PostgreSQL Data Warehouse Implementation and Performance Optimization For Energy Companies*

*Published: https://ieeexplore.ieee.org/abstract/document/10553614
Note: (Sub-titles are not captured in Xplore and should not be used)

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

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

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

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

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

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

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*—With smart grids replacing traditional energy grids in recent years, energy companies are facing a new challenge to efficiently manage and analyze the collected data. The use of sensors in smart grids has led to a surge in the data being collected. Due to its design, the classic relational Online Transaction Processing (OLTP) relational database management system (RDBMS) starts to become inefficient for queries against the RDBMS beyond a terabyte size. For data extraction efficiency on large volumes of data, OLAP along with data warehousing (DW) has become a popular solution. A Business Intelligence (BI) tools are often used on the top of DWs [1]. The adoption of big-data-driven technologies is still lagging among energy companies.**

In this paper, based on our previous research projects and research results [2]–[5], we show that, through an optimization process, a gain in performance that is up to 2800 times faster compared with that of the original OLTP DBMS using the same hardware and the same operating system is possible.

The applied research project was funded by an NSERC grant and was conducted in 2020-2023 at Okanagan and Langara Colleges.

***Index Terms*—Smart Meters, OLTP, Data Warehouse, Performance, PostgreSQL DBMS, Design, Tuning.**

I. Introduction

Historically, Harris SmartWorks, an NSERC project industrial client, has employed an OLTP database to serve clients' needs. While it performs well for daily transactions, its performance is affected by the increasing complexity of customer data extraction and reporting requests. A data warehouse is a cost-effective and efficient solution to this problem. By storing the data used in the extraction and reporting in a different database, it may be possible to enhance the performance of the OLTP and the efficiency of the reporting process.

We are thankful for the support of our research to NSERC and Harris SmartWorks, as well as to students and faculty of Okanagan and Langara Colleges.

The principles and properties of data warehousing lend themselves well to more efficient data retrieval. Data pre-aggregation to reduce the number of joins, bitmap indexing, partitioning, and parallel processing are targeted techniques to increase performance. Although data warehouses excel in these areas, some concessions must be made to achieve the desired results. These concessions can be strategically chosen only to keep what is necessary for data analysis.

Overall, the transformation of the database structure can lead to significant improvements in speed and space used while purposefully leaving out unnecessary information for data analysis. Note that the data warehouse is not the sole solution. It is usually used in parallel with the OLTP to offset its weaknesses.

Harris SmartWorks, in collaboration with Okanagan and Langara Colleges through several student capstones and funded applied research projects, tested the use of data warehousing from August 2021 to August 2022 to determine if it is a viable solution [2]–[5]. Many previous students' applied research and capstone projects since 2007 contributed to the great success of this project [6]–[22]. The results of these efforts led to the work documented in this paper.

The contributions of this applied research paper are in the following areas: (1) the investigation of several different DW schemas, (2) the rigorous testing for the suggested solutions (including execution path valuations for different queries), and the gain of the performance for specific SQL queries was demonstrated (from 1500 to 2800 times performance improvement according to the comparisons made in figures), (3) the reduction of storage space in addition to the reduction of analytical queries execution time was also demonstrated.

II. Related Works

DW focuses on the "analytical processing of historical data to support business decision-making, whereas OLTP databases focus on transactional processing with queries to run frequently to update and report on the database" [23]–[25]. OLTP databases are designed for multiple concurrent transactions, but DWs "are designed to aggregate large volumes of data for OLAP, business intelligence, machine learning, and data mining" [26].

"A solid understanding of what is available and what needs to be achieved will enhance the choices. The star schema is the most well-known design schema for a DW. In a DW, a fact table is linked to secondary dimensional tables through primary/foreign key relationships" [1]. Snowflake and galaxy schemas are the most commonly used designs for the DW.

"Specialized design schema may be more applicable for specific business needs" [27].

DWs have been used to optimize performance, using "partitioning, aggregation, indexing, and parallelization [28]–[30]. Multiple strategies are used concurrently in many applications, but few conduct testing "between different database variations to gauge performance" [31].

III. OLTP Implementation and Base Performance

Harris SmartWorks provided the research team with the OLTP architecture; five tables were identified as critical from it. They should be the focus of the development of the data warehouse. These tables contain data on meters, such as location and the associated interval readings.

A general description of the contents of the data is provided below:

- Time-stamped read data from every meter channel used
- Meter data, including the first eight channels available to record read data from meters
- Data on all locations where the meters were installed
- A bridging table between meters and locations, with an indicator on whether a meter is active at a given location
- A table to allow for storage of extra channels due to new meters having more channels than expected.

A view was created for channel data that did not fit neatly with the tables above. This view stores data on each channel of each meter, in particular, the channel's unit of measure. It also holds data on any extra channels outside the first eight channels such that all channels on a single meter are contained in this view.

As size is an essential factor in this project's scope, it is crucial to understand how much storage space is being used by the OLTP. Tables I and II show the breakdown of storage usage for each schema table and the size of indexes for each table. To avoid disclosure of information, tables are named table 1, table 2, and so on.

A. Performance Metrics for the OLTP

To evaluate the performance of the databases, seven queries were provided by Harris Smartworks (see [4]):

TABLE I
OLTP's Tables and Their Indexes Size

SmartWorks Tables	Rows of data	Table Size	Index Size	Total Size
Table 1	1,776,000 k	179 GB	239 GB	418 GB
Table 2	90 k	16 MB	6 MB	22 MB
Table 3	39 k	6 MB	2 MB	8 MB
Table 4	83 k	8 MB	2 MB	10 MB
			Grand Total:	**418** GB

TABLE II
Size of Indexes for tables of the OLTP Relating to Table I

	Index Size
all interval readings	**239 GB**
interval readingsp1	98 GB
interval readingsu1	67 GB
interval readingsi1	37 GB
interval readingsi2	37 GB
meters	**6 MB**
metersp1	3 MB
metersi1	3 MB
all locations	**2 MB**
locationsp1	1 MB
locationsi1	1 MB
all meter location info	**2 MB**
meter locationi1	2 MB
Grand Total:	**239 GB**

- Query 1: Month-to-date usage of residential properties
- Query 2: Year-to-date usage of residential properties
- Query 3: Residential night usage ratio
- Query 4: Top 10 residential locations by usage in a month
- Query 5: Top 5 hours of usage during a month
- Query 6: Average maximum interval at residential locations per day for a month
- Query 7: Top 10 largest increases in usage (%) for residential meters

The baseline timing results of these queries to the OLTP are shown in Figure 1.

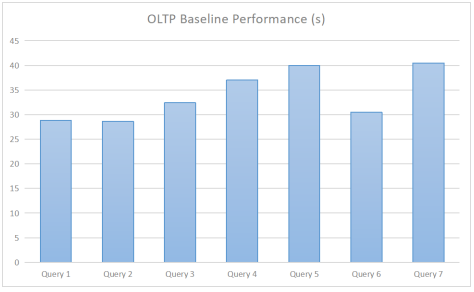

Fig. 1. Baseline Performance of the OLTP Queries from Table VII

IV. Data Warehouse Architecture and Design

The data warehouse, named Star 1 (Version 1), was designed on modified tables from the OLTP and an initially designed Star schema. Identified improvements to the original schema by Harris SmartWorks necessitated a change to the modified version.

The move from the initial Star schema to Star 1 was driven by the need to simplify and streamline the design of the different tables. In particular, two changes were made that would likely result in performance enhancements. First, a unique value (location_key) was removed from the star.location table, as it was unnecessary, and its inclusion resulted in table bloat in that it provided no new data that could not be derived through the Extract, Transform and Load (ETL) process. Next, the read_facts table was refactored to hold a fact for each read from a channel on a meter rather than all read from every channel of a meter in one record at any read time stamp. This allowed for a dynamic setup where read_facts now holds the read_val and Unit of Measurement (UoM) of every channel for each meter while removing reference to channel_id. Additionally, more entries can be added for extra channels rather than more channel columns or another table to hold all the added channels. Although removing an identification column is not a usual practice, the unit of measurement is unique for a given channel_id and commodity type. This means a primary key can be made without the use of channel_id. These changes make Star 1 simpler and streamlined compared to its predecessor.

The storage space taken by the Star 1 schema is tracked as it is a concern in the implementation phase. Table III shows a breakdown of the space requirement for the schema, its tables, and their indexes. Furthermore, Table IV shows the size of the indexes for each table.

TABLE III
SIZE OF TABLES AND THEIR INDEXES ON THE DW

Star 1	**Rows**	**Table Size**	**Index Size**	**Total Size**
read_facts	1,759,000 k	137 GB	170 GB	307 GB
meter_channel	92 k	8 MB	3 MB	10 MB
location	39 k	4 MB	1 MB	5 MB
date_time	1,284 k	84 MB	28 MB	111 MB
			Grand Total:	307 GB

TABLE IV
SIZE OF INDEXES FOR EACH TABLE OF THE DW STAR 1

	Index Size
Star1.read_facts	**170 GB**
read_facts_pk	67 GB
alt_key_idx	67 GB
read_dt_idx	37 GB
Star1.meter_channel	**3 MB**
meter_id_pk	3 MB
Star1.location	**1 MB**
loc_no_pk	1 MB
Star1.date_time	**28 MB**
read_dtm_pk	28 MB
Grand Total:	**170 GB**

Baseline performance testing on the Star 1 schema was conducted with the seven queries described above. These queries must return the same results as those for the OLTP to ensure a meaningful performance comparison between the DW and the OLTP. The results of these performance tests can be seen in Figure 2.

Query results were also compiled using different location classes to see specifically how the size of the data returned would impact performance. The RI class contains records related to residential usage data, and the CO class contains records related to commercial usage data. Table V shows the sizes involved in the testing of two location classes. Looking at Figure 3, the results show a marginal increase in query performance when there is a smaller amount of data to parse. As mentioned, query 5 does not utilize the location class, so it is omitted. Table VI shows the comparison of query times for both of the location classes.

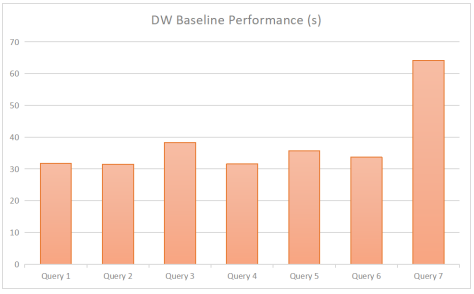

Fig. 2. Baseline performance of the DW Queries from Table VII

Fig. 3. DW Performance Results for the Location Class RI and CO

TABLE V
NUMBER OF LOCATIONS AND RECORDS INVOLVED IN THE TWO LOCATION CLASSES - 2021-02-01 TO 2021-03-01

Loc Class	# of Locations	Number of Records
RI	29,254	14,017,499
CO	152	48,384

TABLE VI
QUERY TIMES (IN MS) FOR THE LOCATION CLASSES RI AND CO

Query	Q1	Q2	Q3	Q4	Q6	Q7
RI	31.7	31.5	38.3	31.6	33.7	64.2
CO	27.0	26.3	33.2	26.7	26.9	53.6

V. PERFORMANCE COMPARISONS BETWEEN THE OLTP AND THE DW

Initial performance comparison in terms of query time between the OLTP and the DW are shown in Figure 4

TABLE VII
QUERY TIMES (IN MS) OF OLTP VS THE DW

Query	OLTP (ms)	STD	DW (ms)	STD	OLTP/DW
Query1	28,833	427	31,693	1,075	0.91
Query2	28,569	105	31,477	1,016	0.03
Query3	32,421	746	38,334	1,103	0.85
Query4	36,993	780	31,625	754	1.17
Query5	50,027	750	35,678	952	1.40
Query6	30,524	747	33,678	988	0.91
Query7	40,546	3,171	64,159	944	0.63

and Table VII. It is easy to see that performance is comparable between the OLTP and the DW for the majority of the queries.

An area of concern on the project is the physical storage size of the DW versus the OLTP RDBMS [4]. To that end, the DW keeps only the data necessary for analysis. The storage comparison can be seen in Figure 5. There is a reduction in data size moving from the OLTP to the DW.

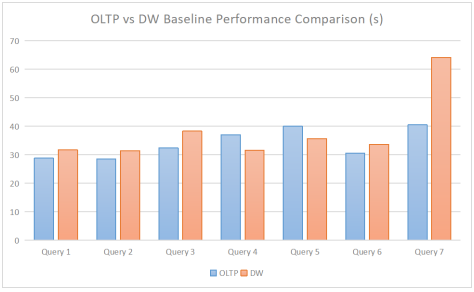

Fig. 4. Baseline Performance - OLTP versus DW

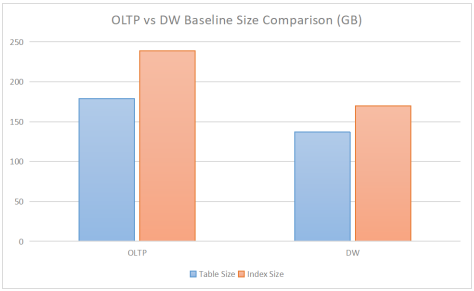

Fig. 5. Storage comparison of OLTP and the DW Based on Tables I and III

VI. PERFORMANCE TESTING OF DW WITH PRE-AGGREGATION VS OLTP

The DW's baseline performance is insufficient to support the transition from the OLTP to a DW. It was realized that users do not always need to read data in fifteen-minute intervals. Queries that aggregate data hourly, daily, or monthly are common. Therefore, the use of pre-aggregation in the DW makes sense. Since this approach involves summarizing multiple data readings into one, it results in smaller tables. The most significant benefit is that queries do not have to spend computational time on aggregation. Data retrieval times are minimized by front-loading computationally heavy aggregation operations as part of the ETL process.

The DW and OLTP DBMS systems were tested on Dell Servers with 40 cores, 96 GB RAM, and 4 TB SSD drives using PostgreSQL 12 in Alma Linux 64-bit OS.

Four aggregation scenarios were selected to demonstrate the effectiveness of pre-aggregation:

- Monthly_Usage: Aggregates all the readings for one channel of a meter in a month into one record
- Daily_Usage: Aggregates all the readings for one channel of a meter in a day into one record
- Location_Class_Reads_hrly: Aggregates all the readings for a location class every hour into a record
- Meter_Type_Reads_hrly: Aggregates all the readings for a meter type every hour into one record

Although pre-aggregation would be an excellent tool for performance, the concern was the extra storage required for these tables. To minimize the extra space used, the aggregation scenarios were used to aggregate what is useful. This helped to reduce the size of individual tables and prevent having too many tables that stack up in size. Table VIII shows the physical storage space used by the pre-aggregated tables. Table IX shows the physical space used by the indexes of these tables. Most importantly, Figure 6 compares the OLTP and DW in terms of physical storage used, with the pre-aggregated tables included.

TABLE VIII
SIZE OF PRE-AGGREGATED TABLES AND THEIR INDEXES

PreAgg	Rows	Table Size	Index Size	Total Size
Monthly_Usage	2,241 k	216 MB	126 MB	342 MB
Daily_Usage	65,519 k	6,234 MB	3,942 MB	10,176 MB
Location_Class	1,180 k	94 MB	47 MB	141 MB
Meter_Type	2,847 k	243 MB	136 MB	379 MB
			Grand Total:	11 GB

TABLE IX
SIZE OF PRE-AGGREGATED TABLES INDEXES

	Index Size
Star1.Monthly_Usage	**174 MB**
monthly_usage_pk	126 MB
Star1.Daily_Usage	**3,942 MB**
daily_usage_pk	2,539 MB
read_dt_idx	1,404 MB
Star1.Location_Class_Reads	**47 MB**
location_class_reads_pk	47 MB
Star1.Meter_Type_Reads	**136 MB**
meter_type_reads_hrly_pkey	136 MB
Grand Total:	**4,299 MB**

There is a noticeable improvement in speed through the use of the pre-aggregated tables as detailed in Figures 7 and 8. The results show that the Location class table has the best results, with the meter_type and monthly_usage tables following close behind. The Daily_usage table has less of an improvement but is still much better than the baseline for OLTP (see Runtime of OLTP vs DW Pre-Aggregated for 10 test runs results in our initial research paper here [4]).

Comparable results from the tests of the OLTP using the CO location_class to those using the pre-aggregated queries

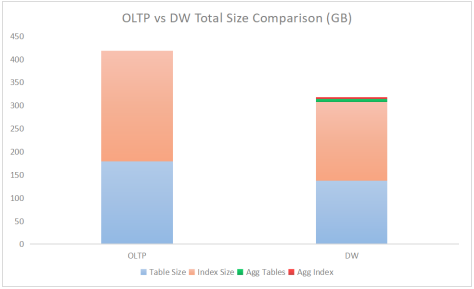

Fig. 6. Storage comparison of OLTP and DW based on Tables I, III and VIII

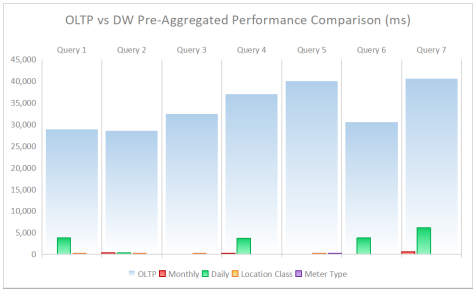

Fig. 7. OLTP vs PDW Performance

are in Figure 9 and Table X. Query times within the DW are lower than those within the OLTP. The pre-aggregation queries use the RI location_class compared with the OLTP using CO location_class.

TABLE X
QUERY TIME (IN MS) FOR OLTP WITH LOCATION_CLASS CO VERSUS PRE-AGGREGATION)

Query	Q1 (ms)	Q2 (ms)	Q3 (ms)	Q4 (ms)	Q6 (ms)	Q7 (ms)
OLTP	854	911	901	852	944	1,673
DW Agg	13 Loc Class	13 Loc Class	13 Loc Class	267 Monthly	3,779 Daily	593 Monthly
OLTP/DW Ratio	65.69	70.07	69.30	3.19	0.24	2.82

VII. PERFORMANCE IMPROVEMENT FOR THE DW

To further improve the performance of the DW, the following techniques were considered in the research:

- Partitioning
- Parallel Querying
- Bitmap indexing

A. *Partitioning*

Partitioning has been made viable for performance improvements in version 14 of PostgreSQL, which was used for the research. It has several advantages and some drawbacks, as discussed below.

Using range partitioning, a large table with millions of records can be split into multiple tables, each corresponding to a different range of data. Figure 10 shows an example range partition on time and the ability to sub-partition on more fine-grained ranges.

There are some simple rules on how partitioning works. One is that they work with inheritance, meaning child tables must

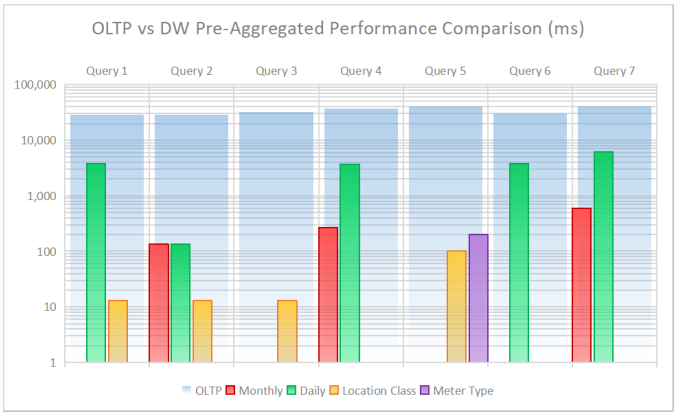

Fig. 8. OLTP vs PDW Performance May 12th.

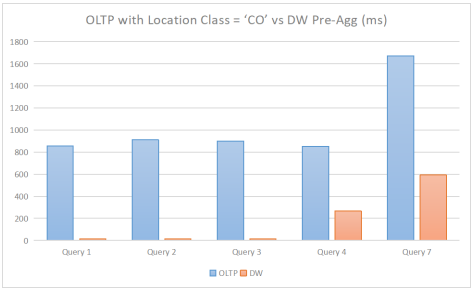

Fig. 9. Comparison of the OLTP queries with location class CO vs the fastest Pre-Aggregated queries. Time data from Table X.

have the same columns as their parents. Also, a partitioned table cannot be reclassified as a standard table and vice versa. Data must be transferred to a new table. Data cannot be moved between partitions by users unless it is also changed to fit into its new partition [32].

The pre-aggregated read facts table has a three-layer structure that helps with search performance and organization. Each partition has a range of values designated to it, determining what data it will hold.

Accessing the data is simple: query the top table, and the program can automatically tell where to look and search through the lower tables. This also allows for quickly dropping old data. You can select the specific partition, call for it to be dropped, and eliminate the whole data section without

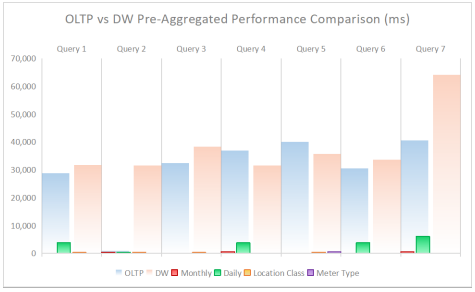

Fig. 10. Diagram of potential partitioning structure

scanning the rest of the table to find all the relevant values.

As shown in Table XI, partitioning allows for significant improvements. Figure 11 compares the performance of OLTP, DW and Partitioned DW.

TABLE XI
PERFORMANCE OF OLTP, DW VS PARTITIONING (IN MS)

4 Workers	OLTP	DW	Partitioning
Query 1	28,100	31,693	3,368
Query 2	29,805	31,477	4,033
Query 3	30,980	38,334	5,290
Query 4	35,537	31,625	3,339
Query 5	39,778	35,678	3,554
Query 6	39,344	33,678	20,293
Query 7	40,359	64,159	8,045

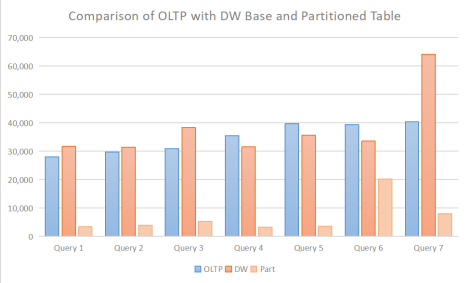

Fig. 11. Comparison of OLTP, DW, and DW Partitioning, The partitioned table has four worker processes

Further optimization of performance is also possible. For example, putting partitions on separate storage devices to allow for more optimized usage of I/O resources is one method that can be applied.

B. Parallel Processing

Parallelization is a useful tool to reduce run time by dividing up the work that needs to be done in executing a query. The documentation in [32] shows how it works in PostgreSQL. Splitting the work of one query between many worker processes reduces the time to execute a single query. It does not necessarily work for all processes, but it does work for many. Also, settings are available to fine-tune performance.

In PostgreSQL, parallelization is used at the discretion of the optimizer. In determining the most effective route, the optimizer will consider parallelization where applicable and apply it if it is time-efficient to do so.

If the optimizer decides that parallelization should be deployed, it creates a gather or gathers merge node, requests a number of workers, and splits the work between the workers acquired. A gather node quickly merges all records retrieved by the workers. A gather merge node will work at a slower pace to ensure that a sorted order is maintained.

There is a leader process that will take care of the gathers. When a larger amount of data comes in, the leader will do less of a worker's duties and spend more time directing. For example, on a small set of data being parallelized, the leader will not have as much to gather and will spend some time working with the workers.

Parallelization is not guaranteed to work at a high level of efficiency using default parameters. There are options to tune performance and resource usage. To make parallel queries more efficient, *parallel_setup_cost* and *parallel_tuple_cost* can change how the optimizer calculates the cost of using parallel queries.

The most important aspect of parallelism is knowing where it comes into play. These areas are scans, joins, aggregates, and appends. The leader process also has its own set of tasks it may take on part from its workers. There are also some places where parallel queries will be restricted from running. Each area will be briefly described below.

Parallelism works for scans (specifically sequential, bitmap heap, and b-tree index scans). This is promising as using indexes with parallel processing, or even bitmap heaps, allows for further performance improvement. Also, in *"Create Table ... As"*, the SELECT statement within it can be parallelized, but not the part where data is written. This also works for CREATE and REFRESH commands with materialized views, meaning that processing time could be further reduced.

For joins, the outer joins and nested loop joins with a possible inner can all be parallelized. Hash joins have two versions. One is half parallel, where every process performs the inner join, which can be slow depending on the table used. Another version is a full parallel where all processes work together on the inner join, followed by their outer join. There is also a merge join option, which is inefficient in most cases.

In aggregation, workers will make partial aggregate nodes with their data subset. Gather or gather merge will be used to bring all the records together into a finalized aggregate node. In an aggregation process, parallelization efficiency is maximized when the data is more highly aggregated and vice versa.

Appends have two methods. Normal appends have workers splitting the work of each child node, consecutively working through one at a time. Parallel appends allow workers to be allocated concurrently between all child nodes while allowing for multiple workers on one node in case it has a more significant workload. The leader process can work on parallel restricted processes, as was mentioned earlier [32].

Finally, a few circumstances will prevent the optimizer from using parallel queries. These include data being written, database rows being locked, the possibility of the query being suspended, parallel unsafe functions, reaching maximum parallel workers or a worker process getting to a place where parallelization could happen if it were not a worker. Aggregation cannot be parallelized if DISTINCT, ORDER BY, or grouped sets are used.

Parallelization is a great tool to have to increase efficiency in query execution. It can be used in many places, but most importantly, it can be set up with several workers ready to use. The optimizer will take care of the rest, as it is what determines when parallelization should be used. By default, PostgreSQL is set up to allow the optimizer to schedule two

workers per query and has already been used by both the DW and OLTP.

According to the Postgress 2022 technical documentation [32], Materialized views are updated manually using the REFRESH MATERIALIZED VIEW *viewname;* command. Data access speeds on materialized views are faster than the base table from which they originate.

Materialized views also have access rules. These are similar to triggers with subtle differences in how they work, allowing them to excel in certain areas. Rules, instead of firing for every individual row that would be affected by a trigger, will take the query used and replace it with a new query that uses the NEW and OLD parameters to control what is changed. This allows for queries that affect large amounts of the data to be redirected only once rather than for every individual row and is, therefore, more efficient in time. Blank rules can prevent updates or inserts much more effectively for this exact reason. The downsides of rules are that they are a bit more complicated to write and cannot ensure parity of foreign key constraints.

Having to refresh the materialized view manually would not be ideal. However, it would be possible to set up a scheduler to automate the command at a given time. Also, according to the documentation [32] for REFRESH MATERIALIZED VIEW, there is the CONCURRENTLY option. This option requires a unique index on the table that includes all rows of the view. When this option is used, it allows for the view to be accessed during refresh and can even be faster for minor updates.

Materialized views are a valuable tool for a DW implemented in PostgreSQL. It gives access to faster data retrieval and rules for more efficient data control.

Materialized views were made, replicating the daily usage, monthly usage, and the location class pre-aggregated tables. They were then given the same indexes as these tables and tested with the same queries. The query performance on the materialized views versus the OLTP is presented in Table XII. Table XIII "Performance Comparison between DW Materialized Views and OLTP" and Figures 12 to 14 show the comparison of performance between materialized views and pre-aggregated tables. The Table XIII is based on Table XII.

TABLE XII
PERFORMANCE COMPARISON BETWEEN DW MATERIALIZED VIEWS AND OLTP

Query	OLTP (ms)	DW (ms)	OLTP/DW Ratio	View
1b	28,833	3,941	7	Daily
1c		10	2,883	Loc. Class
2a	28,569	269	106	Monthly
2b		3,654	8	Daily
2c		20	1,428	Loc. Class
3c	32,421	22	1,474	Loc. Class
4a	36,993	290	128	Monthly
4b		3,730	10	Daily
5c	40,044	107	374	Loc. Class
6b	30,524	3,723	10	Daily
7a	40,546	609	67	Monthly
7b		5,956	7	Daily

TABLE XIII
PERFORMANCE COMPARISON BETWEEN DW MATERIALIZED VIEWS AND OLTP

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Monthly PA	n/a	135	n/a	267	n/a	n/a	593
Monthly MV	n/a	269	n/a	290	n/a	n/a	609
Daily PA	3,769	135	n/a	3,371	n/a	3,779	6,139
Daily MV	3,941	3,654	n/a	3,730	n/a	3,723	5,956
Loc PA	13	13	13	n/a	101	n/a	n/a
Loc MV	10	20	22	n/a	107	n/a	n/a

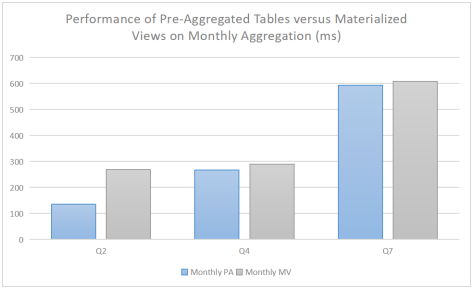

Fig. 12. Time data from Table X.

C. Bitmap indexing

PostgreSQL does not allow for manual creation of Bitmap indexes. BRIN (Block Range Index) is a potential alternative, which is good for large data sets. It covers packaging adjacent pages and storing them in the index if certain conditions are met. This is most useful if there is a sorted order. For example, the index would note the dates in its range when queries are run on a date. If it contains a desired date, the process will mark it to be run through. Otherwise, the entire range is discarded for the search. The amount of pages per range is set at the creation of the index.

It has the added security that it executes directly on the database as a user sets up the job, and only the user can view the created jobs.

VIII. CONCLUSIONS

In this paper, we discussed the results and the various considerations in moving a large volume of data from an OLTP to a DW to improve reporting and analysis capabilities. Results on optimizing performance in processing time and data storage were shared. We show that it is possible to gain 1500 to 2800 times in performance using the DW compared to the underlying OLTP using the same hardware and operating systems. Our research results prove that, for a company, DW implementation and performance optimization are valuable solutions for reducing processing time, storage space, and computational resources, especially in a cloud environment.

ACKNOWLEDGMENT

We thank Diomari Fortes of Okanagan College for his work building a testing data warehouse and gathering research as a student and former research team member. Students from capstone project teams at Okanagan College and Langara

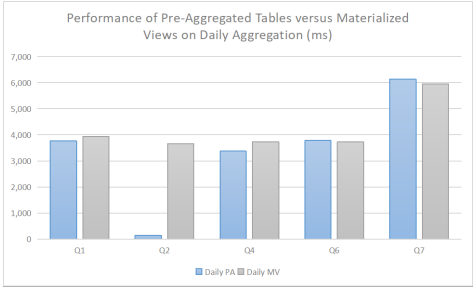

Fig. 13. Time data from Table X.

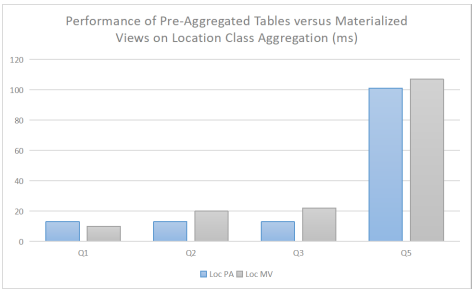

Fig. 14. Time data from Table X.

College have also contributed to the research and development of the DW. Their contributions were very much appreciated.

We thank the reviewers for their suggestions on how to improve our paper.

REFERENCES

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

[2] Y. Khmelevsky and G. J. Hains, "Parallel Programming Applied Research Projects for Teaching Parallel Programming to Beginner Students," 2021. [Online]. Available: https://arxiv.org/abs/2105.13574

[3] C. Mazur, J. Ayers, G. Hains, and Y. Khmelevsky, "Machine Learning Prediction of Gamer's Private Networks," *CoRR*, vol. abs/2012.06480, 2020. [Online]. Available: https://arxiv.org/abs/2012.06480

[4] D. Joiner, M. Clement, S. T. Chan, K. Pereira, A. Wong, Y. Khmelevsky, J. Mahony, and M. Ferri, "DW vs OLTP Performance Optimization in the Cloud on PostgreSQL (A Case Study)," in *2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2022, pp. 1–8.

[5] 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 (FTC) 2020, Volume 2*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2021, pp. 107–123.

[6] 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*, 2007, pp. 3–5.

[7] ——, "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*, 2007, pp. 3–5.

[8] Y. Khmelevsky, "Information and data protection within a RDBMS," *Condensed Matter Physics*, 2008.

[9] ——, "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. [Online]. Available: http://dl.acm.org/citation.cfm?id=1536292

[10] Y. Khmelevsky, M. Govorov, and L. Burge, "Okanagan College and Vancouver Island University educational joint projects results," in *Proceedings of the 14th Western Canadian Conference on Computing Education - WCCCE '09*, 2009, pp. 65–69. [Online]. Available: http://portal.acm.org/citation.cfm?id=1536274.1536293

[11] Y. Khmelevsky, "Research and teaching strategies integration at post-secondary programs," in *Proceedings of the 16th Western Canadian Conference on Computing Education*, ser. WCCCE '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 57–60. [Online]. Available: https://doi.org/10.1145/1989622.1989638

[12] 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*, 2011, pp. 52–56.

[13] Y. Khmelevsky and V. Voytenko, "Strategies for teaching mobile application development," in *18th Western Canadian Conference on Computing Education*, vol. 18, 2013, pp. 8–13.

[14] ——, "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.

[15] N. McDonald, D. Atkinson, Y. Khmelevsky, and S. McMillan, "Sport wearable biometric data encrypted emulation and storage in cloud," in *Canadian Conference on Electrical and Computer Engineering*, 2016.

[16] D. Atkinson, N. McDonald, and Y. Khmelevsky, "Reporting personal and corporate data for secure storage in cloud," in *2016 IEEE International Conference on Cybercrime and Computer Forensic, ICCCF 2016*, 2016.

[17] 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: ACM, 2016, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2910925.2910949

[18] N. Mcdonald, D. Leader, C. K. Chiang, Y. Khmelevsky, R. Bartlett, and A. Needham, "A new online tool for gamer network performance analysis," in *2016 IEEE International Conference on Cybercrime and Computer Forensic (ICCCF)*, 2016, pp. 1–6.

[19] N. Mcdonald, D. Atkinson, C. Frank, Y. Khmelevsky, and S. McMillan, "Biometric data emulation and encryption for sport wearable devices (A case study)," in *2016 Annual IEEE Systems Conference (SysCon)*, 2016, pp. 1–6.

[20] 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. [Online]. Available: http://dx.doi.org/10.1145/3085585.3088491

[21] M. Cocar, R. Harris, and Y. Khmelevsky, "Utilizing Minecraft bots to optimize game server performance and deployment," in *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2017, pp. 1–5.

[22] B. Ward, Y. Khmelevsky, G. Hains, R. Bartlett, A. Needham, and T. Sutherland, "Gaming network delays investigation and collection of very large-scale data sets," in *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, 2017.

[23] T. Sinha, "OLAP vs. OLTP: What's the Difference?" *IBM Blog*, 3 2021. [Online]. Available: https://www.ibm.com/cloud/blog/olap-vs-oltp

[24] 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.

[25] 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.

[26] P. Martins, P. Tomé, C. Wanzeller, F. Sá, and M. Abbasi, "Comparing Oracle and PostgreSQL, Performance and Optimization," Cham, pp. 481–490, 2021. [Online]. Available: https://go.exlibris.link/s8crchW0

[27] 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)*, 2019, pp. 70–75.

[28] H. Mahboubi and J. r. Darmont, "Query Performance Optimization in XML Data Warehouses," in *E-Strategies for Resource Management Systems*. IGI Global, 2017, pp. 232–253. [Online]. Available: https://doi.org/10.4018%2F978-1-61692-016-6.ch014

[29] A. Rosenberg, "Improving Query Performance in Data Warehouses ," Seattle, p. 7, 2006. [Online]. Available: https://go.exlibris.link/m4kHc20P

[30] H. Märtens, E. Rahm, and T. Stöhr, "Dynamic query scheduling in parallel data warehouses," Chichester, UK, pp. 1169–1190, 2003. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.786

[31] 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 ," pp. 232–240, 2017. [Online]. Available: https://go.exlibris.link/jjrcywDr

[32] T. P. G. D. Group, "Documentation PostgreSQL 10.20," pp. 445–456, 2022.