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 the OLTP. The results here indicate a promising direction to take business analytics with data warehousing, as customers will experience significant cost savings and a reduction in time to receive desired results from their data storage platforms in the cloud. The work in this case study is a continuation of previous work in a much larger project concerning integrating database technologies with machine learning to improve natural language processing solutions as a cost-saving measure for utilities consumers.**

***Index Terms*—Data warehousing, databases, performance optimization, PostgreSQL, big data**

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), Harris SmartWorks Division of Harris Computers, Okanagan College, and Langara College.

I. Introduction

Applications of effective data warehouse design and retrieval optimization are two heavily studied topic areas often driven by partnerships of research teams and industrial companies. This case study is one of such a partnership where the industrial partner sought an enhanced data retrieval method to facilitate a speed up in real-time access of its data by customers. The previous storage mechanism, a classical OLTP database, did not meet either customer or business expectations as the basis for a new product (a critical cloud deployment area). Thus, a data warehousing solution using both star schema and pre-aggregated tables was implemented to facilitate increases in performance over the OLTP database. To confirm the agreed upon retrieval performance increase, several tests will be conducted and a number of techniques were taken into consideration that include data warehouse design schema, pre-aggregation of data, EXPLAIN plans, cost-effective databases, and the management of massive data sets with the overall goal of improving the effectiveness of the data set.

Kimball and Ross [1], in their definitive work on data warehousing, suggest the use of star schema consisting of a central fact table linked to secondary dimensional tables through primary/foreign key relationships. The approach highlighted here is sufficient for smaller data warehouses but begins to fail in data retrieval as the different tables increases to massive sizes and when multiple joins must be performed in a single query or set of queries. One approach to applied industrial data warehousing solutions is to include dimensional tables of pre-aggregated data that

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 efficiency. Developing the most effective solution is therefore a matter of testing on specific 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 efficient method for data retrieval. Understanding how the optimizer chooses to access specific 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 specific 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 inefficiencies and identifying problem areas that can be fine-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 efficiency 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 first 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 first glance and a DB designer would react positively to a significant decrease in query run time, they must account for this caching effect. The first test is likely to represent a true run time rather than the average of all ten runs that includes the cached results. Removing the data cache in between each test run will ensure correctness of performance results in average query run time. Additionally, it is important to consider what kinds of data are cached. Many DBMSs will cache multiple sets of information including actual retrieved results, indexes, and even query execution plans [5].

Harris SmartWorks, the industrial partner, works in the utilities industry and has implemented smart meters with its customers. A smart meter, in contrast to a classical meter, can collect many different kinds of data across multiple channels continually. Imagine the classic scenario where a person from the utilities company arrives to a property to record voltage, water usage, etc. once a month from an analog meter. The volume of data collected in that case would be rather small as it is obtained only once a month. Conversely, smart meters frequently collect data at sub-minute intervals across hundreds to thousands of locations, sometimes even on a single property for a single customer. The volume of data generated in this case becomes massive, so simple relational database structures lose effectiveness in tracking the data as the tables grow. Alternative strategies for managing and accessing this data become necessary over time as the business scales up.

This research is a continuation of a larger, ongoing Natural Sicence and Engineering Research Council (NSERC) funded project between Okanagan College, Langara College, and Harris SmartWorks on the development of an integrated natural language processing (NLP) suite with the ultimate goal of reducing utilities costs for Smartworks' customers [7]–[9]. The performance optimization sought here is one piece of a much larger puzzle that has seen continuous development for several years. One can refer to preparatory work that has been carried out in our previous research projects in [9]–[28] on machine learning, NLP, database performance optimization, and remote teamwork solutions.

II. Related Works

Discussion of the merits of a data warehouse (DW) over an OLTP database in the cloud must begin with the key differences in their functionality: how data is processed. Data warehouses focus on analytical processing of historical data to support business decision making whereas OLTP databases focus on transactional processing with many queries to be run frequently [29]–[31]. OLTP databases have strengths in multiple, concurrent transactions over the internet for often simple queries while maintaining data integrity. On the other hand, data warehouses are designed to aggregate large volumes of data for analytical processing with applications in business intelligence, machine learning, and data mining [32].

There are many existing design schema commonly used for data warehouse creation. Choosing a design schema varies from client to client and their particular needs. Thus, a solid understanding of what is available will strengthen the choices made. The star schema is the most well known design schema for data warehousing. In a DW, a fact table is linked to secondary dimensional tables through primary/foreign

key relationships [1]. Improvements and variations on this model come in the form of both snowflake schema and galaxy schema. In a snowflake schema model, a central fact table still exists, but the dimensional tables are slightly normalized and may have additional relationships of their own.

In a galaxy schema model, there are multiple fact tables that share the same dimensional tables. These designs are sufficient for many applications, however specialized design schema may be more applicable for specific business needs. The starnest schema by Garani et al. [33] is one such application where components of the star and snowflake schema are integrated to more accurately represent spatiotemporal data, a common type of data among applied industrial clients. In this application, hierarchical nesting of attributes allows for increased efficiency in retrieval of results as meaningfully proximate data are represented physically in the same tables. This also reduces the need for both the complexity and number of computationally expensive joins.

There has been considerable development in other areas to optimize data warehousing performance including partitioning, aggregation, indexing, parallelization, among others [34]–[36]. The use of multiple strategies concurrently is prevalent in many research areas, but few do testing between different database variations [37].

III. Data Warehouse Design

After consideration of existing database structure and study of applicable design schema, a star schema was chosen to best fit the client's needs. The cloud storage statistics of the OTLP schema and the star schema are shown in Table I. The data in these tables represents approximately five years worth of real, obfuscated customer data from a single customer with several thousand smart meters. It is usually expected that a data warehouse will be larger than its OLTP counterpart, however in this case, data not specifically required for analysis were removed in the ELT process. This resulted in a much smaller storage footprint overall for the data warehouse.

TABLE I
SIZE OF TABLES AND INDEXES ON THE ORIGINAL OLTP DATABASE AND DATA WAREHOUSE

	Table Size (GB)	Index Size (GB)	Total Size (GB)
OLTP DB	179	239	418
DW	137	170	307

To assess performance of the DW in comparison to the corresponding OLTP, seven testing queries were designed with three primary motivations from the DW design requirements:

1) the queries are based on solicited feedback from customers.
2) they are intended to be "heavy" queries, such that in theory it would be easier to see if there are any significant performance improvements.
3) they are intended to make use of the various indexes on the main data table in the OLTP, again to potentially highlight areas of significant improvement.

These queries were designed in reference to the OLTP, and thus, equivalent queries needed to be made following data warehouse construction to return the same information from the data warehouse. When considering a total solution, data warehouse design is not the only testing criteria. Queries were also optimized over time to improve retrieval time, both with the OLTP and the DW. It may seem odd to optimize the OLTP queries when the intention is to demonstrate the superiority of the DW, but it allows for an opportunity to refine the performance of the DW even further. If improvements are made to the OLTP queries, then mirrored improvements should be able to be achieved in the DW should it truly prove more efficient.

IV. Practical Measurements and Results

Initial baseline testing demonstrated approximately equal query run time between the OLTP and the DW. Baselines were carried out on both the OLTP and the DW with no optimizations. The results of this baseline are seen in Table II.

TABLE II
BASELINE QUERY RUN TIMES

Query	OLTP Run Time (s)	DW Run Time (s)
1	28.8	31.7
2	28.6	31.5
3	32.4	38.3
4	37.0	31.6
5	40.0	35.7
6	30.5	33.7
7	40.5	64.2

Following the baseline testing, it was identified that a typical customer does not need to drill down to the full specifics of quarter-hourly readings (the most prevalent data entry in the OLTP and DW); many are content with hourly, daily, or monthly data. In that vein, three tables were created with pre-aggregated data to store hourly, daily, and monthly usage. The space requirements in each the OLTP database and the DW with pre-aggregated tables are shown in Figure 1, while the time comparisons of testing criteria are

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 significant 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.

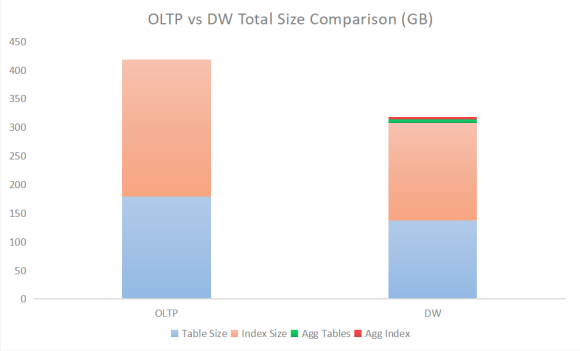


Fig. 1. Storage comparison of OLTP and star schema. OLTP: table size ≈ 179 GB, index size ≈ 239 GB. DW: table size ≈ 137 GB, index size ≈ 170 GB, pre-aggregated tables ≈ 11 GB.

The query run times are significantly 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 significantly 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 fill out the data warehouse. In the case of the pre-aggregated tables, functions were designed to aggregate data based on specific 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 specified number of timed tests, clear the cache between each query, finish timing the total program run time, and output the results to a file for future analysis. The user can specify the number of tests to carry out, as well as which query to test, as command-line arguments. A snippet of the script can be seen below in Listing 2 where the clearing of the cache for each test is highlighted.

```
...
# Stops the Database
# Clears OS caches
# Output of restart functionality discarded
# Starts the Database
sudo -iu postgres bash << EOF > /dev/null
    2>&1
    export PGDATA=/data/data
    pg_ctl stop
    sync
    sudo sh -c "echo 3 > /proc/sys/vm/
    drop_caches"
    pg_ctl start
EOF
...
```

Listing 2. Automated bash testing script code snippet for clearing of cache to equilibrate consistency of tests

B. EXPLAIN Plans

In PostgreSQL, the EXPLAIN function serves to estimate the cost of running a query and, with additional options, it can also be used to time a function. When calling the ANALYZE option on EXPLAIN in PostgreSQL, many more 'statistics' are included as part of the output [5]. Of particular interest to this research are the cost and run time associated with running a query. Cost is an arbitrary unit based on sequential page fetches, and thus only relative values matter for comparison [5]. It is also important to note that the 'statistics' provided by an EXPLAIN plan are based on a random sampling of the data, so they are likely to change between runs as only an estimate is returned. However, the addition of the ANALYZE option will cause the query to be run and actual results will be obtained. It is also important to run tests on full-sized tables; smaller dummy tables representative of a small portion of the data are unlikely to produce reliable and true results that actually pertain to the target database.

Analysis of EXPLAIN plans on each query motivated the idea to pre-aggregate the tables to improve

TABLE III
AVERAGE RUN TIMES OF ALL TESTING QUERIES. QUERY DESIGNATIONS: A - MONTHLY, B - DAILY, C - LOCATION CLASS, D - METER TYPE.

Query	OLTP (ms)	DW (ms)	OLTP/DW Ratio	Aggregation
1b	28,833	3,769	7.65	Daily
1c		13	2,217.92	Location Class
2a	58,569	135	211.62	Monthly
2b		135	211.62	Daily
2c		13	2,197.62	Location Class
3c	32,421	13	2,493.92	Location Class
4a	36,993	267	138.55	Monthly
4b		3,371	9.92	Daily
5c	40,044	101	396.48	Location Class
5d		199	201.23	Meter Type
6b	30,524	3,779	8.08	Daily
7a	40,546	593	68.37	Monthly
7b		6,139	6.60	Daily

results. Looking at the data in Table II, the OLTP database runs faster than the data warehouse at baseline level. However, once the pre-aggregated tables with indexes are built, the results are noticeably faster, as in Table III. This is true for both the table pre-aggregated to specifically collate monthly data, and the table that is pre-aggregated with daily data that is then further aggregated to output monthly results. Pre-aggregating results shows a significant reduction in query run time. The most significant results were retrieved 2,493 times faster with the pre-aggregated tables in the data warehouse over the traditional OLTP. The smallest increase in performance was observed to be approximately 6 times faster than that of the OLTP.

Listing 3 shows a snippet of the EXPLAIN plans from query 2 against the OLTP and the pre-aggregated monthly table. The query optimizer shows its chosen path for how best to execute a query once it has been submitted. That is, it will try to decide what methods it will carry out to execute the query as quickly as possible. In executing query 2 against the OLTP, there are 14 nested sets of grouping, sorting, merging, and scanning for an estimated cost of 19,641,940 units. In contrast, the monthly pre-aggregated query 2 shows just 4 nested levels for an estimated cost of 56,623. Though cost, being an arbitrarily chosen unit, is not necessarily a definitive indicator of efficiency, in this case the incredible relative difference alongside the fewer number of nested operations demonstrates the efficiency possible with the pre-aggregated tables in the DW.

```
# OLTP
Finalize GroupAggregate  (cost
    =19641240.68..19641940.14 rows=1821 width
    =64) (actual time=30672.914..30682.655
    rows=1 loops=1)
   Group Key: (to_char(rf.read_dt, 'YYYY-MM'
    ::text))
   ->  Gather Merge  (cost
    =19641240.68..19641885.51 rows=3642 width
    =64) (actual time=30672.902..30682.645
    rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial GroupAggregate  (cost
    =19640240.65..19640465.11 rows=1821 width
    =64) (actual time=30663.234..30663.237
    rows=1 loops=3)
               Group Key: (to_char(rf.read_dt
    , 'YYYY-MM'::text))
               ->  Sort  (cost
    =19640240.65..19640306.37 rows=26286
    width=37) (actual time
    =30654.375..30656.536 rows=16128 loops=3)
                     Sort Key: (to_char(rf.
    read_dt, 'YYYY-MM'::text))
                     Sort Method: quicksort
    Memory: 1130kB
                     ->  Hash Join  (cost
    =526237.03..19638311.00 rows=26286 width
    =37) (actual time=4963.753..30648.737
    rows=16128 loops=3)
... (plan continues)

# DW Monthly pre-aggregated
Finalize GroupAggregate  (cost
    =56620.99..56623.32 rows=16 width=40) (
    actual time=147.521..149.212 rows=1 loops
    =1)
   Group Key: read_month
   ->  Gather Merge  (cost=56620.99..56623.00
     rows=16 width=40) (actual time
    =147.506..149.197 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial GroupAggregate  (cost
    =55620.97..55621.13 rows=8 width=40) (
    actual time=140.668..140.668 rows=1 loops
    =3)
               Group Key: read_month
               ->  Sort  (cost
    =55620.97..55620.99 rows=8 width=15) (
    actual time=140.644..140.645 rows=13
    loops=3)
                     Sort Key: read_month
                     Sort Method: quicksort
    Memory: 26kB
                     ->  Parallel Seq Scan on
     monthly_usage  (cost=0.00..55620.85 rows
```

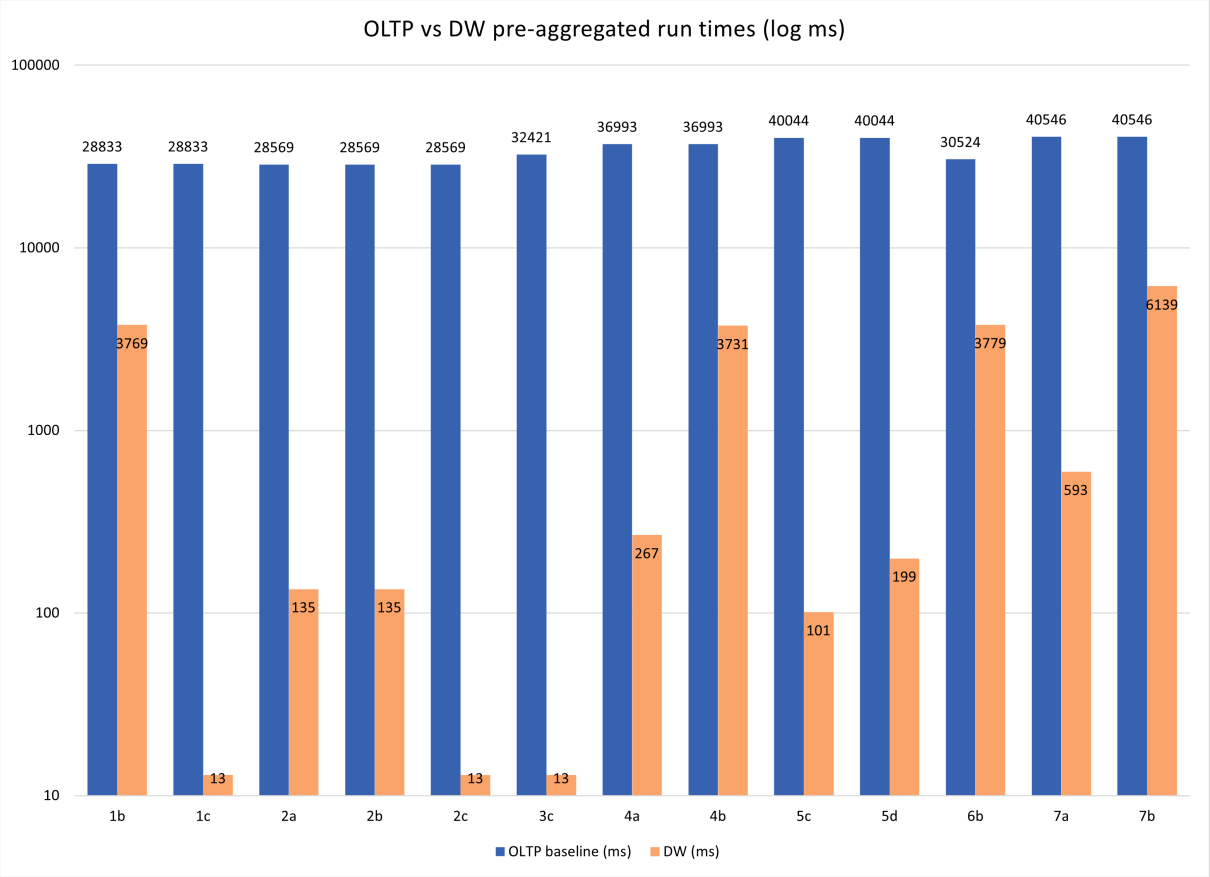


Fig. 2. Run time comparisons of testing queries on the OLTP and DW on a logarithmic scale. The letter designations correspond to different pre-aggregation type for each query. a - monthly, b - daily, c - location class, d - meter type. The performance improvements of the pre-aggregated data warehouse 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 specifically 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 specifically 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 classification) 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 efficiency 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 final 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 specifically 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 times, with several at just tens of milliseconds. The significance of this performance improvement was vastly greater than expected, yet there remains a rich breadth of areas to continue exploring to further optimize the performance of the data warehouse, including advanced indexing techniques, introduction of parallel query execution, and further partitioning of the data warehouse.

Acknowledgements

We acknowledge and thank student research assistants Evan Mackinnon and Sein Park of Okanagan College for their contributions in the initial design and scripting phases of this research as former members of our research team.

References

[1] R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. O'Reilly, 3 ed., 2013.

[2] T. B. Pedersen and N. Tryfona, "Pre-aggregation in spatial data warehouses," 2001.

[3] A. Hassan, F. Ravat, O. Teste, R. Tournier, and G. Zurfluh, "Differentiated multiple aggregations in multidimensional databases," 2015.

[4] H. Wang, X. Qin, X. Zhou, F. Li, Z. Qin, Q. Zhu, and S. Wang, "Efficient query processing framework for big data warehouse: an almost join-free approach," *Frontiers of Computer Science*, vol. 9, no. 2, pp. 224–236, 2015.

[5] T. P. G. D. Group, "Documentation PostgreSQL 10.20," 2022.

[6] S. Corporation, "Oracle Database Performance Tuning," vol. 12, no. 2, 2003.

[7] A. Wong, C. Chiu, A. Abdulgapul, M. N. Beg, Y. Khmelevsky, and J. Mahony, "Forecasting of Hourly Utility Usage." 2021.

[8] A. Wong, C. Chiu, A. Abdulgapul, M. N. Beg, Y. Khmelevsky, and J. Mahony, "Estimation of Hourly Utility Usage Using Machine Learnin," in *SysCon 2022 (accepted for the publication)*, 2022.

[9] A. Wong, D. Joiner, C. Chiu, M. Elsayed, K. Pereira, Y. Khmelevsky, and J. Mahony, "A Survey of Natural Language Processing Implementation for Data Query Systems," in *2021 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, pp. 1–8, 2021.

[10] G. J. D. R. Hains, Y. Khmelevsky, and T. Tachon, "From natural language to graph queries," in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pp. 1–4, IEEE, 2019.

[11] G. Hains, A. Jakobsson, and Y. Khmelevsky, "Towards formal methods and software engineering for deep learning: Security, safety and productivity for dl systems development," in *12th Annual IEEE International Systems Conference, SysCon 2018 - Proceedings*, 2018.

[12] G. Hains, Y. Khmelevsky, R. Bartlett, and A. Needham, "Game private networks performance: Analytical models for very-large scale simulation," in *2016 IEEE International Conference on Cybercrime and Computer Forensic, ICCCF 2016*, 2016.

[13] G. Hains, C. Li, D. Atkinson, J. Redly, N. Wilkinson, and Y. Khmelevsky, "Code generation and parallel code execution from business UML models: A case study for an algorithmic trading system," in *Proceedings of the 2015 Science and Information Conference, SAI 2015*, 2015.

[14] G. Hains, C. Li, N. Wilkinson, J. Redly, and Y. Khmelevsky, "Performance analysis of the parallel code execution for an algorithmic trading system, generated from UML models by end users," in *Parallel Computing Technologies (PARCOMPTECH), 2015 National Conference on*, pp. 1–10, IEEE, 2015.

[15] G. Hains, C. Mazur, J. Ayers, J. Humphrey, Y. Khmelevsky, and T. Sutherland, "The WTFast's Gamers Private Network (GPN®) Performance Evaluation Results," in *2020 IEEE International Systems Conference (SysCon)*, pp. 1–6, IEEE.

[16] Y. Khmelevsky and S. Dhanjal, "Information Security and Data Protection in Computer Science Education," in *12th Western Canadian Conference Education on Computing Education (WCCCE-2007), Thompson Rivers University, Kamloops, Canada, May*, pp. 3–5, 2007.

[17] Y. Khmelevsky, "SW Development Projects in Academia," *WCCCE 2009 - Proceedings of the 14th Western Canadian Conference on Computing Education*, vol. 1, no. 250, pp. 60–64, 2009.

[18] Y. Khmelevsky, "Ten Years of Capstone Projects at Okanagan College: A Retrospective Analysis," in *Proceedings of the 21st Western Canadian Conference on Computing Education*, (New York, NY, USA), pp. 7:1–7:6, ACM, 2016.

[19] Y. Khmelevsky, K. Chidlow, K. Sugihara, and K. Zhang, "Engaging and Motivating Students Through Programming Competitions and GIS Applied Research Projects," *Proceedings of the 22nd Western Canadian Conference on Computing Education*, 5 2017.

[20] Y. Khmelevsky, G. G. Hains, and C. Li, "Automatic Code Generation Within Student's Software Engineering Projects," in *WCCCE '12*, pp. 29–33, 2012.

[21] Y. Khmelevsky and G. J. Hains, "Parallel Programming Applied Research Projects for Teaching Parallel Programming to Beginner Students," 2021.

[22] Y. Khmelevsky, X. Li, and S. Madnick, "Software development using agile and scrum in distributed teams," in *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, 2017.

[23] Y. Khmelevsky, H. Mahasneh, and G. J. D. R. Hains, "A stochastic gamer's model for on-line games," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–4, IEEE, 2017.

[24] Y. Khmelevsky, M. Rinard, and S. Sidiroglou-Douskos, "A Source-to-source Transformation Tool for Error Fixing," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, (Riverton, NJ, USA), pp. 147–160, IBM Corp., 2013.

[25] Y. Khmelevsky, V. Ustimenko, G. Hains, C. Kluka, E. Ozan, and D. Syrotovsky, "International collaboration in SW engineering research projects," in *Proceedings of the 16th Western Canadian Conference on Computing Education - WCCCE '11*, 2011.

[26] Y. Khmelevsky and V. Voytenko, "Hybrid Cloud Computing Infrastructure in Academia.," in *WCCCE 2015 - the 20th Western Canadian Conference on Computing Education, At May 8-9, 2015.*, Vancouver Island University (VIU), Nanaimo, British Columbia, Canada., 2015.

[27] A. Wong, C. Chiu, G. Hains, J. Humphrey, Y. Khmelevsky, C. Mazur, and H. Fuhrmann, "Gamers Private Network Performance Forecasting - From Raw Data to the Data Warehouse with Machine Learning and Neural Nets," in *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (submitted for the review)*, 2021.

[28] C. Mazur, J. Ayers, J. Humphrey, G. Hains, and Y. Khmelevsky, "Machine Learning Prediction of Gamer's Private Networks (GPN®S)," in *Proceedings of the Future Technologies Conference*, pp. 107–123, Springer, 2020.

[29] T. Sinha, "OLAP vs. OLTP: What's the Difference?," *IBM Blog*, 3 2021.

[30] K. S. Ranti, D. Tuapattinaya, C. Chang, and A. S. Girsang, "Data warehouse for analysing music sales on a digital media store," *Journal of Physics: Conference Series*, vol. 1477, no. 3, 2020.

[31] G. S. Reddy, R. Srinivasu, M. P. C. Rao, and S. R. Rikkula, "Data Warehousing, Data Mining, OLAP and OLTP Technologies are essential elements to support decision-making process in industries," *International Journal on Computer Science and Engineering*, vol. 2, no. 9, pp. 2865–2873, 2010.

[32] P. Martins, P. Tomé, C. Wanzeller, F. Sá, and M. Abbasi, "Comparing Oracle and PostgreSQL, Performance and Optimization," 2021.

[33] G. Garani, A. Chernov, I. Savvas, and M. Butakova, "A Data Warehouse Approach for Business Intelligence," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 70–75, 2019.

[34] H. Mahboubi and J. r. Darmont, "Query Performance Optimization in XML Data Warehouses," in *E-Strategies for Resource Management Systems*, pp. 232–253, IGI Global, 2017.

[35] A. Rosenberg, "Improving Query Performance in Data Warehouses ," 2006.

[36] H. Märtens, E. Rahm, and T. Stöhr, "Dynamic query scheduling in parallel data warehouses ," 2003.

[37] F. A. Khan, A. Ahmad, M. Imran, M. Alharbi, Mujeeb-ur-rehman, and B. Jan, "Efficient data access and performance improvement model for virtual data warehouse ," 2017.