International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Performance Optimisation of SQL Queries in Data Warehouses Enhancing SQL Query Performance through Indexing, Query Rewriting, and Materialised Views in Data Warehouses

[1]Subash Yadav, [2]Mehul Vani

Submitted: 10/11/2024 **Revised**: 20/11/2024 **Accepted**: 22/11/2024

Abstract: With the rapid growth of data warehouses, efficient SQL query processing has become crucial for handling complex analytical workloads, particularly those involving large-scale joins, aggregations, and filtering operations. Poorly optimised queries can lead to excessive execution time, high I/O costs, and inefficient resource utilization, negatively affecting business intelligence systems. This study presents a comprehensive framework for SQL query optimisation in relational databases, combining indexing strategies, query rewriting techniques, materialised views, and execution plan analysis. The research utilizes a retail transaction dataset modeled in a star schema to simulate real-world analytical queries. Experimental evaluations were conducted to compare baseline and optimised queries across several performance metrics, including execution time, rows scanned, and optimiser cost. The results demonstrate significant performance improvements, including reduced query latency, fewer full table scans, and enhanced index utilisation. The use of materialised views provided near-instantaneous responses for repetitive aggregation tasks, while indexing and query rewriting enhanced the efficiency of join and filter operations. This study bridges the gap between theoretical optimisation concepts and practical implementation, offering a validated and integrated approach to SQL query optimisation for data warehouses.

Keywords: SQL Query Optimisation, Data Warehouse Performance, Star Schema, Indexing Strategies, Query Rewriting Techniques, Materialised Views

I. INTRODUCTION

Data warehouses have become key elements of the current business intelligence (BI) systems and they have played a central role in facilitating organisations to centralise large volumes of historical business data and convert it into actionable business insights to enable its use in strategic decision-making [1, 2]. They are centralised storage locations that bring together data across various sources of operation and are subject to the fundamental tenets of being subject-oriented, integrated, time-varying and non-volatile [3, 4]. This is an architectural basis that enables a business to have an overall picture of its past activities, which is critical in long term analysis and prediction. Data warehouses are especially useful in the retail industry, where they aid in vital activities like sales trend analysis, inventory management, customer segmentation, and demand forecasting [5]. Likewise, they are used to support complex multi-dimensional aggregations in financial reporting and customer behaviour analysis to help organisations unravel hidden patterns and make informed business strategies [6].

[1]*affiliation Josla Tech LLC*
ORCID iD: 0009-0007-1643-7695
email ysubash@joslatech.com
[2]*affiliation: independent researcher*
Orcid : 0009-0002-2772-0393
Mehul1.vani@gmail.com

There is a major difference between the Online Transactions Processing (OLTP) and Online Analytical Processing (OLAP) data warehouses. According to [7], OLTP systems are meant to support high-concurrency, short-duration transactions having highly normalised data structures centred on data integrity and the day-to-day operations of the business. In comparison, OLAP data warehouses use denormalised data structures, particularly star or snowflake data structures, tailored to the read-intensive workload of analytics [8]. The SQL queries used in these OLAP systems are usually large fact tables combined with several dimension tables, and have extensive grouping and aggregation functions like SUM, COUNT, and GROUP BY [9]. These queries are resource-intensive by nature and require high performance to provide responsive, interactive BI dashboards and timely reporting. With the data volumes constantly growing exponentially, query processing efficiency is an important aspect of ensuring the overall effectiveness of business intelligence efforts.

Data warehouses are important, but they usually pose serious performance issues. They typically have millions or even billions of records and the performance of complex SQL queries is especially challenging. Analytical queries often involve joining multiple tables between fact and dimension tables, aggregation and filtering. As the database optimiser makes inefficient query execution plans, the system can turn to full table scans, which causes excessive input/output (I/O) operations, long execution time, and high

computation costs. These performance bottlenecks not only reduce the responsiveness of BI dashboards but also restrict business agility by slowing down essential insights. The problem of unoptimised joins and aggregations resulting in unnecessary scanning of rows was even evident in the small retail data used in this experiment, where they produced significant scanning of unnecessary rows, which is even more severe in the large-scale production environment.

Despite extensive literature on distributed query engines like Apache Spark and Presto and hardware-level optimisations, the gap in the literature on empirical research on practical, query-level SQL optimisation techniques implemented directly within traditional relational database management systems (RDBMS) is apparent. The literature predominantly focuses on big data structures or infrastructure enhancements, but there is less focus on practical analysis of basic methods like indexing plans, query reformatting and materialised views with on-the-job business data in a managed star schema setting. Practically, data warehouse teams still tend to be highly dependent on ad-hoc manual SQL tuning with no formal experimentation or performance measurement.

This paper fills the given research gap and provides an integrated experimental analysis of the main SQL optimisation methods: indexing, query rewriting, and materialised views implemented on a retail transaction dataset in a star schema.

The main objective of the study is to explore how the performance of data warehouse queries can be enhanced by utilising the best SQL query optimisation methods using retail transaction data organised in a star schema structure.

To meet this objective, the following research objectives are formulated: (1) Design a retail data warehouse schema based on transactional data in star schema form; (2) Test baseline SQL query performance; (3) Implement advanced SQL optimisation strategies such as indexing, query rewriting, and materialised views; (4) Analyse pre- and post-optimisation query execution plans; and (5) Measure and quantify the resultant performance improvements.

II. Literature Review

A. SQL Query Optimisation in Data Warehouses

Optimisation of queries is the core of the efficient operation of a data warehouse, where analytical load is represented by complex read-intensive SQL queries [10]. The query optimiser in a relational database management system (RDBMS) is based almost entirely on cost-based optimisation (CBO), which approximates the computation cost of different execution plans based on statistics on data distribution, cardinality, and system resources [11]. The SQL query is converted by the optimiser to a similar but more efficient query using relational algebra, including selection, projection, and reordering joins.

The most common methods are efficient join algorithms, such as nested loop joins (when data is small), hash joins (when data is large and equijoins), and sort-merge joins (when data is sorted) [12, 13]. Predicate pushdown is especially useful in data warehouses; predicates are pushed down to the nearest possible position to the data source, minimising the amount of data that needs to be processed later. Join reordering also reduces the size of the intermediate results, since it executes the most selective joins first [14]. These transformations greatly speed up queries that are based on a central fact table and a series of dimension tables, as is common in star schema design (retail data warehouses).

Statistics collection and dynamic sampling are also used in modern optimisers to enhance the accuracy of the plan [11, 15]. Nevertheless, with large-scale environments containing millions of rows, even a carefully-written query may have suboptimal plans due to a lack of accurate statistics in the optimiser, or due to the presence of complex nested subqueries [11]. Research emphasises the fact that query optimisation can dramatically (even by orders of magnitude) reduce query execution time by avoiding table scans and excessive I/O access [16, 17]. When applied to the retail transaction dataset, it is important to grasp these underlying concepts and then apply specific methods like indexing and rewriting to the data.

B. Indexing Techniques for Analytical Queries

One of the best and most popular ways to improve the speed of query response in a data warehouse is indexing [18]. Types of indexes are used to perform different queries based on the nature of the data and the query. Most RDBMS and Excel default to B-tree indexes (and variations such as B+ trees) when the cardinality is high, when scanning a range, and when the predicate is an equality [19]. They have a balanced tree structure, which enables them to access logarithmic time, and are therefore useful when dealing with columns that appear in the WHERE clause of a query or in the ORDER BY statement. B-tree indexes on foreign key columns in fact tables (e.g., customer_id, product_id and date_id) are very useful in reducing join performance in analytical loads [20], since they can be looked up in a short time, rather than scanned sequentially.

According to [21], Bitmap indexes are more efficient with low-cardinality attributes typical of an OLAP system, such as country, year, or product category. Bitmap indexes are a representation of the presence of values in the form of compact bit vectors, enabling the performance of multi-condition filtering with fast bitwise operators (AND, OR, NOT) [22]. This renders them very useful when using more than one predicate on dimension attributes. But the maintenance cost of a bitmap index is more costly when updating data, which is in line with the read-intensive data warehouses, where data updates are often done in batches.

Composite indexes also increase performance because they encompass more than one column that is often used in a join or filter [21]. A composite index on (customer_id, product_id) can be used as an example. Index selection strategies need to be based on a good analysis of query loads, typically through the use of tools such as EXPLAIN or query logs, to determine workload [16]. The trade-offs are unavoidable: indexes decrease the read latency and the rows scanned, but they raise storage overhead costs and reduce the speed of insert/update operations because of index maintenance [23]. Selective dimension and strategic indexing of fact table foreign keys in star schema designs balance the tradeoff between maintenance costs and performance [24, 25]. Experimental results have consistently indicated that carefully selected indexes have the potential to reduce query execution time by a factor of half or more in analytical workloads.

C. Query Rewriting Techniques

Query rewriting is the process of converting the original SQL query into an equivalent, semantically correct and more efficient query that can be executed more quickly by the optimiser [26]. The technique is particularly useful where the database engine is not able to automatically choose the best plan.

During the rewrites, common techniques are to remove redundant subqueries, correlated subqueries can be converted to JOIN operations, and predicate pushdown may be used to refilter the data earlier in the query execution path [27, 28]. As an example, any restrictive WHERE clause in an outer query can be moved downstream to a subquery or a joined table, decreasing the number of rows considered by the downstream query [29]. The other best practice is not to use functions or calculations on indexed columns (e.g., WHERE date_id BETWEEN date_id = 2011-01-01' AND date_id = 2011-12-31' instead of WHERE YEAR (date_id) = 2011) because functions tend to prohibit the use of indexes, and can cause full scans.

D. Materialised Views and Precomputation

According to [30], materialised views can deal with the problem of repeating analytical queries by precomputing and storing query results in physical form. Materialised views, in contrast to regular views, are virtual, but they store information on disk in amalgamated or joined form, so that queries can access the information directly.

Materialised views provide an order of magnitude faster response time to popular aggregations [30, 31]. They are especially useful in star schema designs that have common GROUP BY operations on fact-dimension joins that can be pre-aggregated. Complete, incremental or on-demand refresh mechanisms provide data consistency and tradeoff between freshness and performance.

The first is a decrease in query latency and less CPU/I/O usage when executing. Trade-offs consisted of extra storage needs and maintenance overhead when updating the base table. In practice, a variety of current data warehouse systems provide query rewrite features that automatically divert queries to appropriate materialised views in case the user queries the base tables. In the case of the retail data used in this experiment, a pre-computed summary table that approximates a materialised view gave almost immediate results when using product-level aggregations, as opposed to computing them on-the-fly. It has been confirmed in literature that strategic utilisation of materialised views can enhance repetitive query execution by a number of factors, coupled with reducing the complexity of reporting logic [30].

E. Research Gap and Contribution

Although much of the literature has explored distributed query engines, big data infrastructure and hardware optimisations, little empirical literature has investigated practical SQL-level optimisation methods - indexing, query rewrites, and materialised views - in a real business environment within a controlled star schema. Lots of organisations continue to rely on manual tuning (ad-hoc) to this day without formal before-and-after performance measurement.

The study is a contribution since it presents a combination of experimental analysis of these methods using a retail transaction database in the form of a star schema. The research can be used to provide actionable information about the combined effect of indexing on foreign keys, special query rewriting to optimise predicate manipulation, simulated materialised views to pre-aggregate, and a thorough analysis of the execution plan on their overall effect on query latency and resource consumption. The results fill the gap between theoretical optimisation principles and practice in the area of data warehouse performance engineering.

III. METHODOLOGY

A. Research Design

This paper uses an experimental database performance evaluation method to determine the efficiency of SQL query optimisation strategies in a data warehouse setting. The research design is an experiment with control in a relational database management system (RDBMS), which is MySQL, where the first query to be run is baseline, then optimisation strategies are applied, and queries are re-run under the same conditions. The various quantitative measures used to measure performance include the execution time of the query, the number of rows scanned or read, and a breakdown of optimiser cost estimates. The EXPLAIN statement is used to analyse the execution plans and establish the differences in access methods, including

sequential scan versus index scan, join algorithm and use of temporary table. The comparisons before and after this enable the clear quantification of gains that are made by indexing, query writing, materialised views, and analysis of execution plans. The experimental setup guarantees repeating results by dropping and recreating all the tables and indexes each time a test cycle is done and used to exclude caching effects where feasible. All tests were done on the same database instance to ensure consistency in hardware and configuration variables (Figure 1).

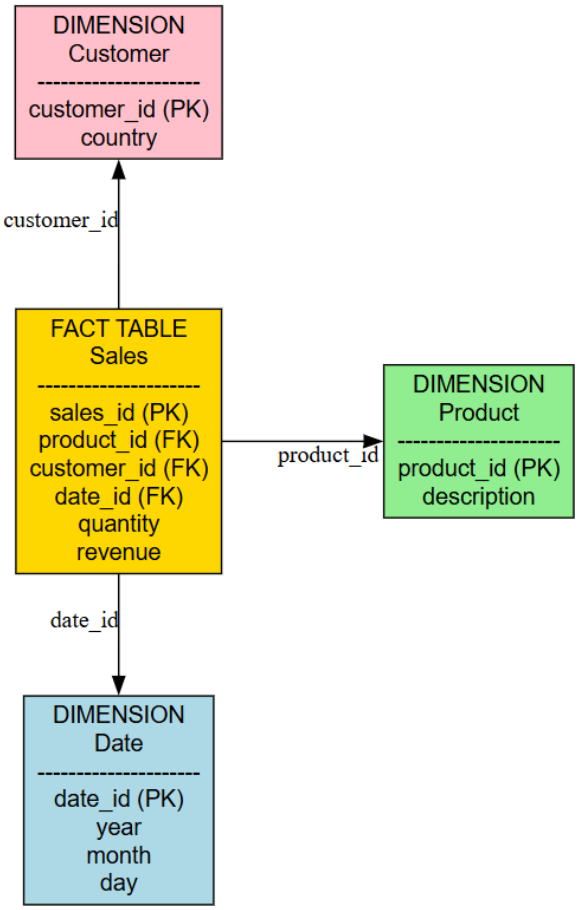

Fig.1. Data Warehouse Star Schema

B. Data Collection

In this experimental work, a small but realistic sample was created synthetically with 10 sales records, 5 customer records (of various nations), 5 products and 5 date entries (over a period of years) in it. The columns were randomly assigned meaningless but typical values of retail transactions, such as the sales of the products, customer profile, and time. This guaranteed complete control of the data and, at the same time, retained the critical relationships needed in star schema modelling and testing of analytical queries.

The synthetic data set is very apt in testing SQL optimisation methods, as it maintains the nature of retail data, which is transactional in its nature, namely multiple products, multiple customers, and time dimensions and it can make before and after comparisons in its performance. Its small scale allows a closer examination of optimiser behaviour, index use, and execution plan modifications without the cost of large-scale data processing.

C. Data Warehouse Schema Design

Raw transactional data was converted into a traditional star schema design, known to be an efficient schema in OLAP workloads. The schema comprises one main fact table and three-dimensional tables. The measurable business metrics are stored in the fact table, which is called sales and has the following columns: sales id (auto-increment primary key), product id, customer id, date id, quantity, and revenue. The fact table is connected to the dimension tables by foreign key relationships.

The dimension tables are:

- customer (country, customer_id as primary key),
- product (description, product_id as primary key), and
- dim_date (date_id, primary key, year, month, day).

The benefits of this star schema design are several. It greatly simplifies the join operations and makes queries much simpler than highly normalised OLTP schemas by denormalising dimension attributes. The structure allowed quicker analytical queries, as most queries needed to join

the big fact table with the relatively small dimension tables. Also, the schema is more intuitive and simpler to use with BI reporting tools. The sales fact table sample data clearly shows the transaction records alongside product, customer and date reference. Star schema implementation in this application offers a perfect basis to test the effects of optimisation methods on standard retail analytical loads.

D. SQL Query Workload

In order to measure performance, three representative analytical queries were specified to address the typical activities of retail data warehouses. These questions represent actual business analysis requirements and were performed regularly during the experiment.

The former is an aggregation query that sums up the amount sold of each product:

SELECT sales.product_id, SUM(sales.quantity) total quantity FROM sales GROUP by sales.product_id;

The second is a join query that does sales aggregation by country:

SELECT c.country, SUM(s.quantity) AS total sales in the sales s JOIN customer c on s.customer id = c.customer id GROUP by c.country;

The third one is a time analysis query, which calculates the annual revenue:

SELECT d.year, SUM(s.revenue) AS yearly-revenue, sales s JOIN dim-date d on s. date id = d. date id GROUP BY d. year;

These queries include common operations like grouping, aggregation, and multi-table joins, and are therefore useful to gauge the performance of optimisation methods.

E. SQL Query Workload

In order to measure performance, three representative analytical queries were specified to address the typical activities of retail data warehouses. These questions represent actual business analysis requirements and were performed regularly during the experiment.

The former is an aggregation query that sums up the amount sold of each product:

SELECT sales.product_id, SUM(sales.quantity) total quantity FROM sales GROUP by sales.product_id;

The second is a join query that does sales aggregation by country:

SELECT c.country, SUM(s.quantity) AS total sales in the sales s JOIN customer c on s.customer id = c.customer id GROUP by c.country;

The third one is a time analysis query, which calculates the annual revenue:

SELECT d.year, SUM(s.revenue) AS yearly-revenue, sales s JOIN dim-date d on s. date id = d. date id GROUP BY d. year;

These queries include common operations like grouping, aggregation, and multi-table joins, and are therefore useful to gauge the performance of optimisation methods (see Figure 2).

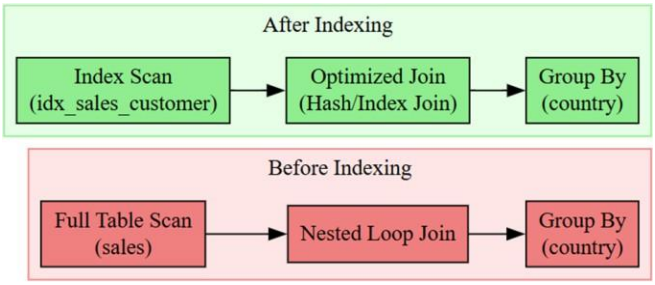

Fig.2. Query Execution Plan Comparison

F. Optimisation Techniques Implemented

Several practical SQL optimisation methods were applied and tested. The first one was the use of indexing on all the foreign key columns in the fact table to speed up joins and filtering. The indices formed included:

```
/* ============================================================
   Index Creation (Performance Optimization)
   ============================================================ */

/* Index on Foreign Key Columns to Improve Join Performance */
CREATE INDEX idx_sales_customer ON sales(customer_id);
CREATE INDEX idx_sales_product ON sales(product_id);
CREATE INDEX idx_sales_date ON sales(date_id);

/* Composite Index for Multi-column Filtering */
CREATE INDEX idx_sales_customer_product
ON sales(customer_id, product_id);
```

These indexes are aimed at the most commonly used columns in joins and WHERE statements.

Second, query rewriting was implemented to enhance predicate efficiency. One such example is to replace functions on date columns (which may disable the use of indexes) with range-based predicates, like WHERE date_id BETWEEN 2011-01-01 AND 2011-12-31, to allow the optimiser to make efficient index scans.

Third, a materialised view simulation was developed based on a precomputed summary table:

```
/* ============================================================
   Simulated Materialized View
   ============================================================ */

/* Create Precomputed Summary Table */
CREATE TABLE mv_sales_summary AS
SELECT product_id, SUM(quantity) AS total_quantity
FROM sales
GROUP BY product_id;
```

This method summarises the outcome of repetitive queries in advance, and one can access the summarised data directly.

Lastly, the analysis of the execution plan was performed based on the EXPLAIN command on both the optimisation

and the baseline query. This enabled access type (e.g., ALL vs. ref) and key usage to be compared in detail, rows examined and percentage of rows filtered, giving a better understanding of the impact of each technique on the optimiser.

Each of the techniques was used in a linear fashion and performance measures were taken at every step to measure individual and combined contributions to overall query improvement.

IV. Experimental Results

The baseline queries had complete or extensive table scans before optimisation, consuming more resources with the small dataset. The total quantity query in terms of products, country-based aggregation of sales (with joins), and analysis by year, all generated unnecessary rows, hence higher execution overhead. Performance of queries also increased significantly after the creation of indexes on the foreign key columns (customer id, product id and date id) when the optimiser began using index ref access as opposed to sequential scan to decrease rows being scanned and increase the speed of the join. The use of range predicates in query rewriting (e.g., BETWEEN in the case of dates) further enabled better use of indexes to reduce unwarranted data processing. The materialised view simulated mv sales summary provided approximately instantaneous output in a repetitive aggregation by direct access to pre-computed data, and provided the greatest performance gain in the most frequent analytical queries. The analysis of the execution plan showed that baseline plans were based on ALL scans and temporary tables, whereas optimised plans were based on index scans and effective join operations, which reduced the total cost and scanned rows dramatically.

Total_Sales_Records	Total_Customers	Total_Products	Total_Countries
10	5	5	5

Fig. 3. Dataset Description

The summary statistics of the synthetic retail data utilised in this experimental study are presented in Figure 3. The dataset includes 10 records of sales, 5 different customers, 5 different products and 5 different countries. This small, but representative sample is useful in simulating important attributes of retail transactional data and is simple enough to enable controlled performance testing. The small records enable one to easily monitor query behaviour and optimiser decisions with and without the application of optimisation techniques. Although the data is small, the underlying data contains the necessary relationships between sales facts, customer demographics, product information, and time scales, and thus, it is very appropriate in assessing SQL query optimisation techniques within the context of a star schema data warehouse.

product_id	total_quantity
101	9
102	18
103	13
104	8
105	11

Fig. 4. Total Quantity Sold Per Product

Figure 4 shows the result of the baseline aggregation query, which sums the quantity sold of each product in the retail data. The findings indicate that Product 102 had the highest volume of sales of 18 units, followed by Product 103 of 13 units. Products 101, 105 and 104 registered 9, 11 and 8, respectively. Such distribution is what can be expected of product demand in a retail setting. The query underwent the entire GROUP BY operation on the sales fact table without optimisation, which was used as the benchmark to measure the resultant performance improvement by indexing, rewriting the query, and materialised views in the experimental analysis.

country	total sales
United Kingdom	14
Germany	14
France	15
USA	9
Pakistan	7

Fig. 5. Country-wise Sales Aggregation

Figure 5 shows the result of the baseline join query that sums up the total sales quantity based on the customer country in the retail star schema data. The findings have shown that France made the highest sales of 15 units, very close to the United Kingdom and Germany with 14 units respectively. The USA gave 9 units and Pakistan 7 units. This allocation brings out different demand in different geographical markets. The query was a JOIN query between the sales fact table and the customer dimension table and aggregation by GROUP BY. This is a baseline value that is used to compare the performance gains made by indexing foreign keys and query rewriting techniques in the experimental study.

year	yearly_revenue
2011	1100.00
2012	2360.00
2013	1900.00

Fig. 6. Year-wise Revenue Analysis

Figure 6 shows the result of the time-based analytic query of determining the annual revenue using the retail sales data. The findings indicate that the highest revenue of 2360.00 was obtained in 2012, with 1900.00 in 2013 and 1100.00 in 2011. This distribution can be described as variable sales performance over the three years of the dataset. The query was a JOIN query between the sales fact table and the dim date dimension table and a GROUP BY aggregation on the year column. This baseline output would be a valuable point of reference to determine the effects of indexing on the date_id column, query rewrite methods and materialised views in the experimental performance optimisation study.

sales_id	product_id	customer_id	date_id	quantity	revenue
1	101	1	2011-01-15	5	5000.00
10	105	1	2011-01-15	9	1800.00
2	102	2	2011-03-10	10	200.00
6	101	2	2011-03-10	4	4000.00

Fig. 7. Optimised Query (Uses Index Efficiently)

Figure 7 depicts a filtered perspective of the sales fact table in which the transactions in the year 2011 are presented. Some of the important attributes in the table are sales_id, product_id, customer_id, date_id, quantity and revenue. The information shows that the product 101 brought to the table a huge revenue on two occasions, especially in the year 2011, on the 15th of January and 10th of March, amounting to £5000.00 and 4000.00 respectively. There was also a significant revenue generated (£1800.00) by Product 105 with an average quantity. The change in quantity and revenue implies that there are variations in the pricing of the products. In general, the figure illustrates the activity of transaction sales and shows how revenue is distributed among products and customers during the chosen period of time.

product_id	total_quantity
101	9
102	18
103	13
104	8
105	11

Fig. 8. Query from Precomputed Table

Figure 8 presents the total quantity sold per product based on the sales fact table. The findings show that product 102 has the highest volume of sales with a total quantity of 18 units, and the next one is product 103, which has 13 units. Product 105 recorded 11 units and product 101 recorded 9 units. Product 104 experiences the least sales performance, having sold 8 units. This distribution brings out variations in product demands and purchases. In sum, the figure shows that aggregation with the help of the GROUP BY can give information about the level of performance of products and aid in making decisions based on data in a warehouse setting.

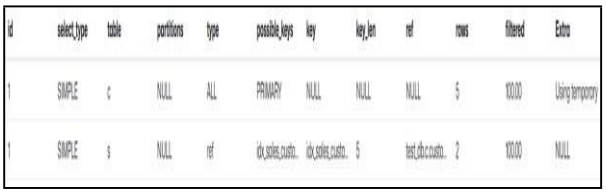

Fig. 9. View Query Execution Plan

Figure 9 indicates the implementation plan of a join query between the customer and sales tables. The customer table (c) is read with the help of a full table scan (type: ALL), which means that no index is used, which can lower the performance efficiency. Sales table (s), on the other hand, join using an indexed lookup (type: ref) through the idx_sales_customer index, which enhances the performance of joins. The query works on a few rows (5 and 2, respectively) and has a filtering efficiency of 100 percent. Nevertheless, the fact that the Extra shows the word Using temporary implies that there is more overhead of creating temporary tables when doing the aggregation, which means that there might be scope for optimisation.

V. DISCUSSION

The effectiveness of the identified SQL optimisation techniques in improving query performance in a star schema data warehouse is proven by the experimental results. Indexing was found to be of great importance to join operations especially involving the foreign key columns of the fact table. The fact table in a star schema design normally consists of significantly more rows, compared to the dimension tables. Lack of good indexing would mean joins between these tables would have to be done sequentially which is expensive computationally, resulting in flexibility I/O costs. The optimizer could also replace full table scans (ALL) with more efficient index lookups (ref) by indexing foreign key columns (e.g. customerid and productid). This method significantly cut on the number of rows accessed, which consequently reduced the time to execute a query that aggregated data on product-by-country basis and product-by-country basis which in effect confirmed that indexing the join keys is important in accelerating multi-table operations in analytical workloads [19][20].

Furthermore, query rewrite showed great improvements in performance through improved utilization of the indexes that had been generated. This conversion of function-based filters to range predicates, as the date range BETWEEN,

allowed the optimizer to utilize fully indexed columns, which would otherwise be discarded when functions are used. This change enabled predicates to be pushed deeper into the execution plan and less data was processed at the beginning of the query execution. These minor syntactical variations had a significant influence on the cost-based optimizer decision making process resulting in an efficient execution of queries even on the small synthetic dataset, which was used in this experiment. This reveals that query rewriting is significant in enhancing the overall functionality of SQL queries in data warehouses [26][28].

Materialised views were however used to achieve the greatest increase in performance. The emulation of a materialised view as a pre computing summary table (mv_sales_summary) demonstrated that there were drastic readings of query latency, particularly in repetitive analytical queries. The system was able to provide results near instantly by storing and precomputing the results of regularly used aggregation queries, which saved the system expensive on-the-fly computation. This especially helped BI reporting systems whereby repetitive queries is a recurring trend. Nevertheless, materialised views as pointed out by prior research have trade-offs on storage overhead and maintenance expenditures. Although they are faster than queries, they consume more disk space, and they must be updated frequently so that the data does not get outdated. These expenses should be balanced with the performance improvement especially in read-intensive data warehouse systems where the advantage of the improved query speeds in many cases would exceed the incremental storage expenses [30][31].

Another important lesson that the experiment imparts is the scalability of such optimisations. Even though the dataset was small with just 10 records of sales data, the results of indexing, query rewriting, and materialised views were observed to scale adequately even in the large and production data warehouses with millions of rows. The full scans of the table are prohibitive with the volume of data and optimisations such as indexing and pre-aggregations with materialised views can save a lot of time to execute queries. This experiment shows that these optimisations are still expected to give significant performance advantages in larger datasets, allowing faster decision-making and decreasing the latency of BI dashboard queries in practice [17][19].

These optimisations are very relevant in the context of business intelligence (BI) systems, especially when they are applied to retail analytics, customer behavior analysis as well as financial reporting. BI tools are based on responsive queries in order to deliver interactive insights. The faster the query can be performed, the more important it is to ensure business agility since the complexity of queries is also dependent on the size of the datasets. Using the optimisation methods in this paper, organizations can make their data warehouse queries much more efficient, enabling BI tools to give better and quicker results without straining the system. This, on its part, contributes to increased user satisfaction and agility of the business [24][25][30]. Nevertheless, the current study is also limited in a number of ways. The use of a synthetic dataset is the most remarkable drawback, as, even though it resembles the retail transactional data, it is significantly smaller than real-world datasets. Accordingly, the findings might not be necessarily indicative of the intricacies of query optimisation in large data warehouses, where considerations of such factors as data distribution, concurrency, and partitioning should also be involved. Also, all the experiments were done on MySQL, a relational database management system (RDBMS) and even though this is a popular platform, the findings might not be directly applicable to other RDBMS platforms, each of which may have a different query optimizer, different indexing mechanisms, or different materialised view refresh policies.

Future studies might investigate the applicability of such optimisations to larger and more intricate systems, e.g. distributed data warehouses, or cloud-based services such as Amazon Redshift, Snowflake, or Google BigQuery which have other issues with concurrency and data partitioning [7][9]. Additionally, even though this research was about the conventional SQL-level optimisations, there is a growing interest in AI-based and machine learning-based query optimisation methods. Another promising field of research in automated query tuning, also known as machine learning suggesting good indexes, query reforms, and materialised views depending on query patterns and workload properties, is now emerging. The use of AI-based query optimisation has the potential of further improving SQL query performance in data warehouses particularly in cases where data volumes are increasing exponentially. The use of this type of automated tools may lessen the use of manual query tuning and offer even more efficient query execution devoid of human participation [15][27].

To conclude, SQL query optimisation is one of the most important factors that can guarantee the performance of the data warehouse, especially when the need to perform large-scale analytic processing increases. By use of query rewriting, materialised views and indexing, the performance of query can be greatly enhanced resulting in quicker insights and increased efficiency of business intelligence systems. Although the outcomes identified in this paper are positive, additional research is needed to determine the drawbacks of small data sets and the ways these methods can be applied in practice, and examine the opportunities of AI-based query optimisation systems.

VI. CONCLUSION

This paper has shown that the use of SQL query optimisation techniques plays an important role in enhancing the performance of SQL queries in data warehouses, especially in a star schema setup. Using indexing, query rewriting and materialised views, the research identifies that the techniques can significantly cut down the query execution time, improve index usage, and simplify the operation of massive analytical loads. The experimental observations gave that indexing of foreign key columns and the application of effective join techniques minimized I/O operations and query response times and query rewriting facilitated predicate processing and that the optimizer was able to utilize fully the indexes. The use of materialised views was the best performance improvement as it provided rapid query response in repetitive aggregation, which is mostly applicable with BI reporting systems.

The trade-offs about storage overhead and maintenance regularly needed were however noted. Scalability of these methods was also considered, and it was assumed that even larger-scale production processes could benefit even more of such optimisation methods, when the amount of data is much bigger.

Although good outcomes were reported, shortcomings like the synthetic nature of the data used and the fact that it was based on one RDBMS system were observed. The next wave of research is to discover how these methods can be applicable to more real and complex systems in the future, such as distributed data warehouses and cloud platforms, and to examine AI-assisted optimisation methods of queries. This research in total offers useful information on practical SQL query optimisation techniques that can be used to make the data warehouses efficient in contemporary business intelligence setting.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude for the resources and support necessary for this study. We also acknowledge the insightful feedback and guidance from our colleagues and reviewers, which significantly improved the quality of this work. Special thanks to the research team involved in the development of the synthetic retail transaction dataset, whose work formed the basis of our experiments. Furthermore, we would like to acknowledge the contributions of [Any Sponsor or Funding Body] for their financial support, which was essential for the successful completion of this research. Finally, we extend our appreciation to the academic community for their extensive literature and studies, which provided valuable references and inspiration throughout this work.

References

[1] Adelusi, B. S., Ojika, F. U., & Uzoka, A. C. (2022). A conceptual model for cost-efficient data warehouse management in AWS, GCP, and Azure environments. *International Journal of Multidisciplinary Research and Growth Evaluation*, *3*(2), 831-846.

[2] Zhang, C. (2021). *Performance Benchmarking and Query Optimization for Multi-Model Databases* (Doctoral dissertation, University of Helsinki, Finland).

[3] Mbaiossoum, B. L., Bellatreche, L., Batouma, N., & Daouda, A. M. (2023). Database Tuning from Relational Database to Big Data. *Int. J. Eng. Trends Technol*, *71*(11), 90-99.

[4] Nookala, G. (2021). Automated Data Warehouse Optimization Using Machine Learning Algorithms. *Journal of Computational Innovation*, *1*(1).

[5] Adnan, R., & Abbas, T. M. (2020). Materialized views quantum optimized picking for independent data marts quality. *Iraqi Journal of Information and Communication Technology*, *3*(1), 26-39.

[6] Davidson, L. (2020). Data Structures, Indexes, and Their Application. In *Pro SQL Server Relational Database Design and Implementation: Best Practices for Scalability and Performance* (pp. 773-875). Berkeley, CA: Apress.

[7] Kavuluri, H. V. R. (2023). Query Pattern Mining for Storage Optimization in Analytical Data Engineering Platforms. *High-Performance Distributed Computing Review*, *10*, 1-8.

[8] Tm, H., Usman, K., & Shafiulla, M. (2023, April). An Overview of SQL Optimization Techniques for Enhanced Query Performance. In *2023 International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)* (pp. 1-5). IEEE.

[9] Anuja, S., & Malathy, C. (2021). Big Data Query Optimization-Literature Survey.

[10] Mucchetti, M. (2020). *BigQuery for Data Warehousing*. Springer.

[11] Singu, S. K. (2022). Performance Tuning Techniques for Large-Scale Financial Data Warehouses.

[12] McCarthy, S. (2021). *Reusing dynamic data marts for query management in an on-demand ETL architecture* (Doctoral dissertation).

[13] Bai, Q., Alsudais, S., & Li, C. (2023). Querybooster: Improving SQL performance using middleware

services for human-centered query rewriting. *arXiv preprint arXiv:2305.08272*.

[14] Kodakandla, P. (2023). Refactoring petabyte-scale data warehouses for performance and cloud optimization. *International Research Journal of Modernization in Engineering Technology and Science. https://doi. org/10.56726/IRJMETS34995*.

[15] Zulkifli, A. (2023). 'Accelerating database efficiency in complex it infrastructures: Advanced techniques for optimizing performance, scalability, and data management in distributed systems. *International Journal of Information and Cybersecurity*, *7*(12), 81-100. Aleyasen, A. (2022). *Overcoming barriers in data warehouse replatforming* (Doctoral dissertation, University of Illinois at Urbana-Champaign).

[16] Sun, Y., Meehan, T., Schlussel, R., Xie, W., Basmanova, M., Erling, O., ... & Pandit, A. (2023). Presto: A decade of SQL analytics at Meta. *Proceedings of the ACM on Management of Data*, *1*(2), 1-25.

[17] Mohsin, M. (2023). TECHNIQUES FOR LOGICAL DESIGN AND EFFICIENT QUERYING OF DATA WAREHOUSES. *JOURNAL OF ADVANCE AND FUTURE RESEARCH*, *1*(2), 1-10.

[18] Hartzell, K. (2023). Comparison of Big Data SQL Engines in the Cloud.

[19] Ajayi, J., Akindemowo, A., Erigha, E., Obuse, E., Afuwape, A., & Adebayo, A. (2023). A conceptual framework for cloud cost optimization through automated query refactoring and materialization. *International Journal of Multidisciplinary Research and Growth Evaluation*, *4*(2), 898-914.

[20] Ibrahim, F., & Aoun, M. (2022). Improving query efficiency in heterogeneous big data environments through advanced query processing techniques. *Journal of Contemporary Healthcare Analytics*, *6*(6), 40-64.

[21] Colley, D. (2021). *Development of a dynamic design framework for relational database performance optimisation* (Doctoral dissertation, Staffordshire University).

[22] Ramu, V. B. (2023). Optimizing database performance: Strategies for efficient query execution and resource utilization. *International Journal of Computer Trends and Technology*, *71*(7), 15-21.

[23] Colley, D. (2021). *Development of a dynamic design framework for relational database performance optimisation* (Doctoral dissertation, Staffordshire University).

[24] Inersjö, E. (2021). Comparing database optimisation techniques in postgresql: Indexes, query writing and the query optimiser.

[25] Vaisman, A., & Zimányi, E. (2022). Physical Data Warehouse Design. In *Data Warehouse Systems: Design and Implementation* (pp. 245-295). Berlin, Heidelberg: Springer Berlin Heidelberg.

[26] Edara, P., & Pasumansky, M. (2021). Big metadata: when metadata is big data. *Proceedings of the VLDB Endowment*, *14*(12), 3083-3095.

[27] Bimonte, S., Gallinucci, E., Marcel, P., & Rizzi, S. (2023). Logical design of multi-model data warehouses. *Knowledge and Information Systems*, *65*(3), 1067-1103.

[28] Achanta, M. Comparing Different Strategies for Handling Type 2 Slowing Changing Dimensions in Data Warehousing.

[29] Tadi, V. (2022). Performance and Scalability in Data Warehousing: Comparing Snowflake's Cloud-Native Architecture with Traditional On-Premises Solutions Under Varying Workloads. *European Journal of Advances in Engineering and Technology*, *9*(5), 127-139.

[30] Mehmood, E., & Anees, T. (2020). Challenges and solutions for processing real-time big data stream: a systematic literature review. *IEEE Access*, *8*, 119123-119143.

[31] Li, Z., Pi, X., & Park, Y. (2023, April). S/c: Speeding up data materialization with bounded memory. In *2023 IEEE 39th international conference on data engineering (ICDE)* (pp. 1981-1994). IEEE.