

The 21st century has witnessed remarkable technological advancements, revolutionizing the storage and processing of data. Traditional paper and file-based systems have given way to sophisticated and innovative technologies. Businesses and organizations are continually adapting to these changes to stay ahead of the competition and meet evolving customer needs. As data sets become increasingly large, the need for correspondingly large databases arises. Managing data in such databases often involves multiple operations, which can be time-consuming. Therefore, it is crucial to make these operations simpler and ensure high-quality service to ensure the efficient management of large databases. The research focus on mainly reducing the time consumption for SQL Query execution using physical tuning method which makes changes to database structure and logical tuning method which re-writes sub-queries. The objective of this paper is to provide an overview of SQL query tuning techniques and best practices, and to equip database administrators and developers with the knowledge and tools necessary to optimize SQL queries for improved database performance.

**Keywords:** Optimization Techniques, SQL Query Processing, Tuning methods

## 1. INTRODUCTION:

Structured Query Language (SQL) is a powerful programming language that is widely used in managing relational databases. It is an essential tool for businesses that rely on large amounts of data to make critical decisions. However, inefficient SQL queries can result in poor performance and increased hardware requirements, which can be costly for businesses. As a result, SQL query optimization has become increasingly important in the present industry to ensure that database systems run efficiently and effectively. This paper explores SQL query optimization using physical and logical tuning methods.

In the present industry, SQL query optimization has become a critical task for businesses that rely on data-driven decision making. With the increase in the volume of data generated by businesses, optimizing SQL queries has become essential to ensure that database systems can handle the load efficiently. By using physical and logical tuning methods to optimize SQL queries, businesses can improve system performance, reduce hardware requirements, and ultimately save costs.

SQL query optimization is the process of improving the performance of SQL queries by reducing their execution time and resource usage. It involves analyzing the structure of the database, the schema design, and the query itself to identify areas of inefficiency and opportunities for optimization. “There are two main categories of SQL query optimization: physical and logical tuning“

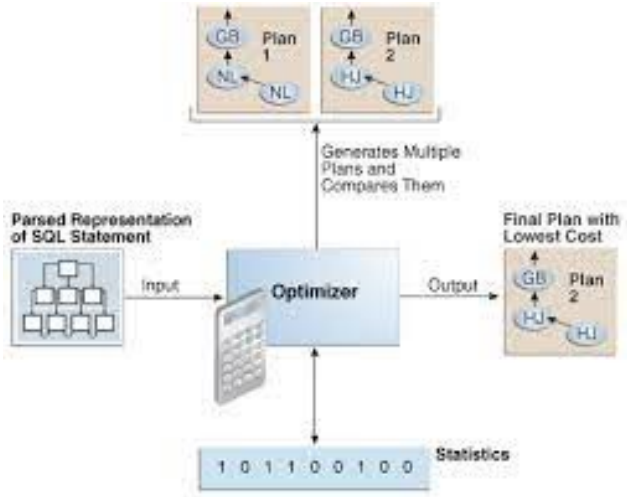
Physical tuning involves optimizing the physical layout of the database, such as indexing and partitioning, to reduce the number of disk accesses required for query processing. This type of optimization can have a significant impact on query performance, as it reduces the amount of time required to access data from the database. Logical tuning, on the other hand, involves optimizing the query structure and database schema to improve query performance. This may involve rewriting queries to use more efficient joins or changing the schema design to eliminate redundant data.

Even though there are many query optimizing methods available in tuning we go with query optimizer because using this method we can also execute join statements in our SQL query

efficiently in such a way, SQL query executes in very less time and then saving system resources.

**2. Query optimizer:**

When performing a query improvement, it is recommended that we use table-based cardinalities. During this process, there are three methods as discussed above which we apply to attain effectiveness. Applying DBS has it pill-based with line practicality we also have other methods of handling queries a part of the above which are referred to as query handling techniques known as communicated query management, which indicates that several surges of data coming in from the source are cycled into improvement strategies. Query Processing II is another type of query preparation that includes non-interfering administrators, rough calculations, sliding window calculations, and online information stream mining. Window, steady development, and exploiting stream imperatives are three ways used by non-hindering administrators to unblock stream administrators. Windowed inspection and symmetric hash join are two types of sliding window calculations. The board mentions figure stream markers, anticipating, choice trees, relapse evaluation, design coordinating, and similitude recognition while discussing online information streams. Sliding windows, bundled handling, inspecting and abstracting, and inhibiting administrators are some of the queries noting strategies. Clusters, preparation, and rundown are the three types of bunch handling. In testing, the update is moderate, but the figuring is lightning fast, and the outline hints at the extensive inquiry for genuine data. Administrators who are obstructing planning, typical, and minimum abilities. Query Transformation is a basic level of query advancement that involves converting a SQL statement into a Query Tree or Query Diagram and then reworking the query using heuristic question handling methods. The following guidelines were used to send the Query change: Perform early selection and projection duties, Perform more modest activities. While doing back-to-back join activities, join first, calculate common statements only once and preserve the results.




However, with the increasing speed of the Selection plan time, the query articulation can be further reduced, as the complexity of competing plans will considerably due to the complexity of the query more efforts are required to investigate the query plan rather than focusing on the results generated by the DBS. For optimal results, a software optimizer has to establish a balance between optimal execution that consumes time and an option for a shorter time. If the query is not machine code as it is in most cases it is required that compiler optimization be taken as the first choice as it will increase SQL query optimization efficiency. The next section focuses on a comprehensive approach for the choice of plans for a complicated SQL query with several tasks that would yield better execution. We also realize that if a combined system of the revolutionized system possesses the combination of compiler technology and the SQL optimization for effective execution of SQL queries in large datasets.

**2.1 Skyline dynamic Programming:**

Dynamic Programming (DP) is a commonly used approach for plan selection. In a shorter amount of time, it can count and recognize the best plan. DPSA performs well when finding results for relatively less complicated queries if there is an increase in the complexity of the query then there is observed reduced performance of the DP similar to IDP which performs well when the subsets of executing plans are reduced. Then, we introduce **Skyline Dynamic Programming (SDP)**, an outstanding DP algorithm in this presentation. In a SQL question, the Skyline operator acts as a channel. It keeps the goods that aren't as bad as the rest. For example, while booking international

travel from the United States to China, we may prefer the route with the fewest stops; yet, a non-stop flight is quiet it requires resources because the operator can only show flight options that have a good comparison to others in terms of cost. This technique has a novel metric in that; it takes parts of the join graph rather than focusing on the entire join graph and incorporates multiple approaches of pruning strategy based on a combination of Selectivity, Cardinality, and Costs. In comparison to original DP calculations, it saves time and space in execution.

### 2.2. Executing Join statements using Query Optimizer:

When executing a join statement, the query optimizer needs to choose an access path to retrieve data from each table, select a join method (such as nested loop, sort merge, cartesian, or hash join), and determine the join order. To optimize the join operation, the optimizer first looks for tables that have UNIQUE and PRIMARY KEY constraints and places them at the beginning of the join order. Next, the optimizer optimizes the join of the remaining tables and determines the cost of the join using various methods.

Even though there are many query optimizing methods available in tuning we go with query optimizer because using this method we can execute join statements in our SQL query efficiently in such a way, SQL query executes in very less time and then saving system resources.

For large data sets with tables related by an equality condition join, the optimizer uses hash joins. This involves building a hash table on the join key in memory using the smaller of the two tables and then scanning the larger table to find the joined rows. This method is most effective when the smaller table can fit in available memory, as the cost is limited to a single read pass over the data for the two tables.

## 2.2.1. TYPES OF JOINS:

Mainly there are three types of joins based on the way they retrieve the records. They are Inner join, Outer join and joining more than two tables.

**Inner Join:** An inner join (sometimes called a simple join) is a join of two or more tables that returns only those records that satisfy the join condition. i.e it is looking for only the matching

records.

**Outer Join:** An outer join extends the result of a simple join. An outer join returns all records that satisfy the join condition and also returns some or all of those records from one table for which no record from the other table satisfy the join condition. In other words, first of all it is looking for matching records and remaining records (usually with null values) from one table or both the tables.

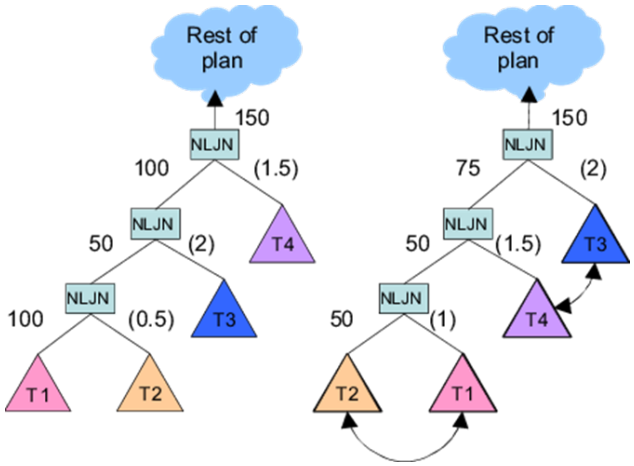
**Joining more than Two tables:** It is a generalized joining process in which number of tables is exceeding by two. Here, the minimum number of joining conditions required depends on the number of tables being joined. To join N number of tables, at minimum, we need (N – 1) joining conditions.

## 2.3. Nested Loop Joins:

Nested loop joins are a basic join algorithm that compares each row from one table with each row from the other table based on the join condition. The algorithm works by iterating over each row in the first table and for each row, it searches the second table for matching rows using the join condition. The join condition is typically based on equality comparisons between columns in the two tables. Nested loop joins are most efficient when one of the tables is small or when there is an index on the join key. When one table is small, the nested loop join algorithm can iterate over the entire table relatively quickly, and for each row, it only needs to search a small portion of the second table. When there is an index on the join key, the database system can use the index to quickly find the matching rows in the second table. However, nested loop joins can become slow when both tables are large. When both tables are large, the algorithm needs to perform a large number of comparisons, which can be computationally expensive. Additionally, nested loop joins require multiple passes over the data, which can result in a high number of I/O operations.

To improve the performance of nested loop joins, database systems can use techniques such as block nested loop joins or indexed nested loop joins. Block nested loop joins work by processing data in blocks, rather than processing one row at a time. This can reduce the number of I/O operations and improve performance. Indexed nested loop joins work by using an index on the

join key in the inner table, which can significantly reduce the number of comparisons required. In general, nested loop joins are a useful join algorithm for small or medium-sized tables, or when there is an index on the join key. However, for larger tables, more efficient join algorithms such as merge joins or hash joins may be more appropriate. The choice of join algorithm depends on factors such as the size of the tables, the available indexes, and the specific join conditions.




Nested loop plan

Nested loop joins are a type of join algorithm that compares each row from one table with each row from the other table based on the join condition. This algorithm is efficient when one of the tables is small or when there is an index on the join key. However, it can become slow when both tables are large.

For example, consider the following two tables:

Table A:

| id | name | age |
|---|---|---|
| 1 | John | 25 |
| 2 | Jane | 30 |
| 3 | Mark | 40 |

Table B:

| id | city | country |
|---|---|---|
| 1 | Paris | France |
| 2 | Tokyo | Japan |

Suppose we want to join Table A and Table B on the 'id' column. The query optimizer can choose to use a nested loop join in this case, as Table B is small and there is an index on the 'id' column in both tables. The join operation would look like:

SELECT *

FROM Table A

INNER JOIN Table B

ON Table A.id = Table B.id;

The query optimizer would then generate an execution plan that compares each row from Table A with each row from Table B based on the 'id' column. The matching rows would be returned as the result set. In this example, the result set would be:

| id | name | age | city | country |
|---|---|---|---|---|
| 1 | John | 25 | Paris | France |
| 2 | Jane | 30 | Tokyo | Japan |

Thus, the nested loop join algorithm can efficiently execute join statements when one of the tables is small or when there is an index on the join key.

## 2.4 Merge Joins:

Merge joins are a join algorithm that is more efficient than nested loop joins when both tables are sorted on the join key. The merge join algorithm works by sorting both tables on the join key and then merging the rows from both tables based on the join condition. This algorithm is efficient

because it requires less CPU and I/O operations than nested loop joins.

Let's consider an example to understand merge joins better. Suppose we have two tables, "orders" and "customers," with the following schema:

orders(order_id, customer_id, order_date, total_amount) customers(customer_id, customer_name, city, country).

customers(customer_id, customer_name, city, country)

Suppose we want to join these tables on the "customer_id" column. To perform a merge join, we first need to sort both tables on the "customer_id" column. Once both tables are sorted, we can start merging the rows from both tables based on the join condition. The merge join algorithm works by comparing the values of the join key in both tables and merging the matching rows.

For example, suppose we have the following data in the "orders" table:

| Order id | Customer id | Order date | Total amount |
|---|---|---|---|
| 1 | 101 | 2022-01-01 | 100.00 |
| 2 | 102 | 2022-01-02 | 150.00 |
| 3 | 103 | 2022-01-03 | 200.00 |

And the following data in the "customers" table:

| Customer ID | Customer Name | City | Country |
|---|---|---|---|
| 101 | John Doe | New York | USA |
| 102 | Jane Doe | Los Angeles | USA |
| 103 | Bob Smith | Chicago | USA |

To perform the merge join, we would first sort both tables on the "customer_id" column. After sorting, the tables would look like this:

orders(order_id, customer_id, order_date,total_amount)

$1 \,||101|2022-01-01|100.00$

$2 \,||102|2022-01-02|150.00$

3 ||103|2022 − 01 − 03|200.00

customers(customer_id, customer_name, city, country)

101 ||*JohnDoe*|*NewYork*|*USA*

102 ||*JaneDoe*|*LosAngeles*|*USA*

103 ||*BobSmith*|*Chicago*|*USA*

Once the tables are sorted, we can start merging the rows from both tables based on the join condition. The merge join algorithm compares the values of the join key in both tables and merges the matching rows. In this example, the result of the merge join would be:

order_id ||*customer_id*|*order_date*|*total_amount*|*customer_name*|*city*|*country*

1 ||101|2022 − 01 − 01|100.00|*JohnDoe*|*NewYork*|*USA*

2 ||102|2022 − 01 − 02|150.00|*JaneDoe*|*LosAngeles*|*USA*

3 ||103|2022 − 01 − 03|200.00|*BobSmith*|*Chicago*|*USA*

In summary, merge joins are an efficient join algorithm when both tables are sorted on the join key. The algorithm works by sorting both tables and then merging the rows based on the join condition. The choice of join algorithm depends on factors such as the size of the tables, the available indexes, and the specific join conditions.

## 3. CONCLUSION:

In this paper, the concept of executing join statements with query optimization including nested loop joins & merge joins are examined. For starters, we show how Skyline Dynamic Programming (SDP) may enhance execution times when compared to normal DP calculations, especially in complicated query circumstances with several JOIN processes. Then, for the query execution section, we explain how join statements are executed using the query optimization technique. And we also explained how complex joins like nested loop joins and merge joins SQL query are optimized before being executed, this overall methods reduce time taking for executing a SQL joint query.

## 4.References

1) Oracle Database SQL Reference 10g Release 1 (10.1), Documentation

2) Fundamentals of Database Systems, Fifth Edition, by Ramez Elmasri, Shamkant B.Navathe, Pearson Publications, 2009.

3)Database Management Systems, By – Raghu rama krishnan, Gehrke, Third Edition, McGraw-Hill Publications, 2003.

4) Pratt, Phillip J (2005), A Guide To SQL, Seventh Edition, Thomson Course Technology, ISBN 978-0-619-21674-0

5) Shah, Nilesh (2005) [2002], Database Systems Using Oracle – A Simplified Guide to SQL and PL/SQL Second Edition (International ed.), Pearson Education International, ISBN 0-13-191180-5

6) Yu, Clement T.; Meng, Weiyi (1998), Principles of Database Query Processing for Advanced Applications, Morgan Kaufmann, ISBN 978-1-55860-434-6, retrieved 2009-03-03

7)M. Tamer Özsu; Patrick Valduriez (2011). Principles of Distributed Database Systems (3rd ed.). Springer. p. 46. ISBN 978-1-4419-8833-1.

8) C. J. Date (2011). SQL and Relational Theory: How to Write Accurate SQL Code. O'Reilly Media, Inc. pp. 133– 135. ISBN 978-1-4493-1974-8.

9) Chaudhuri, S., Shim K. Optimization of Queries with User defined Predicates. In Proc. of VLDB, Mumbai, 1996.

10) Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim K. Optimizing Queries with Materialized Views. In Proc. of IEEE Data Engineering Conference,Taipei, 1995.

11) Chaudhuri, S., Shim K. Including Group-By in Query Optimization. In Proc. of VLDB, Santiago, 1994