DW vs OLTP Performance Optimization in the Cloud on PostgreSQL (A Case Study)

Dakota Joiner
Computer Science
Okanagan College
Kelowna, Canada
0000-0002-3094-0015

Mathias Clement
Computer Science
Okanagan College
Kelowna, Canada
0000-0001-8206-307X

Shek (Tom) Chan
Mathematics and Statistics
Langara College
Vancouver, Canada
0000-0001-6143-7175

Keegan Pereira
Computer Science
Okanagan College
Kelowna, Canada
0000-0002-2893-3406

Albert Wong
Mathematics and Statistics
Langara College
Vancouver, Canada
0000-0002-0669-4352

Youry Khmelevsky
Computer Science
Okanagan College
Kelowna, Canada
0000-0002-6837-3490

Joe Mahony
Research and Development
Harris SmartWorks
Ottawa, Canada
JMahony@harriscomputer.com

Michael Ferri
Research and Development
Harris SmartWorks
Ottawa, Canada
mferri@harriscomputer.com

***Abstract*—This case study shows the performance issues and solutions for a data warehouse (DW) performing well to serve industrial partners in improving customer data retrieval performance. An online transaction processing (OLTP) relational database and a DW were deployed in PostgreSQL and tested against each other. Several test cases were carried out with the DW, including indexing and creating pre-aggregated tables, all guided by in-depth analysis of EXPLAIN plans. Queries and DW design were continually improved throughout testing to ensure that the OLTP and DW were compared equally. Seven queries (requested by the industrial client) were used to thoroughly test different performance aspects concerning client feedback and the complexity of requests for all areas the DW might cover. On average, the data warehouse showed a one to three magnitudes increase in query execution performance, with the highest calibre results coming in at 2,493 times faster than the OLTP. All test cases showed an increase in performance over the OLTP. Additionally, the data contained in the DW took up 24% less storage space than**

is frequently utilized in customer queries [2], [3]. Other approaches include parallel processing across an integrated cluster of computers [4]. Clearly, there exist many different solutions to increase effciency. Developing the most effective solution is therefore a matter of testing on specifc systems to determine what meets the outlined needs for a given client.

When running an SQL query, there is often a piece of software called the Query Optimizer that chooses the most effcient method for data retrieval. Understanding how the optimizer chooses to access specifc data can inform database design decisions [5]. The EXPLAIN PLAN is a tool that displays the choices made by the Query Optimizer for a given SQL statement with specifc reference to information such as where loops are carried out, how many rows are accessed, how many results are saved, and how many results are thrown away, etc. [6]. Utilizing and understanding an EXPLAIN PLAN can help in pinpointing ineffciencies and identifying problem areas that can be fne-tuned to reduce number of joins, number of row accesses, and number of results thrown away. The EXPLAIN PLAN alone cannot provide the necessary information to validate the effciency of a design schema or query, but rather is a powerful tool to be included as part of a testing suite.

As part of an attempt to facilitate speed of retrieval, many DBMSs (including PostgreSQL, the DBMS used in this case study) will cache results in RAM for easy retrieval at a later time [5]. While useful for practical applications, caching carried out in a testing environment can lead to biased performance results and reporting of retrieval time. Consider a situation where a test query is run ten times consecutively. After the frst run, the DBMS may cache results and the following nine tests will show a considerably shorter retrieval time. One might believe data retrieval to be quite fast when looking at the average time per execution. While it might seem reasonable at frst glance and a DB designer would react positively to a signifcant decrease in query run time, they must

shown in Table III and Figure 2. One might worry about the additional space requirements to include several extra tables derived from an already large fact table, but these additions still result in storage savings compared to the OLTP database while providing signifcant reduction in query run time. The pre-aggregated tables are considerably smaller (total 11 GB), bringing the total storage size of the data warehouse to 100 GB which is less than that of the original OLTP database.

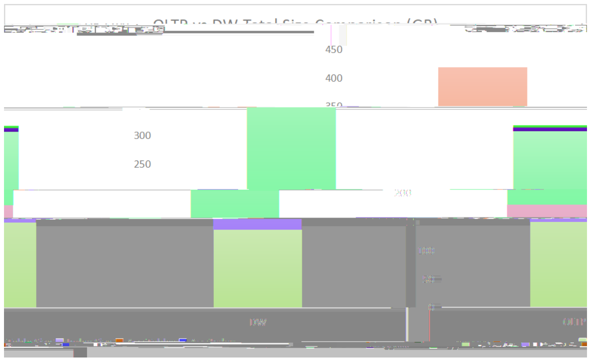

Fig. 1. Storage comparison of OLTP and star schema. OLTP: table size 179 GB, index size 239 GB. DW: table size 137 GB, index size 170 GB, pre-aggregated tables 11 GB.

The query run times are signifcantly shorter in the pre-aggregated tables compared to the OLTP and non-aggregated table. The difference in query run time can likely be attributed to the smaller size of each table. Pre-aggregating the results cuts down computational time signifcantly as results are only retrieved and do not have to be aggregated at query run time each time the query is run.

A. Testing Framework

To faciliate testing, a dedicated virtual machine running Alma Linux version 8.5 and PostgreSQL version 10.17 was created. A test database containing real customer data, which will remain obfuscated, was loaded into the database. The virtual machine had 64 GB of dedicated RAM and 1.6 TB of storage total. An Extract/Load/Transform (ELT) process was carried out to extract the data from the provided OLTP database to fll out the data warehouse. In the case of the pre-aggregated tables, functions were designed to aggregate data based on specifc conditions, those being daily or monthly, location class, and meter type, as in Listing 1.

```
SELECT read_month, SUM(total_usage)
FROM star1.monthly_usage
WHERE to_date(read_month, 'yyyy-mm') >=
        to_date(x, 'yyyy-mm')
   AND to_date(read_month, 'yyyy-mm') <
        to_date(y, 'yyyy-mm')
...
```

Listing 1. An extracted snippet of the ELT process for a monthly data aggregation

To carry out the testing and comparisons, automated shell scripts were written to ensure repeatability and serializability. The general outline of the script, in order of operation, is to clear the database cache, begin a timing function, generate an EXPLAIN plan, carry out a specifed number of timed tests, clear the cache between each query, fnish timing the total program run time, and output the results to a fless function, generate an EXP
out a5ut

TABLE III

A

Fig. 2. Run time comparisons of testing queries on the OLTP and DW on a logarithmic scale. The letter designations correspond to different pre-aggregation type for each query. a - monthly, b - daily, c - location class, d - meter type. The performance improvements of the pre-aggregated data warehouse are plainly seen here.

```
    =8  width=15)  (actual  time=6.197..140.461
    rows=13  loops=3)
```

Listing 3. Snippets of the EXPLAIN plan for query 2 on the OLTP (top) and query 2 on the monthly pre-aggregated table (bottom).

V. Discussion

Performance optimizations in databases have been heavily studied, though there exists little published research speci cally testing the comparison of data warehouses and OLTP databases in the cloud. Data warehouses are often used for business decision making, but the direction taken here allows for transaction-like retrieval of aggregated business data at a level unachievable with an OLTP. The pre-aggregation of tables in a data warehousing setting speci cally designed to enhance customer experience has resulted in a signicant reduction in query time for data retrieval. In addition to pre-aggregating relevant time-based data, other tables were experimentally pre-aggregated for further testing. Additional tables were created aggregating by location class (an operational classi cation) and separately by meter type. Compilation of results in this way shows further promising results in reducing query execution time.

Future target areas to further improve ef ciency include the introduction of bitmap indexes, partitioning, and parallelization. The idea is to integrate all ideas together into the most effective version of the system. Firstly, bitmap indexing can further increase the speed of retrieval by mapping indexes in a bit map to a similar group of rows instead of by single rows. Congruently, parallelizing the execution of queries and employing bitmap indexes at the same time is likely to lead to further reductions in execution time. Parallel execution can break up large complex jobs into several smaller, simpler jobs and then aggregate the results of each of those to produce the nal desired results [35]. This situation seems ideal when, in the tested cases here, there are billions of records to access and aggregate. Finally, increased degrees of partitioning may also be useful.

VI. Future Work

Performance optimization will be further enhanced with the implementation of the aforementioned targets: partitioning, parallelization, and bitmap indexing with rigorous performance testing of both systems in the cloud. The automated scripting for retrieval of materialized views is considered as a customer-facing solution to decrease space usage as not every query needs to be represented as a pre-aggregated table. Once the data warehouse is in an acceptable state, it will be paired with the machine learning model under development as an integrated environment for queries and information retrieval using English commands.

VII. Conclusion

Testing of query execution time on an OLTP database versus a data warehouse was performed with PostgreSQL. Complex queries were designed to not only simulate "customer-like" data retrieval, but to put the data warehouse under strain and test the full capabilities in all expected use areas. Initial performance between the OLTP database and the DW was approximately at parity, but introduction of indexes and pre-aggregated tables resulted in a vast decrease in run time. The decision to create speci cally pre-aggregated tables was driven by analysis of detailed EXPLAIN plans and related performance requirements. All queries showed an increase in performance on the pre-aggregated tables in the range of one to three orders of magnitude faster than the parent OLTP database, with all producing results in sub ten-second

[22] Y. Khmelevsky, X. Li, and S. Madnick, “Software develop-