MCO1: Query Proccessing in a Data Warehouse

David Joshua Corpuz
david.joshua.corpuz@dlsu.edu.ph
College of Computer Studies, De La Salle University
Manila, Philippines

Rafael Dimagiba
rafael_dimagiba@dlsu.edu.ph
College of Computer Studies, De La Salle University
Manila, Philippines

Steven Errol Escopete
steven_escopete@dlsu.edu.ph
College of Computer Studies, De La Salle University
Manila, Philippines

Ralph Pineda
ralph_pineda@dlsu.edu.ph
College of Computer Studies, De La Salle University
Manila, Philippines

ABSTRACT

This paper presents the design and implementation of a data warehouse, OLAP application, and ETL pipeline for the seriousMD appointment dataset aimed at supporting analytical tasks and decision-making processes. The data warehouse adopts a star schema dimensional model, featuring an appointment fact table and dimensions for doctors, patients, and clinics, considering simplicity, scalability, and performance. The ETL process, facilitated by Python and Apache Nifi, involves data preprocessing using Python's pandas library for cleaning and organization, followed by additional transformation and loading within Apache Nifi. With advanced SQL constructs such as CASE WHEN and WITH ROLLUP, the OLAP application aims to provide insights and analysis crucial for decision-making in healthcare policy, resource allocation, workforce planning, and patient care optimization. The paper also discusses query optimization strategies, including correct database design, index usage, and query restructuring, to enhance performance. Results from validation processes, including functional and performance testing, are presented, demonstrating the effectiveness of the implemented solutions.

KEYWORDS

Data Warehouse, Query Processing, Query Optimization, ETL, OLAP

ACM Reference Format:
David Joshua Corpuz, Rafael Dimagiba, Steven Errol Escopete, and Ralph Pineda. 2024. MCO1: Query Proccessing in a Data Warehouse. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The SeriousMD Appointment dataset contains the collection of appointments recorded by SeriousMD. The main table in this dataset is the appointment table which consists of information such as appointment status, start and end times, type, and whether it is done virtually or not as well as anonymized information which serves as the patients', doctors', and clinics' identification. Aside from this table, there are three other tables which contain information about patients, doctors, and clinics respectively.

Through the use of a data warehouse, data from this dataset can be archived into a nonvolatile repository allowing stakeholders to gain insights which may lead to more productive business operations as expressed by Oracle [1]. This objective can be realized through online analytical processing (OLAP) and its operations. To manipulate data models whether it is in a star schema, a schema which is composed of a fact table and dimension tables, or a snowflake schema which has additional secondary dimension tables, OLAP provides five operations: (1) Roll up, which aggregates a specific dimension, (2) Drill down, which is the opposite of the former, (3) Slice, which generates a two-dimensional model from the original three-dimensional model, (4) Dice, which produces a subcube from the original, and (5) Pivot, which rotates the data model [2].

This paper illustrates the process the group performed to obtain an understanding in the data given. In particular, the data warehouse formation, its data population through Python and Apache Nifi, dashboard designing, query performance evaluation and optimization, and conclusion are detailed. Through the accomplishment of these tasks various findings were observed concerning the development of an OLAP application and the effect of different optimizations strategies in terms of hardware, schemas, and the use of indices.

2 DATA WAREHOUSE

The data warehouse utilized to consolidate the data and develop deeper understanding and insights that may prove useful in business decisions consists of four tables. The appointments table serves as the fact table having the attributes of the patient ID (pxid), clinic ID, doctor ID, appointment ID (apptid), status, time queued, queue date, start and end times, type, and whether it is done virtually or not. This table is determined as the fact table as the appointment data is identified to be the central focus of this paper.

This fact table has a relationship with three dimension tables in the data warehouse. The clinic dimension table stores information about the clinics used in the fact table in particular, the clinic ID, hospital name, whether it is a hospital or not, and the city, province, and region it is located in. On the other hand, the doctor dimension table has information about the doctors present in the appointments.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

It contains the doctor ID, specialty (mainspecialty), and age. The third and last dimension table concerns patient data which contains the patient's id (pxid), age, and gender.

The data warehouse uses a star schema with the purpose of faster querying speed at the cost of possible data redundancy. The model simplifies querying by organizing data into easily understandable structures allowing users to query the fact table and join it with dimension tables using straightforward SQL queries. Figure 1 shows the star schema used in this data warehouse.

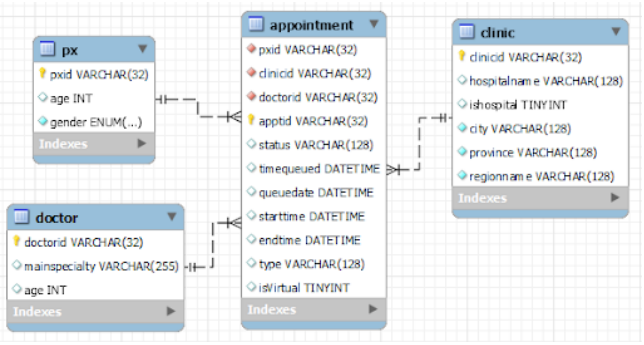

Figure 1: Database Schema

There are two main concept hierarchies present in the data warehouse. The first concept hierarchy is the location hierarchy evident in the clinics table. This table contains data about the location of the clinics namely in terms of province, region, and city. In these three attributes, city and region is identified to be the lowest level and highest level respectively, with the province being in the middle of the two. The other concept hierarchy determined concerns the time dimension. This can be found in the appointment table where the time and date where the appointment is queued, together with the start and end time are recorded. The levels were identified in the order of time, day, month, and year from lowest to highest.

3 ETL SCRIPT

Extract, transform, and load (ETL) is the process where data, possibly from various sources, is cleaned and organized and then stored in a data warehouse [3]. In this project, ETL is done through Python and Apache Nifi. Python and its library, pandas, which is a data manipulation tool [4] is used to preprocess the data before it is loaded into Apache Nifi, a data processing and distribution tool [5], for additional data transformation and loading.

The data preprocessing done through Python and pandas starts with checking the unique values of each column of the dataframe generated from the original CSV files provided using the pandas.unique function. From this initial verification, it is decided that the only data to preprocess is the patients and doctors records.

The first operation done to the patients data is the removal of rows with duplicate pxids. It was decided that only the first record will be kept. The next set of operations concerns the age values in the records. The non-null age values are converted to integers using the pandas.DataFrame.astype function and the rows with negative age values are removed using row selection based on a condition. After all the operations are completed, the edited data are then exported as a CSV file which will then be loaded into Apache NiFi.

The data preprocessing done in the doctors data concerns the 'mainspecialty' attribute. Records with 'mainspecialty' values deemed to be not a specialty are removed. These values include but are not limited to values such as emails, names, and unrelated phrases. Values with typographical errors and inconsistent character casings are also transformed to a specific constant representation. As with the patients data, the final data is also exported as a CSV file to be loaded into NiFi.

In NiFi, all tables are loaded using a similar processing pipeline. GetFile is used to load the CSV file generated by the data preprocessing done beforehand. UpdateAttrbute is then used to append the table name in the csv loaded by NiFi without changing the original CSV file source. ConvertRecord will then change the CSV into JSON format. The schema of the tables have to be defined in ArvonRegistry so ConvertRecord can use the table name appended by UpdateAttribute to determine the correct data types of the attributes. This JSON file can then be uploaded to the warehouse using PutDatabaseRecord.

4 OLAP APPLICATION

The OLAP application is to provide insights and analysis on various aspects of medical practice, including the distribution of medical specialties over time, the prevalence of different age groups among patients, and the comparison of virtual and nonvirtual appointments across different provinces and clinics. The application is intended for decision-making and analytical tasks related to healthcare policy, resource allocation, workforce planning, and patient care optimization. Some of the key decision-making or analytical tasks the application is designed for include specialty distribution analysis, regional appointment distribution trends, virtual appointment utilization, healthcare demographics and utilization patterns.

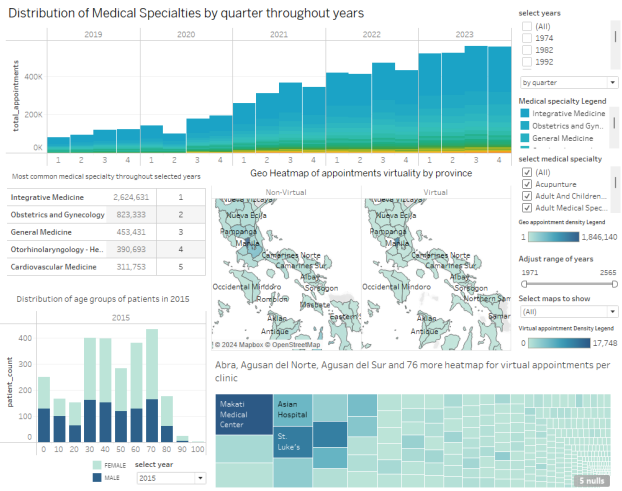

Figure 2: OLAP application dashboard

Stacked bar charts were primarily employed to enable healthcare professionals, administrators, and researchers to visually compare

the distribution of medical specialties or demographics across various dimensions. Their ease of interpretation, visual clarity, and applicability to categorical data make them a suitable choice for conveying complex information and aiding decision-making processes in the appointment dataset [12]. Additionally, heatmaps allow for easy comparison across multiple variables simultaneously, making them particularly useful for analyzing multidimensional data sets. By employing color gradients to represent varying levels of intensity or frequency, heatmaps effectively highlight areas of concentration or scarcity within the dataset, aiding in the identification of hotspots or areas requiring intervention [13].

4.1 Stacked Bar Chart for Medical Specialty Distribution Analysis

The report offers a comprehensive visual analysis tool for understanding the distribution of medical specialties across specified time periods. It visually represents changes in medical specialty demand, enabling users to identify emerging fields, analyze shifts in healthcare needs, and evaluate training trends. Additionally, it aids policymakers in formulating healthcare policies and forecasting future trends.

Through stacked bar charts, users can easily compare the proportions of different specialties over quarters or months, facilitating temporal analysis. The report allows for multi-year comparisons and provides filtering options to focus on specific specialties of interest, enabling users to identify trends, anomalies, and emerging patterns. By leveraging these insights, stakeholders can make informed decisions regarding resource allocation, workforce planning, and policy development within the healthcare sector.

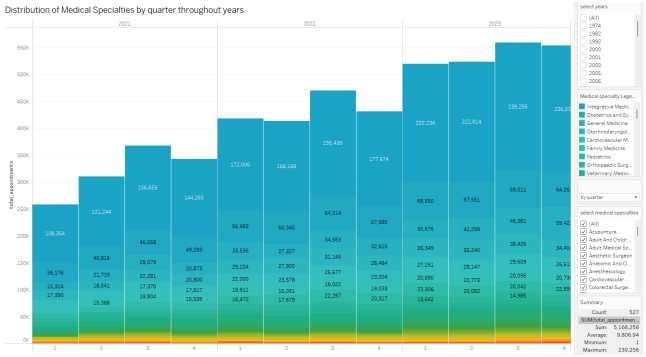

Figure 3: Distribution of medical specialties by quarter throughout 2021, 2022, and 2023

In Figure 2, the analysis of appointment trends reveals a steady increase in appointments each year across 2021, 2022, 2023, indicating a growing demand for medical services. This trend persists across all quarters, suggesting a sustained need for healthcare throughout the year rather than seasonal fluctuations. Notably, doctors specializing in integrative medicine emerge as the most sought-after practitioners, constituting approximately 30% of appointments each quarter. This prevalence underscores a potential shift in patient preferences towards holistic healthcare approaches. Consequently, healthcare facilities must adapt their resource planning, training programs, and marketing strategies to accommodate this growing interest in integrative medicine while ensuring efficient service delivery to meet the rising demand for appointments.

The corresponding SQL statement for the analytical report is as follows:

```
SELECT
    d.mainspecialty,
    YEAR(a.queuedate) AS appointment_year,
    QUARTER(a.queuedate) AS appointment_quarter,
    MONTH(a.queuedate) AS appointment_month,
    COUNT(*) AS total_appointments
FROM
    appointment a
        JOIN
    Doctor d ON a.doctorid = d.doctorid
GROUP BY d.mainspecialty, YEAR(a.queuedate),
QUARTER(a.queuedate), MONTH(a.queuedate)
WITH ROLLUP
```

By utilizing aggregate functions such as COUNT(*), the query calculates the total number of appointments for each combination of medical specialty, year, quarter, and month. Date functions such as YEAR(), QUARTER(), and MONTH() facilitate temporal organization, allowing for analysis across different time periods. Crucially, the GROUP BY clause groups appointments by medical specialty, year, quarter, and month, facilitating aggregation and analysis. The addition of WITH ROLLUP enables the generation of subtotal rows for each level of the hierarchy specified in the GROUP BY clause, facilitating analysis at different aggregation levels.

The report examines data across different dimensions including year, quarter/month, and medical specialty. Users have the option to adjust parameters, such as selecting whether to analyze data by quarter or month, filtering medical specialties, and specifying which years to include in the analysis. Roll-up and drill-down operation can be applied to aggregate data from detailed levels (i.e., individual months) to higher levels (i.e., quarters). In this report, users can also slice or dice the data by selecting particular medical specialties or years to focus on in the analysis.

4.2 Geo Heat Map for Regional Appointment Distribution Trends

Analytical reports generated offer a comprehensive understanding of healthcare dynamics across regions. These reports enable users to track appointment distribution trends over time, comparing the prevalence of virtual and non-virtual appointments within provinces. By conducting trend analysis of a provincial level and comparing data between different years, users can discern emerging patterns and assess the impact of various factors such as technological advancements or policy changes.

The report can highlight regional disparities in virtual appointment accessibility, aiding policymakers and healthcare providers in identifying areas requiring targeted interventions to improve healthcare access. User preference analysis provides insights into patient behavior, facilitating the tailoring of services to meet evolving needs. Moreover, heat maps displaying appointment density

help in resource allocation and infrastructure planning, particularly in areas with high demand for healthcare services. The analytical report play a crucial role in informing strategic decision-making, optimizing service delivery, and ultimately enhancing patient outcomes in healthcare systems.

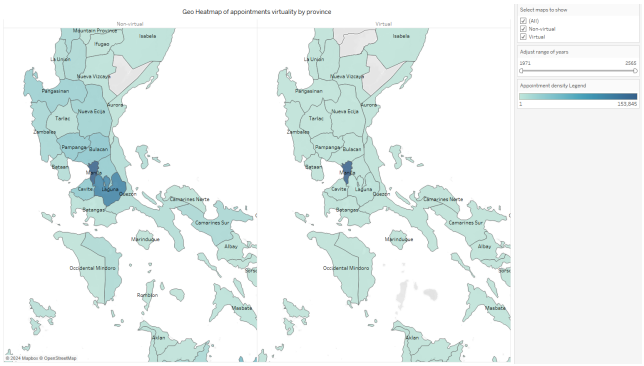

Figure 4: Geo heatmap of virtual and non-virtual appointments by province

In Figure 3, the visualization highlights a notable disparity in appointment types across various provinces, with non-virtual appointments prevailing in many regions, particularly concentrated in Manila and Laguna. This concentration suggests a higher demand or accessibility for in-person medical consultations in these areas. Conversely, virtual appointments appear to be notably dense primarily in Manila, while other regions show a comparatively lower frequency of virtual consultations. This disparity in virtual appointment distribution may reflect varying levels of technological infrastructure, healthcare accessibility, or patient preferences across different regions.

The corresponding SQL statement for the analytical report is as follows:

```
SELECT
    YEAR(a.queuedate),
    c.province,
    a.isVirtual,
    COUNT(*) AS total_appointments
FROM
    appointment a
        JOIN
    Clinic c ON a.clinicid = c.clinicid
    GROUP BY YEAR(a.queuedate), province,
        isVirtual
```

The utilization of aggregate functions, particularly the COUNT() function, plays a pivotal role in summarizing appointment data within the context of the provided query. By employing COUNT(), the query efficiently calculates the total number of appointments, taking into account various dimensions such as year, province, and appointment type (virtual or non-virtual). With the GROUP BY clause, the query organizes the data into distinct groups determined by the specified columns, enabling the calculation of appointment counts for each unique combination of these attributes. This function aggregates data based on these specified columns, and further enhances the analysis by grouping appointments according to their respective attributes.

The query effectively dices the data by year, province, and isVirtual, providing aggregated appointment counts for each combination. The report analyzes data based on dimensions such as province, isVirtual, and years. Users have the ability to specify parameters including range of years and isVirtual, which they can select to display in the report. Slice and dice operations can be used in analyzing a subset of data by filtering maps and applying conditions on appointments in a specific range of years.

4.3 Heatmap for Virtual Appointment Utilization

The report displays the number of virtual appointments, non-virtual appointments, and total appointments for all clinics or those in a selected province. This information can help users understand the utilization of virtual appointment services across different clinics and provinces. It allows for a comparison of virtual appointment adoption rates between clinics and regions, highlighting areas of success and areas for improvement. The report offers insights into resource allocation guiding efforts to optimize service delivery. This analytical tool serves as a valuable resource for healthcare administrators, policymakers, and clinicians seeking to leverage telemedicine effectively and address evolving patient needs.

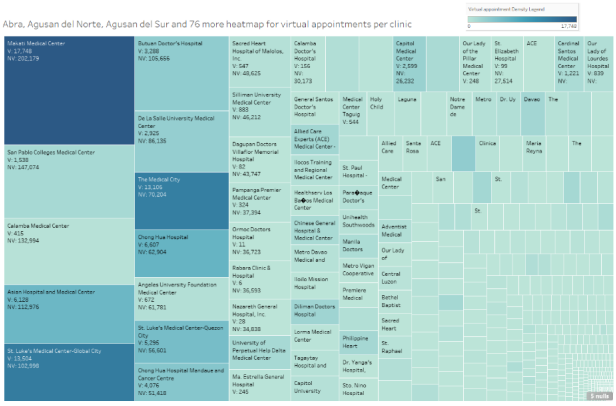

Figure 5: Heatmap for virtual appointments per clinic in all provinces

In Figure 4, the data highlights Makati Medical Center as the institution with the highest total number of appointments. Specifically, Makati Medical Center records 17,748 virtual appointments, indicating a substantial need on remote medical consultations. Other prominent hospitals such as Medical City, St. Luke's Medical Center, and Asian Hospital and Medical Center also have notable densities of virtual appointments, with 13,106, 13,504, and 6,128 appointments, respectively. In contrast, smaller clinics and medical facilities exhibit lower densities of virtual appointments, typically around 2,000 or less. This distribution underscores the varying

adoption rates of virtual healthcare services among different healthcare providers, with larger institutions like Makati Medical Center leading the way in embracing telemedicine practices. This may be attributed to its technological infrastructure, reputation, and patient outreach efforts.

The corresponding SQL statement for the analytical report is as follows:

```
SELECT
    c.hospitalname AS clinic,
    province,
    SUM(CASE
        WHEN a.isVirtual = 1 THEN 1
        ELSE 0
    END) AS virtual_appointments,
    SUM(CASE
        WHEN a.isVirtual = 0 THEN 1
        ELSE 0
    END) AS non_virtual_appointments,
    COUNT(*) AS total_appointments
FROM
    appointment a
        INNER JOIN
    Clinic c ON a.clinicid = c.clinicid
GROUP BY province, clinic
```

In the SQL query, CASE statements are employed to categorize appointments into virtual and non-virtual based on the value of the 'isVirtual' flag. The SUM function is then utilized within these CASE statements to aggregate the counts of virtual and non-virtual appointments separately. This allows for a comprehensive analysis of appointment types within the dataset. Additionally, the COUNT function is utilized to calculate the total number of appointments, providing an overall perspective on the appointment volume. Finally, the GROUP BY clause is used to organize the results by province and clinic, enabling the aggregation of appointment counts for each clinic within each province.

The query efficiently dices the data based on province, clinic, and isVirtual, presenting summarized appointment counts for each unique combination. Additionally, the query implicitly performs a pivot operation by presenting counts of virtual and non-virtual appointments as separate columns, facilitating comparison of appointment types across clinics within each province.

It analyzes key dimensions including province, clinic, and isVirtual. The analytical report employs the slice operation by allowing users to filter results based on specific provinces, enabling focused analysis on appointment data within those regions. Users can further slice the data by selecting specific provinces with the geographic map above.

4.4 Stacked Bar Graph for Healthcare Demographics and Utilization Patterns

The analytical report showcasing a stacked bar graph of patient counts distributed by gender across various age groups within a chosen year offers valuable insights into healthcare demographics and utilization patterns. By visually representing the gender

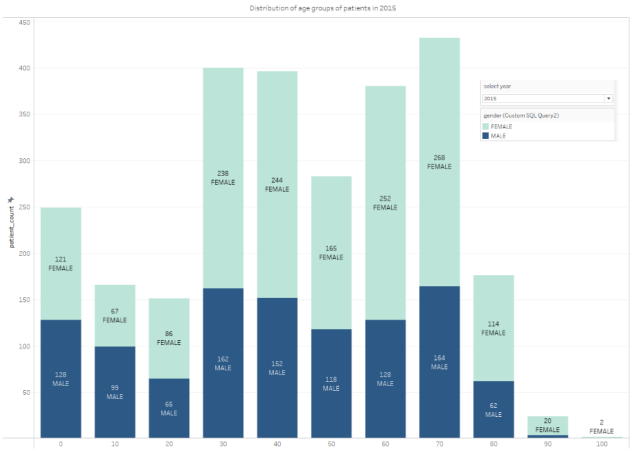

Figure 6: Gender distribution across age groups in 2015

distribution across different age brackets, the report enables quick identification of demographic trends and disparities. Analyzing the data facilitates understanding of healthcare needs among different demographics, aiding in resource allocation and targeted intervention development. Moreover, tracking changes over time allows for monitoring of healthcare utilization patterns and informs policy decisions regarding resource allocation and infrastructure development. The report serves as a benchmarking tool for evaluating organizational performance and guiding policy formulation to address the specific healthcare needs of diverse demographic groups effectively.

In Figure 5, age groups spanning from 30 to 70 years old in 2015 exhibit the highest number of appointments, indicating a significant demand for medical services within this demographic range. Among these age groups, female patients show a notable increase in appointments compared to male patients. This trend suggests that females in the 30 to 70 age brackets are more proactive in seeking healthcare services or may have specific health concerns requiring frequent medical attention for the year 2015. Possible factors contributing to this gender disparity in appointment rates could include differences in healthcare-seeking behavior, prevalence of specific health conditions, or increased awareness and utilization of healthcare services among female patients. Healthcare providers may take note of these demographic trends to ensure that their services are tailored to effectively meet the diverse needs of their patient population.

The corresponding SQL statement for the analytical report is as follows:

```
SELECT
    YEAR(a.queuedate),
    gender,
    FLOOR(p.age / 10) * 10 AS age_group,
    COUNT(*) AS patient_count
FROM
    appointment a
        JOIN
```

```
    px p ON a.pxid = p.pxid
GROUP BY YEAR(a.queuedate) , age_group , gender
ORDER BY age_group;
```

In the provided SQL query, the FLOOR function is utilized to categorize patient ages into brackets by rounding down to the nearest multiple of 10, facilitating the grouping of patients into age cohorts (e.g., 0-9, 10-19, 20-29, etc.). The GROUP BY clause is then employed to organize the data based on the year of the appointment, the age group, and the gender, enabling the aggregation of patient counts within each distinct group. Within these grouped sets, the COUNT(*) function calculates the total number of patients, providing a summarized view of patient demographics across different age groups and genders within the selected year.

In the provided SQL query, the grouping of age into brackets using the FLOOR function enables a roll-up operation, summarizing detailed patient age data into broader age groups (e.g., 0-9, 10-19). Furthermore, the query utilizes a dice operation by dicing the data based on gender and year, resulting in separate counts for male and female patients within each age group for the selected year.

The report examines data across dimensions including year, age group, and gender. Users have the option to select parameters specifying all or a specific year to display in the analysis. The data can be sliced further by showing patient counts in each gender across different age groups in a selected year.

5 QUERY PROCESSING AND OPTIMIZATION

Query optimization is the iterative process of enhancing the performance or efficiency of queries in terms of some relevant cost measuring criteria. This is done first and foremost to maximize the utilization of resources such as CPU and memory. This reduces cost in terms of required computational effort or need system resources, which can enable organizations to more effectively handle data assets or large databases to pursue business or academic-related purposes.

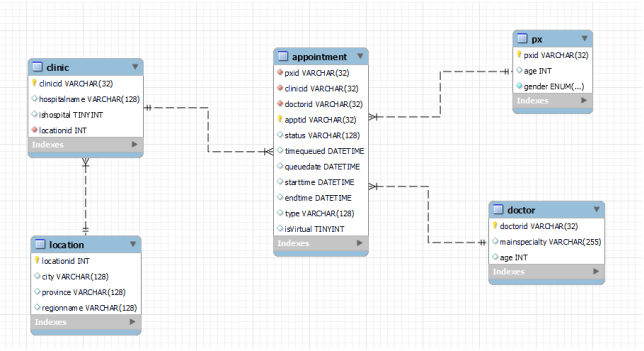

Figure 7: Database Schema with Normalized Table

As part of the optimization strategies that were explored and tested with the proposed SQL queries, the original schema was modified. Specifically, the clinic table was normalized to establish a snowflake schema as seen in Figure 6.

The following SQL queries were used to perform the normalization:

Listing 1: SQL Normalization Script

```
CREATE TABLE location (
    locationid INT AUTO_INCREMENT PRIMARY KEY,
    city VARCHAR(128),
    province VARCHAR(128),
    regionname VARCHAR(128),
    UNIQUE KEY (city, province, regionname)
);

INSERT INTO location (city, province, regionname)
SELECT city, province, regionname
FROM
    (SELECT DISTINCT
        city, province, regionname
     FROM clinic) AS uniquelocations;

ALTER TABLE clinic ADD locationid INT NULL;

UPDATE clinic AS c
JOIN     location AS l ON
         c.city = l.city AND
         c.province = l.province AND
         c.regionname = l.regionname
SET c.locationid = l.locationid;

ALTER TABLE clinic
DROP COLUMN city,
DROP COLUMN province,
DROP COLUMN regionname;
```

The unique values for city, province, and regionname in the clinic table are taken and placed into a new location table with a corresponding location id. A new 'locationid' column is then placed into the clinic table having the matching values with the location table inserted based on city, province, and regionname. These columns are dropped thereafter.

For each query, simple and composite secondary indexes will be formulated based on the GROUP BY columns wherever applicable, as recommended by the MySQL 8.0 Reference Manual[7].

On a hardware level, buffering was managed through the `innodb_buffer_pool_size` configuration variable in MySQL Workbench.

6 RESULTS AND ANALYSIS

To evaluate the OLAP application, functional and performance testings were done. Functional testing ensures the correctness of the reports generated while performance testing assures satisfactory report generation speeds [6]. Both of these tests allow end users to correctly and conveniently gain insights which may aid in business decisions and operations.

6.1 Functional Testing

Functional testing evaluates the accuracy of the reports generated by an application. The verification of the correctness of the ETL

process was done through checking of data types after each data transformation in Python while it is done through proper schema definition in the Apache NiFi pipeline.

To validate the correctness of the OLAP operations, another database was created having the subset of data in the data warehouse. To be specific, only 1000 appointments are present in this database; however, to prevent missing records referenced in these appointments, all records in all dimension tables are included. After the formation of the database, multiple test queries were identified, executed, and validated. The result of these test queries are verified through manual checking of a randomly-picked record present in the result. This verification involves the comparison of primary and foreign keys and the comparison of the number of rows returned with the expected outcome.

Listing 2: Filter records based on a value within the same table

```
SELECT *
FROM px_test
WHERE age>40
```

Listing 3: Merge data from multiple tables based on the primary and foreign keys of the fact and dimension tables respectively.

```
SELECT *
FROM appointment_test a, px_test p,
     clinic_test c, doctor_test d
WHERE a.pxid = p.pxid
AND a.clinicid = c.clinicid
AND a.doctorid = d.doctorid
```

Listing 4: Aggregate data in a table

```
SELECT mainspecialty, COUNT(mainspecialty)
FROM doctor_test
GROUP BY mainspecialty
```

6.2 Performance Testing

Hardware specifications for the machine used for performance testing:

- **CPU**: Ryzen 5 2600 3.4 GHz, 6 cores 12 threads
- **Memory**: 16 GB DDR4 3200 MHz

Before being able to individually test the effectiveness of each of the optimization strategies that were previously proposed, the buffer size was first increased through executing the following SQL Statement:

```
SET GLOBAL
innodb_buffer_pool_size = 4294967296;
```

This was the equivalent of 4GB of memory dedicated to running the queries, when by default, this value is set to 134217728 which is the equivalent of 128 MB.

This was done in order to be able to more reasonably compare the performance of the queries. With the smaller buffer pool size, queries ran for impractically long durations of time. It took over 20 minutes for example to run a single query on 128 MB for buffer pool size, making it infeasible to record multiple executions, over multiple queries, over multiple possible optimization strategies, which lead to the necessity of this change.

Following this, the database design as an optimization strategy was tested. The first and second analytical reports were both run five times each for both the snowflake-schema and star-schema versions of the database after which the running times would be compared. In order to accommodate the new structure of the database, the second SQL query was modified:

Listing 5: SQL Query 2 Modified for Snowflake Schema

```
SELECT   YEAR(a.queuedate),
         l.province,
         a.isVirtual,
         COUNT(*) AS total_appointments
FROM appointment a
JOIN Clinic c    ON a.clinicid=c.clinicid
JOIN Location l ON c.locationid=l.locationid
GROUP BY     YEAR(a.queuedate),
             l.province,
             a.isVirtual;
```

The following table presents the recorded execution times for the first SQL Query:

Table 1: Time Trials for Database Design (First Iteration)

Star Schema		Snowflake Schema	
Query 1	Query 2	Query 1	Query 2
85.797 sec	61.390 sec	71.000 sec	112.594 sec
41.453 sec	45.640 sec	40.515 sec	46.719 sec
42.156 sec	45.672 sec	39.578 sec	46.015 sec
42.156 sec	46.015 sec	39.297 sec	46.797 sec
42.610 sec	45.782 sec	39.735 sec	46.500 sec

Clearly, the execution time for every query seems to spike at its first run-through. It is only from the second execution of the same query onwards that the execution times stay consistent. One likely reason for this is that from the second execution onwards, the data being requested by the SQL query is already stored in the memory, bypassing the need to read from the disk, thus speeding up the execution time[8].

The time trials were redone with the SQL Queries being executed in an alternating order, rather than sequentially. This was done in order to observe a possible difference in execution time.

Table 2: Time Trials for Database Design (Second Iteration)

Star Schema		Snowflake Schema	
Query 1	Query 2	Query 1	Query 2
105.765 sec	102.953 sec	115.594 sec	114.172 sec
91.485 sec	104.313 sec	111.204 sec	116.375 sec
91.313 sec	104.590 sec	111.906 sec	112.254 sec
92.875 sec	102.315 sec	114.016 sec	113.797 sec
90.50 sec	101.785 sec	109.890 sec	114.531 sec

On average, there was an increase of 18.114 seconds in execution time for the first query and 11.033 seconds for the second query when being run on the snowflake schema compared to when it was executed on the star schema. When taking the average for both queries, there is a total average increase of 14.558 seconds for execution time on the snowflake schema.

This increase in execution time could be explained by the redundancy present in denormalized tables. This redundancy decreases the need for and join operations and therefore speeds up the overall execution time for any queries[9].

The next optimization strategy that will be explored will be the usage of indexes.

Primary indexes are automatically generated upon creation of a table with a primary key. To test its effectiveness for query performance, the execution time was recorded for queries with and without using primary indexes. The SQL queries are modified to indicate that no indexes will be used[10]

Listing 6: SQL Query 1 Using No Indexes

```
SELECT
    d.mainspecialty,
    YEAR(a.queuedate) AS appointment_year,
    QUARTER(a.queuedate) AS appointment_quarter,
    MONTH(a.queuedate) AS appointment_month,
    COUNT(mainspecialty) AS total_appointments
FROM
        appointment a
USE INDEX ()
JOIN
        Doctor d ON a.doctorid = d.doctorid
GROUP BY
        d.mainspecialty,
        YEAR(a.queuedate),
        QUARTER(a.queuedate),
        MONTH(a.queuedate) with rollup;
```

With this, the execution times are recorded.

Table 3: Time Trials for Primary Indexes on SQL Query 1

With Primary Indexes	Without Primary Indexes
39.266 sec	59.672 sec
39.891 sec	62.766 sec
41.922 sec	60.516 sec
39.687 sec	60.672 sec
39.891 sec	61.446 sec

There is an around 20 second increase in execution time for query 1 when executing without a primary index as seen in Table 2. This could be attributed to the ability of primary indexes to provide quick paths to needed data [11].

Moving on to secondary indexes, these are implemented based on the GROUP BY clauses in the SQL statement based on the MySQL 8.0 Reference Manual[7]. The indexes are instantiated using CREATE INDEX command as specified below:

Listing 7: Creation of Secondary Indexes for Query 1

```
CREATE INDEX idx_mainspecialty
    ON doctor(mainspecialty);
CREATE INDEX idx_queuedate
    ON appointment(queuedate);
```

As each query is tested, secondary indexes that were created for previous trials are dropped so as to not create any unintended effects on time as shown in the code below:

Listing 8: Dropping Secondary Indexes Created for Query 1

```
DROP INDEX idx_mainspecialty ON doctor;
DROP INDEX idx_queuedate ON appointment;
```

Whenever possible, composite indexes are also created as shown below for Query 2:

Listing 9: Creation of Composite Index on Appointment for Query 2

```
CREATE INDEX idx_appointment
    ON appointment(isVirtual, queuedate);
CREATE INDEX idx_province
    ON clinic(province);
```

Table 4: Time Trials for Secondary Indexes on SQL Query 1

With Secondary Indexes	Without Secondary Indexes
38.375 sec	39.328 sec
38.703 sec	38.484 sec
39.047 sec	39.593 sec
38.563 sec	39.703 sec
38.672 sec	39.203 sec

38,672 39,062

The difference in execution times with the inclusion of secondary indexes was very marginal especially in comparison to the effect that the primary indexes had, with an average of 0.39 seconds time saved using secondary indexes. While the values are indeed similar, the fact that the top 4 longest records taken from table 4 are all from the execution of the query without using secondary indexes could suggest that there is still a real advantage to using the indexes.

Table 5: Time Trials for Secondary Indexes on SQL Query 2

With Secondary Indexes	Without Secondary Indexes
43.297 sec	44.250 sec
43.359 sec	43.641 sec
43.172 sec	43.313 sec
43.172 sec	43.609 sec
43.765 sec	44.297 sec

Similar to the trials conducted on the first query, secondary indexes seem to also only have a marginal positive effect on performance. In this case, even with the usage of composite indexes, there was only a 0.469 second average decrease in execution time.

In another trial, we see the effect of having unnecessary indexes. Indexes are created for every single column in the database, with the execution time recorded running the third query.

Table 6: Time Trials Using All Possible Secondary Indexes on SQL Query 3

With All Secondary Indexes	Without Secondary Indexes
51.844 sec	48.391 sec
51.297 sec	48.531 sec
49.985 sec	48.578 sec
51.328 sec	48.688 sec
51.000 sec	50.110 sec

For this trial, the additional indexes appear to hamper performance. The indexes added an average of 2.231 seconds for execution time.

7 CONCLUSION

This paper demonstrates the process of the creation of a data warehouse, the utilization of an ETL tool to populate said data warehouse, the development of an OLAP dashboard, and the evaluation and optimization of queries to provide insights that may prove useful in business operations.

Through the completion of this project, various insights regarding the development of OLAP applications were gathered. Firstly, proper construction of a data warehouse starts with examination of data as knowing values present in the data will allow proper data preprocessing which will lead to a more appropriate database schema formation and facilitation of OLAP operations. In terms of OLAP and data visualization and dashboards, the members' experience with these concepts and tools displayed its capabilities to present valuable business insights and trends useful to organizations different to OLTP operations the members are more familiar with. Lastly, query optimization such as the use of indices, and different schema types and hardware configuration has shown its possible value in managing large amounts of data where query times become significantly longer.

REFERENCES

[1] Oracle. 2020. What Is a Data Warehouse?. Retrieved from https://www.oracle.com/ph/database/what-is-a-data-warehouse/

[2] Amazon Web Services. What is OLAP (Online Analytical Processing)?. Retrieved from https://aws.amazon.com/what-is/olap/

[3] Amazon Web Services. What is ETL (Extract Transform Load)?. Retrieved from https://aws.amazon.com/what-is/etl/

[4] pandas. pandas. Retrieved from https://pandas.pydata.org/

[5] The Apache Software Foundation. Apache NiFi. Retrieved from https://nifi.apache.org/

[6] A. Liudmila. 2023. The Importance of Database Testing for Functional Testers. Retrieved from https://www.linkedin.com/pulse/importance-database-testing-functional-testers-liudmila-akulovich

[7] "MySQL :: MySQL 8.0 Reference Manual :: 8.2.1.17 GROUP BY Optimization," dev.mysql.com. https://dev.mysql.com/doc/refman/8.0/en/group-by-optimization.html

[8] "MySQL Performance: InnoDB Buffers & Directives," Liquid Web, Dec. 18, 2018. https://www.liquidweb.com/kb/mysql-performance-innodb-buffers-directives/#:~:text=InnoDB%20Buffer%20Pool%20(IBP)&text=The%20more%20InnoDB%20tablespace%20that

[9] "When and How You Should Denormalize a Relational Database," rubygarage.org. https://rubygarage.org/blog/database-denormalization-with-examples

[10] "MySQL :: MySQL 8.0 Reference Manual :: 10.9.4 Index Hints," dev.mysql.com. https://dev.mysql.com/doc/refman/8.0/en/index-hints.html

[11] R. Gupta, "SQL Server indexes: Key requirements, performance impacts and considerations," The Quest Blog, Jul. 23, 2020. https://blog.quest.com/sql-server-indexes-key-requirements-performance-impacts-and-considerations/

[12] Indratmo, L. Howorko, J. M. Boedianto, and B. K. Daniel, "The efficacy of stacked bar charts in supporting single-attribute and overall-attribute comparisons," *Visual Informatics*, vol. 2, no. 3, pp. 155–165, Sep. 2018, doi:10.1016/j.visinf.2018.09.002

[13] H. Cheng and H. Cheng, "The benefits of using heatmaps to boost conversion rates," *Freedom to Ascend*, Jun. 21, 2023. https://www.freedomtoascend.com/marketing/marketing-tactics/conversion-rate-optimization/heatmaps/

8 DECLARATIONS

8.1 Declaration of Generative AI in Scientific Writing

During the preparation of this work the author(s) used GPT-3 in order to rephrase ideas to convey them more effectively. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

8.2 Record of Contribution

David Joshua Corpuz - Developed Dashboard. Wrote abstract and Section 4 (OLAP Application)

Rafael Dimagiba - Query Optimization and Performance Testing. Wrote Section 6.2 (Performance Testing) and Section 7 (Conclusion)

Steven Errol Escopete - ETL and Functional Testing. Wrote Section 1, 2, 6, 6.1 and 7

Ralph Pineda - Data Warehousing and ETL. Wrote Section 3 (ETL Script)