An Evaluation of Clinic Performance: Utilizing OLAP Applications for Appointment Analysis

Jose Raphael E. Martinez[1], Jaeme Patrice O. Rebano[2], Wesly F. Samson[3] and Miguel R. Villanueva[4]

College of Computer Studies, De La Salle University

[1]jose_raphael_martinez@dlsu.edu.ph, [2]jaeme_patrice_rebano@dlsu.edu.ph, [3]wesly_samson@dlsu.edu.ph, [4]miguel_r_villanueva@dlsu.edu.ph

ABSTRACT

SeriousMD records contain large-sized datasets that may answer several analytical questions regarding the statistics of clinics in the Philippines. In this paper, it was used for a data warehouse that was created using a star schema and populated using ETL. A dashboard application using different OLAP applications such as Roll-Up, Drill-Down, Slice, and Dice was also created which seeks to answer several analytical reports. Results show that optimization using query restructuring and indexing significantly reduced execution times by up to 17 seconds.

Keywords

Performance, ETL, OLAP, Query Optimization.

1. Introduction

Data management has been a longstanding challenge in the health industry, especially with the increasing volume of data generated by technology. To store, process, and analyze this data effectively, the health industry requires robust systems that can support business intelligence and decision-making. A data warehouse is one such system that facilitates comprehensive data analysis and reporting by integrating data from multiple sources and enabling Online Analytical Processing (OLAP) operations and optimized Query Processing. This paper aims to provide a detailed study and implementation of a data warehouse system, focusing on OLAP operations and Query Processing techniques.

The dataset used in this paper is a collection of records containing anonymous details regarding appointments, clinics, doctors, and patients from SeriousMD, an online medical consultation platform.

For this report, a star schema was used to create the data warehouse with Appointments being the fact table, while Clinics, Doctors, and Patients were dimension tables. This allows for simpler retrieval and analysis of data along with faster query performance [3].

The dataset underwent a thorough cleaning process using Jupyter Notebook, which allowed for the correction of any formatting or missing values in the tables. Apache NiFi was then used to populate the data warehouse.

To better understand the data, an OLAP application was created using Tableau that allows users to visualize the data warehouse from different angles. Four OLAP operations were used in the application to aid in understanding, namely Roll-up, Drill-down, Slice, and Dice. The paper also delves into query optimization techniques to help improve the performance of the application.

Fellow researchers and medical practitioners can use this application to further analyze trends or patterns on when and where services could be more accessible and available to patients.

2. Data Warehouse

The Data Warehouse [13] is a system that supports analytic activities. It is intended for the use of applying queries and analytical reports for large amounts of data. It provides a relational database that incorporates an Extraction, Transformation, and Loading (ETL) solution to prepare the data for analytical reports. This system is important for the application since the datasets used from SeriousMD are large, containing 9.7 million data rows from the appointments alone. Having a data warehouse is beneficial in performing the required queries since it allows the needed OLAP operations to acquire the data at varying levels of detail.

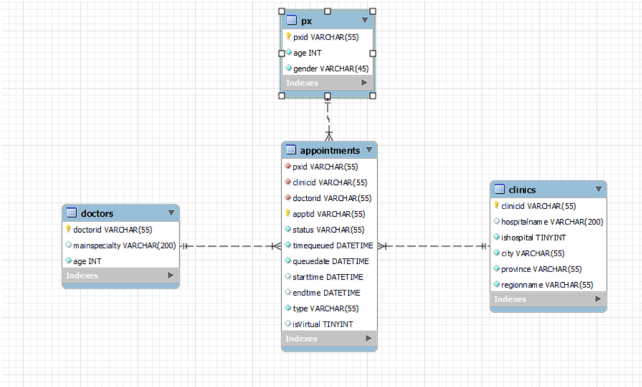

Figure 1. The Dimensional Model using Star Schema

As seen in Figure 1, the dimensional model makes use of a star schema. In a star schema, there is a central fact table surrounded by dimension tables, and each dimension table is directly connected to the fact table through foreign key relationships. The fact table is "appointments" which contains tuples of recorded factual data regarding appointments made. It models the occurrences of appointments, capturing essential metrics and attributes associated with each appointment instance.

The dimensions are clinics, px (patients), and doctors. The hierarchy of the clinic's dimensions is region, province, city, and hospital name. This dimension is helpful for different OLAP operations such as the roll-up and drill-down, especially with attributes concerning the geographical variables. .For the px dimension, it is gender, age. Lastly, the hierarchy for doctors is the main specialty, age. These different attributes from the two tables can be used for different OLAP operations such as slice and dice

which helps in determining the appointment count under different parameters.

The attributes with boolean values in the dataset were initially set to TINYINT in the schema due to MySQL not having a boolean type. After preprocessing, these attributes contained 1 or 0. Although, a data type mismatch occurred when trying to insert the data. To solve this error, the data type was changed to INT which encountered no problems after the change.

3. ETL Script

Extract-Transform-Load (ETL) is a systematic process that is crucial in various industries, organizations, and sectors that benefit from effective data preprocessing, integration, and analysis for better decision-making, insights, opportunities, and risk minimizations. In essence, it involves extracting data from multiple sources, transforming them according to specific requirements, and loading them into desired data destinations. In this project's scenario, the ETL process will be utilized for extracting datasets from the health sector, transforming them, improving their data quality, and generating effective and timely reports that will benefit the overall organization and its beneficiaries (medical practitioners, researchers, and patients). To perform the ETL process in this project, the group decided to utilize the Jupyter Notebook (along with Python's Pandas library for data preprocessing) for the extract and transform process. Jupyter's interactive environment and reproducibility [4], along with the group's prior knowledge of the strengths of Python's Pandas library, were the main factors considered by the group in utilizing the ETL prerequisites. For the process of data loading into the data warehouse, Apache NiFi was used for its user-friendly interface and powerful features in managing data flows [2].

The **extraction** process was first performed by using straightforward Pandas functions (along with Jupyter's interface) such as reading CSVs (*read_csv()*) to extract data from the four source CSV datasets into Pandas' data frames, which serve as the library's main data structure. This approach enabled the group to effectively read and understand the datasets, which resolved the group's first encountered issue of being unable to access and view the original CSVs via Microsoft Excel, etc. due to the CSVs' overwhelming volume of data/records.

For the **transformation** process, the following issues needed to be resolved/transformed via data preprocessing and transformation techniques in Pandas: multiple representations/irrelevant values for doctor specialties, an overwhelming number of appointments with patients/doctors/clinics that can't be found on other source datasets and vice versa, conversion of Boolean values into integer type for MySQL loading compatibility, and NaN/extreme values for age columns that may affect data analysis/loading. The group decided to address the overwhelming number of records in the datasets (especially in appointments) first since the issue would negatively impact later preprocessing steps and loading processes into the data warehouse. In addition, the group also made an assumption wherein doctors, clinics, and patients that are random, non-existing, or may contain irrelevant data values could be reduced if the doctors/patients/clinics not found on the appointments altogether and doctors/patients/clinics from appointments that were not found on their respective datasets were also trimmed.

To solve the first issue, the group merged (equivalent to an inner join based on a column equality) each of the patients, clinics, and doctors datasets with the uncleaned appointments dataset. This was to ensure that all doctors, clinics, and patients that are not found on the appointments based on their IDs will be dropped since they would have no contribution to the fact table and the overall report generation in the application. Afterward, each of the patients, doctors, and clinics datasets were preprocessed. For the patients dataset, NaN/extreme age values were fixed/interpolated according to overall average. For the clinics dataset, only verifications were made since there weren't any issues with the columns and their values. Finally, for the doctor dataset, the majority of the preprocessing revolved around correcting multiple representations and irrelevant data for the doctor specialties. The replace function of the data frames in Pandas was used to replace irrelevant doctor specialties into blank whereas doctor specialties with different spellings/formats that refer to the same specialty were generalized. However, the approach was inconvenient or brute force due to the significant number of inconsistent doctor specialties.

Correcting Doctor Specialties

```
In [22]: doctors_copy_df['mainspecialty'] = doctors_copy_df['mainspecialty'].replace(
             {
             'Radiation Oncology / Otorhinolaryngology': 'Radiation Oncology, Otorhinolaryngology',
             'Radiology family medicine': 'Radiology, Family Medicine',
             'Radiology/ General Physician': 'Radiology, General Physician',
             'Rehab': 'Rehabilitation',
             'Rehabilation Medicine,': 'Rehabilitation Medicine',
             'Rehabilitation  Medicine': 'Rehabilitation Medicine',
             'Rehabilitation & Physical Therapy': 'Rehabilitation, Physical Therapy',
             'Rehabilitation medicine': 'Rehabilitation Medicine',
             'Rehabilitation Medicine ': 'Rehabilitation Medicine',
             'Rehabilitation Therapiat': 'Rehabilitation Therapist',
             'Reproductive Endocrinology and Infertility ; Gynecologic Endoscopy': 'Reproductive Endocr
             'Restorative Medicine': 'Regenerative Medicine',
             'Rheumatologist': 'Rheumatology',
             'Rheumatology - Internal Medicine': 'Rheumatology, Internal Medicine',
             'Rheumatology and Autoimmune Diseases': 'Rheumatology, Autoimmune Diseases',
             'RN/A': ' ',
             'RURAL HEALTH PHYSICIAN': 'Rural Health Physician',
             's': ' ',
             'Secretary': ' ',
             'Skin Health and Aesthetic Dermatology': 'Aesthetic Dermatology, Skin Health',
             'Sleep Medicine/Sleep Studies (Polysomnographer)': 'Sleep Medicine, Sleep Studies (Polysom
             'Sleep Specialist': 'Sleep Medicine, Sleep Specialist',
             'small animal medicine': 'Veterinary Medicine, Small Animal',
             'Small Animal Practice': 'Veterinary Medicine, Small Animal',
             'SolSee': ' ',
```

Figure 2. Correcting Doctor Specialties Using Replace Function in Pandas

Finally, the group merged the appointments dataset to each of the 3 dimension datasets (patients, doctors, and clinics), which reduced the number of records in the appointments significantly from 9.7 million to 320,140 records. This approach ensured that all the appointments referred to existing patients, doctors, and clinics and vice versa, which provided data integrity and foreign keys relationship compatibility between the fact and dimension tables/datasets.Further minimal cleaning was also implemented by converting boolean values into integer type and fixing some null values for the consultation type.

For the **loading** process, the data was first imported into a MySQL database that would act as our source database. From there, the QueryDatabaseTable process was used along with the Database Connection Pooling Service to connect to the database and get the values of the tables from the source schema. Finally, the PutDatabaseRecord process along with AvroReader and another Database Connection Pooling Service was used to connect to the warehouse and copy the contents of the source database onto it. The use of JoltTransform processors was not needed since all of the needed transformations were already done through the Pandas library in Jupyter. No problems were encountered with foreign key constraints during the loading as the fact table was loaded last and merging the datasets (inner joins) with one another during the transformation process ensured the appointments

dataset's compatibility in establishing foreign key relationships with other remaining datasets.

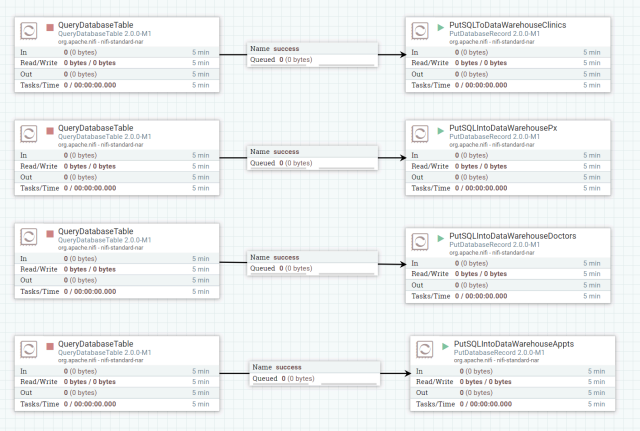

Figure 3. Apache NiFi Processors for Loading Data into the Data Warehouse

4. OLAP Application

According to an article by cleverism [10], OLAP (Online Analytical Processing) is essentially a tool used for analyzing data from all sorts of perspectives and situations. Additionally, IBM [6] stated that the tool is a software for multidimensional analysis at high speeds.

In the application, OLAP operations are utilized to analyze clinic performance based on various appointment factors and granularities. The application provides detailed visualization of analytical situations to compare trends and determine factors affecting healthcare performance.

The application has the goal to be able to generate reports that correlate with the number of appointments by comparing the amount from different factors. The application is meant to be consistent with the data being outputted with the consistent variable being the count of appointments and applying OLAP operations to have deeper insights on how different variables may affect the count. These insights help clinics analyze performance at a granular level, benefiting in deciding which specific attributes should be further improved on. Furthermore, the application provides deeper analytical insights to assist clinics in formulating personnel or resource allocation decisions.

The charts used include bar charts (stacked, horizontal, side-by-side), pie charts and a map visualization chart. Figure 4 uses a horizontal bar chart making it effective for comparing values across the regions, allowing users to easily identify which regions have the highest or lowest number of appointments [9]. Additionally, the orientation of the chart eases the ability to read long category labels. A map chart is efficient as it can show geographical distribution. As seen in Figure 5, it is easier to identify the appointment trends across different provinces within these regions [8]. Pie charts are suitable for showing proportions or percentages of a whole [16]. Figure 6 displays the distribution of appointment statuses within the specified city. A stacked bar chart is used in Figure 7 to view how different specialities contribute to the number of appointments in the region, providing insights into specialty demand. Using the stacked bar chart is helpful in representing the totals [7]. Lastly, the vertical bar chart was implemented which is essentially the same with the horizontal bar chart. In Figure 8, the chart was used to help in the comparisons of the number of appointments for each available region that has a non-null value of count for a specific year. It would allow healthcare analysts to have a regional comparison for a specific time period.

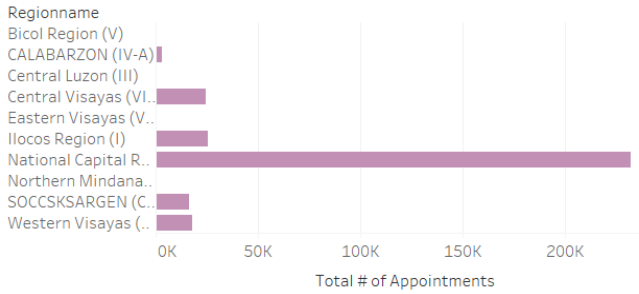

Figure 4. Horizontal Bar Chart of Total # of Appointments per region

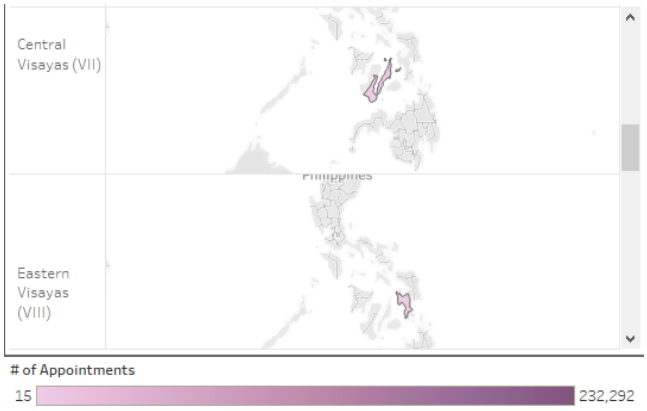

Figure 5. Map Chart of Total # of Appointments per province in Central Visayas (VII) and Eastern Visayas (VIII)

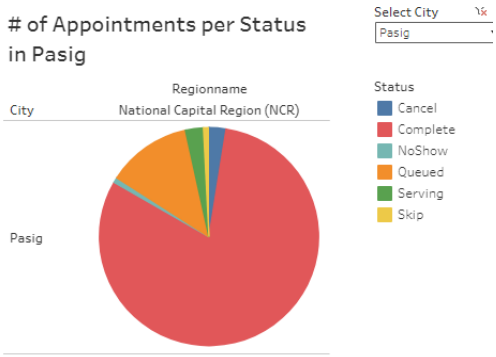

Figure 6. Pie Chart of Total # of Appointments per status in Pasig

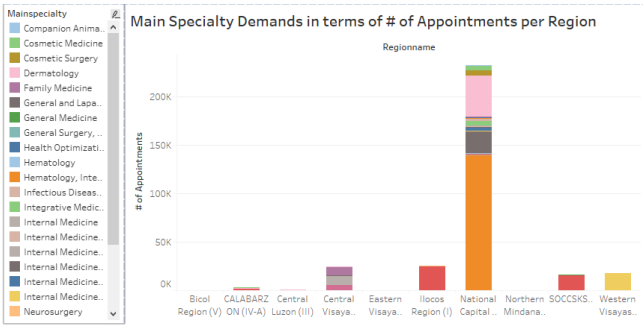

Figure 7. Stacked Bar Chart of Main Specialty Demands in terms of # of Appointments

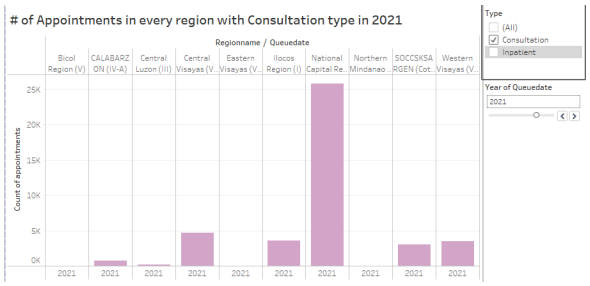

Figure 8. Vertical Bar Chart of Total # of Appointments per region with Consultation type in 2021

The first analytical report aims to find the total number of appointments per region.

Listing 1. SQL Query to Generate Total # of Appointments per region

```
SELECT c.regionname, COUNT(a.apptid) AS `No. of Total Appointments`
FROM appointments a, clinics c
WHERE c.clinicid IN (SELECT (clinicid) FROM appointments
WHERE (a.clinicid = c.clinicid))
GROUP BY c.regionname WITH ROLLUP
ORDER BY c.regionname;
```

The SQL statement, as seen in Listing 1, uses various advanced constructs to achieve this goal. The use of the WHERE statement was used to create a subquery to access the "*clinicid*" value from appointments so that the second WHERE clause serves as the connection between appointments and clinics by their common attribute. The COUNT() function was used in this statement as well to provide the aggregate value to count the number of appointments based on *"apptid"* where this attribute corresponds to one unique instance of an appointment. Additionally, the other advanced constructs include the GROUP BY and ORDER BY functions in which both were used with the intent to arrange the values of appointments by the region.

The OLAP operation ROLL-UP was used in this report due to the data of the number of appointments from the clinics being aggregated to a higher level which is the region the clinics are located in seen in the clinics dimension table.

Similarly with the previous report, the second analytical report intends to find the total number of appointments, but by implementing a different OLAP operation.

Listing 2. SQL Query to Generate Total # of Appointments per province in a region

```
SELECT c.regionname, c.province, COUNT(a.apptid) AS 'No. of Total Appointments'
FROM appointments a, clinics c
WHERE c.clinicid IN (SELECT (clinicid) FROM appointments
WHERE (a.clinicid = c.clinicid))
GROUP BY c.regionname, c.province
ORDER BY c.regionname, c.province;
```

In Listing 2, the same advanced SQL constructs as seen in Listing 1 were utilized, such as the COUNT() for the aggregation of the *"apptid"* attribute, The two WHERE clauses which are used for checking the two attributes and to create a subquery that accesses "*clinicid*" from the appointments table, and GROUP BY/ORDER BY, which were used to arrange the count of appointments based on the region and province. This report may assist healthcare analysts in breaking down the regional performance to a more granular level being within provinces. This would help in the decision-making on how to improve the quality of clinics in terms of their provinces and allocating personnel and resources respectively to the number of appointments.

The second query utilizes the DRILL-DOWN OLAP operation by acquiring the data gathered from different regions of the Philippines and granulating it, allowing for a detailed analysis on the number of appointments. In this scenario, the data is broken down into the various provinces within each region.

The third analysis report seeks to gain the number of appointments based on the status of those appointments per city.

Listing 3. SQL Query to Generate Total # of Appointments per status in a city

```
SELECT c.regionname, c.city, a.status, COUNT(a.apptid) AS `No. of Total Appointments`
FROM appointments a, clinics c
WHERE c.clinicid IN (SELECT (clinicid) FROM appointments
WHERE ((a.clinicid = c.clinicid) AND c.city=<parameter.city>))
GROUP BY c.city, c.regionname, a.status
ORDER BY c.city, c.regionname, a.status;
```

The SQL statement seen in Listing 3 uses various SQL constructs to acquire the amount of appointments per status. The WHERE function was applied twice to connect both the appointments and clinics tables which would enable the access of the city assigned to the appointment along with its current status while also utilizing a subquery. The COUNT() function was used to retrieve the amount of appointments while the GROUP BY/ORDER BY functions remained consistent as well from the previous reports where the output needed to adjust the values for each appointment status based on the cities chosen within a specific region. Lastly the implementation of the WHERE function also allows the SQL statement to filter the data using a specific parameter.

The OLAP operation used is SLICE. This is applied since the analytical report focuses on filtering the data based on the city chosen by the user.

This analytical report showcases the demands of the main specialties of the doctors in terms of the amount of appointments found in a region for each main specialty of the doctors.

Listing 4. SQL Query to Generate Main Specialty Demands in terms of # of Appointments per Region

```
SELECT c.regionname, d.mainspecialty AS `Doctor Specialty`,
COUNT(appt.apptid) AS `No. of Total Appointments`
FROM appointments appt, doctors d, clinics c
WHERE d.doctorid IN (SELECT (appt.doctorid) FROM
appointments WHERE (appt.doctorid = d.doctorid))
AND c.clinicid IN (SELECT (clinicid) FROM appointments
WHERE (appt.clinicid = c.clinicid))
GROUP BY c.regionname, d.mainspecialty WITH ROLLUP
ORDER BY c.regionname, d.mainspecialty;
```

In Listing 4, the SQL statement contains several constructs that were used in the previous reports, such as the COUNT() function, GROUP BY/ORDER BY functions, and the WHERE function. One noticeable difference is the use of two subqueries which serves the purpose of connecting three different tables with each other. The tables of appointments, clinics, and doctors were used with the foreign keys of appointments being the "*clinicid*" and "*doctorid*" and as the primary keys of clinics and doctors, respectively. Another construct applied is the WITH ROLLUP which allows the SQL statement to show the subtotal rows for each specialty and the total number of appointments for a specific region.

The main OLAP operation utilized was the ROLL-UP operation since the report relies on acquiring the number of appointments for each specialty within a region and aggregates the varying subtotal values of each available specialty within the chosen region.

The last analytical report focuses on the number of appointments in every region based on the appointment type within chosen years.

Listing 5. SQL Query to Generate Total # of Appointments per region with specific type in a specific year

```
SELECT c.regionname, YEAR(a.queuedate), COUNT(a.apptid)
AS `No. of Total Appointments`
FROM appointments a, clinics c
WHERE c.clinicid IN (SELECT (clinicid) FROM appointments
WHERE (a.clinicid = c.clinicid AND a.type = <parameter.type>
AND YEAR(a.queuedate) = <parameter.year>))
GROUP BY c.regionname, YEAR(a.queuedate)
ORDER BY c.regionname, YEAR(a.queuedate);
```

The SQL statement in Listing 5 takes advantage of similar constructs such as the COUNT() for the number of appointments, GROUP BY/ORDER BY for arrangement of the values, the WHERE clause which is used to combine the appointments fact table and the clinics dimensional table through a subquery while also being different from all the previous reports. The clause made use of two parameters to perform the OLAP operation needed for the query, the two parameters being the year and the region.

The OLAP operation used in this SQL is DICE. This specific method was applied in order to filter the data by two different parameters which would be the year and appointment type. It offers more insights on which appointment type would have more appointments within a region and it allows healthcare analysts to figure out which appointment type is more in demand based on the total number of appointments within a region.

5. Query Processing and Optimization

Today's dynamic business and industry landscape continues to rely on real-time access to insightful data to keep up with demands and new opportunities [12]. As the landscape grows, databases become much more complex, and the volume of data increases significantly. This expected trend further highlights the importance of utilizing query optimization techniques for real-world applications/software (including this project's OLAP application) to address the beneficiaries' demands more effectively. Essentially, query optimization in a database management system aims to process queries by choosing the most efficient executing plans or strategies [14]. It is crucial for database developers and administrators to learn and apply various and efficient query optimization techniques to improve the performance of applications/software that rely on databases and address the dynamic demands of the business landscape more effectively.

Query optimization strategies in a relational database have four levels:

1. Application Level
2. Database Level
3. DBMS Level
4. Hardware-level

Two optimization techniques were explored in this report, namely, Database level and Application level.

Table 1. Performance of Queries Before Optimization

Q1	Q2	Q3	Q4	Q5
16.440	15.594	3.739	18.808	0.473
16.292	16.274	2.277	17.476	2.309
16.955	16.310	1.178	17.624	0.186
16.549	15.785	1.409	17.660	0.360
16.711	15.739	1.432	17.171	0.182

In Table 1, queries Q1, Q2, and 4 are extremely inefficient, taking over 15 seconds to run with Q4 taking up to 18 seconds. We will apply optimization techniques to try and improve these times.

For the first optimization technique, the group utilized a database-level optimization strategy by establishing foreign key relationships between the fact and dimension tables, further creating primary and secondary indexes. Although before data was loaded into the data warehouse using ETL, optimization by using indices and modifying table structures were already implemented by the group, later discussions will still showcase the performance result of indexing and non-indexing via primary and secondary indexes. Going back to the subject, each table in the data warehouse will have primary indexes (such as appointment ID in the appointments table, patient ID in the patients table, etc.). Furthermore, only the appointments table has secondary indexes (foreign keys) connected to the other tables' primary indexes. By default, MySQL relational databases store data in a balancing tree (BTree) and utilize indexes of BTree type. This approach improves retrieval operations significantly at the expense of cost and data insert time. However, since this project's OLAP application does not involve data insertion, the benefits of BTree indexing outweigh the costs. Indexing is crucial among the tables because all records in the dimensions tables have reference to at least a single record in the fact table (appointments).

Utilizing indexing will enable quick evaluation, filtering, and retrieval of results [11]. BTree indexing is also essential for primary keys as it enables efficient data access performance for queries that involve exact match and range searches [1].

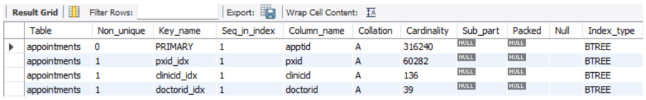
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
appointments	0	PRIMARY	1	apptid	A	316240	NULL	NULL		BTREE
appointments	1	pxid_idx	1	pxid	A	60282	NULL	NULL		BTREE
appointments	1	clinicid_idx	1	clinicid	A	136	NULL	NULL		BTREE
appointments	1	doctorid_idx	1	doctorid	A	39	NULL	NULL		BTREE

Figure 9. Checking The Primary and Secondary Indexes of Appointments Table and Their Index Type

Although searching without indexing is possible (to be showcased later), it will be inefficient since MySQL will have to begin searching each table from the first row and then read through the entire table to find the relevant rows. This operation results in a linear time complexity (O(n)) where n is the number of records in the table. Furthermore, joining tables without indexes (non-indexed nested loop join) is slower and more expensive since it has to examine every pair of tuples in two tables [15], resulting in a time complexity of (O(n*m)) where n and m are the number of tuples/records in table R and S respectively. However, when using indexes in joining tables (index nested-loop join), each search operation has a time complexity of (O(logn)) [5]. For instance, joining tables R and S (with lengths of n and m each) will result in a time complexity of (O(m * logn)). This result is a significant improvement compared to not using indexing.

In terms of the schema design, the group decided to make use of a star schema wherein all the dimension tables are connected to one fact table. This was implemented since the group believed there weren't many significant columns in the dimensions table to denormalize further.

The next technique explored was in the Application level, specifically, query restructuring.

Listing 6. Optimized Q1 Query to Generate Total # of Appointments per region

```
SELECT c.regionname, COUNT(a.apptid) AS `No. of Total Appointments`
FROM appointments a
JOIN clinics c ON a.clinicid=c.clinicid
GROUP BY c.regionname WITH ROLLUP
ORDER BY c.regionname
```

Instead of using a subquery to connect both tables, JOIN was used to utilize the indexes resulting in faster query times.

Listing 7. Optimized Q2 Query to Generate Total # of Appointments per province in a region

```
SELECT c.regionname, c.province, COUNT(a.apptid) AS `No. of Total Appointments`
FROM appointments a
JOIN clinics c ON a.clinicid=c.clinicid
GROUP BY c.regionname, c.province
ORDER BY c.regionname, c.province
```

Similar to Q1, the subquery was replaced with JOIN, drastically improving query performance.

Listing 8. Optimized Q3 Query to Generate Total # of Appointments per status in a city

```
SELECT c.regionname, c.city, a.status, COUNT(a.apptid) AS `No. of Total Appointments`
FROM appointments a
JOIN clinics c ON a.clinicid=c.clinicid
WHERE c.city=<parameter.city>
GROUP BY c.city, c.regionname, a.status
ORDER BY c.city, c.regionname, a.status
```

In this query, the subquery to connect both tables was replaced with JOIN, as well as the WHERE clause of the subquery being moved outside.

Listing 9. Optimized Q4 Query to Generate Main Specialty Demands in terms of # of Appointments per Region

```
SELECT c.regionname, d.mainspecialty AS `Doctor Specialty`,
COUNT(appt.apptid) AS `No. of Total Appointments`
FROM appointments appt
JOIN doctors d ON appt.doctorid = d.doctorid
JOIN clinics c ON appt.clinicid = c.clinicid
GROUP BY c.regionname, d.mainspecialty WITH ROLLUP
ORDER BY c.regionname, d.mainspecialty
```

For Q4, having two subqueries made it the slowest performing query out of them all. Replacing them with JOIN significantly reduced query times and improved performance.

Listing 10. Optimized Q5 Query to Generate Total # of Appointments per region with specific type in a specific year

```
SELECT c.regionname, YEAR(a.queuedate), COUNT(a.apptid)
AS `No. of Total Appointments`
FROM appointments a
JOIN clinics c ON a.clinicid=c.clinicid
WHERE a.type = <parameter.type> AND YEAR(a.queuedate)
= <parameter.year>
GROUP BY c.regionname, YEAR(a.queuedate)
ORDER BY c.regionname, YEAR(a.queuedate)
```

Lastly, subqueries were also replaced in Q5 aiding in better performance. Both conditions for appointment type and queue year were also moved out into a WHERE clause.

These optimizations resulted in improved times for all queries.

6. Results and Analysis

To evaluate the OLAP application, two types of testing were performed: functional testing and performance testing. Functional testing ensures that the OLAP application generates accurate reports. Meanwhile, performance testing evaluates the application's speed in producing the reports, which may impact user satisfaction.

6.1 Functional Testing

Functional testing aims to validate that the software performs according to the specifications of acquiring the total number of appointments for varying situations, parameters, and constraints. This testing was carried out by running the SQL queries in MySQL and the OLAP application with comparisons between the two outputs.

The first test intends to validate the aggregate number of appointments based on a given region.

The test was done two separate times, one for the expected output zse3using MySQL queries and the other is validating the data by analyzing the data in the OLAP application. With the query not needing any change in parameters, the SQL statement was tested several times to validate the consistency of the output.The data that was used differs in specific regions and groups of regions.

The application allows the filtering of multiple regions and acquiring the subtotal and overall total for the different regions. The test case includes one random region, all the regions, a region with a large number of appointments, and a regional with a small number of appointments.

Table 2. Test Results for Total # of Appointments per region

Test Case	Expected	Actual	Pass/Fail
Single Region (Central Luzon (III))	449	449	Pass
All Regions	Total: 320140 Bicol Region (V): 187 CALABARZON (IV-A): 3112 Central Luzon (III): 449 Central Visayas (VII): 24413 Eastern Visayas (VIII): 45 Ilocos Region (I): 25560 National Capital Region (NCR): 232292 Northern Mindanao (X): 82 SOCCSKSARGEN (Cotabato Region) (XII): 16207 Western Visayas (VI): 17793	Total: 320140 Bicol Region (V): 187 CALABARZON (IV-A): 3112 Central Luzon (III): 449 Central Visayas (VII): 24413 Eastern Visayas (VIII): 45 Ilocos Region (I): 25560 National Capital Region (NCR): 232292 Northern Mindanao (X): 82 SOCCSKSARGEN (Cotabato Region) (XII): 16207 Western Visayas (VI): 17793	Pass
Region with large number of appointments (NCR)	232292	232292	Pass
Region with small number of appointments (Northern Mindanao (X))	82	82	Pass

In Table 2, all four test cases gave a pass output in comparing both the expected and actual results. The overall total along with the subtotal and its specific regions remained consistent as well for the test cases which contained all the regions.

The second test intends to validate the aggregate number of appointments per province for each region.

The test was done two separate times, one for the expected output using MySQL queries and the other is validating the data by analyzing the data in the OLAP application. With the query not needing any change in parameters, the SQL statement was tested several times to validate the consistency of the output

Table 3. Test Results for Total # of Appointments per region

Test Case	Expected	Actual	Pass/Fail
Single Province (Leyte)	45	45	Pass
All Provinces	Bicol Region (V): Albay: 187 CALABARZON (IV-A): Cavite: 96 CALABARZON (IV-A): Laguna: 3016 Central Luzon (III): Bulacan: 337 Central Luzon (III): Zambales: 112 Central Visayas (VII): Cebu: 15148 Central Visayas (VII): Negros Oriental: 9265 Eastern Visayas (VIII): Leyte: 45 Ilocos Region (I): La Union: 86 Ilocos Region (I): Pangasinan: 25474 National Capital Region (NCR): Manila: 232292 Northern Mindanao (X): Lanao del Norte: 82 SOCCSKSARGEN (Cotabato Region) (XII): Cotabato: 15 SOCCSKSARGEN (Cotabato Region) (XII): South Cotabato: 16192 Western Visayas (VI): Iloilo: 17793	Bicol Region (V): Albay: 187 CALABARZON (IV-A): Cavite: 96 CALABARZON (IV-A): Laguna: 3016 Central Luzon (III): Bulacan: 337 Central Luzon (III): Zambales: 112 Central Visayas (VII): Cebu: 15148 Central Visayas (VII): Negros Oriental: 9265 Eastern Visayas (VIII): Leyte: 45 Ilocos Region (I): La Union: 86 Ilocos Region (I): Pangasinan: 25474 National Capital Region (NCR): Manila: 232292 Northern Mindanao (X): Lanao del Norte: 82 SOCCSKSARGEN (Cotabato Region) (XII): Cotabato: 15 SOCCSKSARGEN (Cotabato Region) (XII): South Cotabato: 16192 Western Visayas (VI): Iloilo: 17793	Pass
Province with large number of appointments (Iloilo)	17793	17793	Pass
Region with small number of appointments (Cotabato)	15	15	Pass

With the query not needing any change in parameters, the SQL statement was tested several times to validate the consistency of the output.The data that was used is similar from the first test with the type of test cases needed, the only difference being the value of the test cases uses provinces instead of the region.

In Table 3, all test cases passed the different scenarios that were set by the group. The values remained consistent based on the chosen province. Even with the filtering having multiple provinces, the value of the appointment count remained consistent with its corresponding province and region.

The third test intends to validate the aggregate number of appointments per appointment status based on the given city.

The test was done two separate times, one for the expected output using MySQL queries and the other is validating the data by analyzing the data in the OLAP application. Since the third test requires a parameter, it is adjusted as the city value for each case. The SQL statement was tested several times to validate the consistency of the output. The test cases consist of different cities with varying amounts of appointment count. The test cases were adjusted in the WHERE clause of the SQL to adjust the value of the cities that were chosen for each case. It allowed the group to keep changing the parameters, allowing for several unique test cases to be analyzed. The test cases were split into different categories based on unique scenarios such as a random city, a city with large appointment count, a city with small appointment count, and a city with no appointments found in the dataset.

Table 4. Test Results for Total # of Appointments per status in a city query

Test Case	Expected (SQL Query)	Actual (OLAP Applicaton)	Pass/Fail
Single City (Calamba City)	Complete: 1065 Queued: 317 Serving: 11 Skip: 1	Complete: 1065 Queued: 317 Serving: 11 Skip: 1	Pass
City with a large number of appointments (Quezon City)	Cancel: 74 Complete: 2637 NoShow: 60 Queued: 141553 Serving: 148 Skip: 4	Cancel: 74 Complete: 2637 NoShow: 60 Queued: 141553 Serving: 148 Skip: 4	Pass
City with a small number of appointments (Las Pinas)	Queued: 1	Queued: 1	Pass
City with no appointments (Nasugbu)	<none>	<none>	Pass

The results in Table 4 show a consistent pass on all the test cases that were created. The expected results contain the same values that were outputted by the OLAP application, validating the count of appointments per appointment status within the chosen cities.

The fourth test intends to validate the aggregate number of appointments based on all the main specialties that can be found within a given region.

The test was done two separate times, one for the expected output using MySQL queries and the other is validating the data by analyzing the data in the OLAP application. With the query not needing any change in parameters, the SQL statement was tested several times to validate the consistency of the output. The test cases vary on the region and the amount of appointments found in that region that contains doctors with main specialties.The test cases were adjusted in the WHERE clause of the SQL to adjust the value of the regions that were chosen for each case. It allowed

the group to keep changing the parameters, allowing for several unique test cases to be analyzed. Each case is also unique with the pattern being: Having a random region, a region with a large appointment count, and a region with a small appointment count.

Table 5. Test Results for Main Specialty Demands in terms of # of Appointments per Region

Test Case	Expected (SQL Query)	Actual (OLAP Applicaton)	Pass/ Fail
Single Region (*Illocos Region (I)*)	Total: 25560 Neurosurgery: 86 Pediatrics: 25474	Total: 25560 Neurosurgery: 86 Pediatrics: 25474	Pass
All Regions	Total in All Regions: 320140 Bicol Region (V) Total: 187 General Medicine: 187 CALABARZON (IV-A) Total: 3112 Cosmetic Medicine: 64 Cosmetic Surgery: 32 Dermatology: 1736 Pediatrics: 1280 ... (to the last few rows) SOCCSKSARGEN (Cotabato Region) (XII) Total: 16207 General Medicine: 15 Pediatrics: 16192 Western Visayas (VI) Total: 17793 Internal Medicine: 17793	Total in All Regions: 320140 Bicol Region (V) Total: 187 General Medicine: 187 CALABARZON (IV-A) Total: 3112 Cosmetic Medicine: 64 Cosmetic Surgery: 32 Dermatology: 1736 Pediatrics: 1280 ... (to the last few rows) SOCCSKSARGEN (Cotabato Region) (XII) Total: 16207 General Medicine: 15 Pediatrics: 16192 Western Visayas (VI) Total: 17793 Internal Medicine: 17793	Pass
Region with large number of appointments (NCR)	Total: 232292 Companion Animal Medicine: 2 Cosmetic Medicine: 5033 Cosmetic Surgery: 5393 Dermatology: 42538 Family Medicine: 2 ... (to the last few rows) Internal Medicine, Geriatrics: 57 Obstetrics and Gynecology: 1411 Ophthalmology: 21593 Orthopedic Surgery: 51 Pediatrics: 1681 Pulmonology: 154 Veterinary Medicine: 140057	Total: 232292 Companion Animal Medicine: 2 Cosmetic Medicine: 5033 Cosmetic Surgery: 5393 Dermatology: 42538 Family Medicine: 2 ... (to the last few rows) Internal Medicine, Geriatrics: 57 Obstetrics and Gynecology: 1411 Ophthalmology: 21593 Orthopedic Surgery: 51 Pediatrics: 1681 Pulmonology: 154 Veterinary Medicine: 140057	Pass
Region with small number of appointments (Eastern Visayas (VIII))	Total: 45 General Medicine: 45	Total: 45 General Medicine: 45	Pass

The results in Table 5 show consistent PASS results for all the test cases. The number of appointments for each specialty found in the region along with the main specialty list is consistent with both the expected and actual results.

The fifth test intends to validate the aggregate number of appointments based on two varying parameters which are the appointment type and the year of appointment queued.

The test was done two separate times, one for the expected output using MySQL queries and the other is validating the data by analyzing the data in the OLAP application. Different type and year variables were used in testing the data. This was changed in the WHERE clause where the query uses parameters which allows the user to change the two attributes and adjust the data based on the parameters. All years are unique while the appointment type interchanged between "*Consultation*" and "*Inpatient*" value

Table 6. Test Results for Total # of Appointments per region with specific type in a specific year

Test Case	Expected (SQL Query)	Actual (OLAP Application)	Pass/Fail
Consultation and 2024	CALABARZON (IV-A): 36 Central Luzon (III): 5 Central Visayas (VII): 423 Ilocos Region (I): 249 National Capital Region (NCR): 3214 SOCCSKSARGEN (Cotabato Region) (XII): 318 Western Visayas (VI): 199	CALABARZON (IV-A): 36 Central Luzon (III): 5 Central Visayas (VII): 423 Ilocos Region (I): 249 National Capital Region (NCR): 3214 SOCCSKSARGEN (Cotabato Region) (XII): 318 Western Visayas (VI): 199	Pass
Consultation and 2022	CALABARZON (IV-A): 714 Central Luzon (III): 94 Central Visayas (VII): 5179 Ilocos Region (I): 3168 National Capital Region (NCR): 23106 Northern Mindanao (X): 2 SOCCSKSARGEN (Cotabato Region) (XII): 5945 Western Visayas (VI): 3013	CALABARZON (IV-A): 714 Central Luzon (III): 94 Central Visayas (VII): 5179 Ilocos Region (I): 3168 National Capital Region (NCR): 23106 Northern Mindanao (X): 2 SOCCSKSARGEN (Cotabato Region) (XII): 5945 Western Visayas (VI): 3013	Pass
Inpatient and 2020	CALABARZON (IV-A): 2 Central Luzon (III): 14 Central Visayas (VII): 1 National Capital Region (NCR): 21	CALABARZON (IV-A): 2 Central Luzon (III): 14 Central Visayas (VII): 1 National Capital Region (NCR): 21	Pass
Consultation and 2013	National Capital Region (NCR): 2413	National Capital Region (NCR): 2413	Pass
Inpatient and 2022	Central Luzon (III): 1 National Capital Region (NCR): 11	Central Luzon (III): 1 National Capital Region (NCR): 11	Pass

The results of the test cases in Table 6 show a consistent PASS score for all the different test cases. The expected and actual results are consistent down to the regions that contain the two parameters and the number of appointments in that region that also follows the two parameters of appointment type and year.

To validate the **ETL process**, the group decided to compare the number of records in all tables in the source MySQL database to the number of records in the data warehouse after the loading process from the source MySQL database to the data warehouse using Apache NiFi's flows and processors. In addition, the columns and their respective data types in the data warehouse were retained; no data type or foreign key constraint violation was found during and after the loading process.

6.2 Performance Testing

Performance testing aims to measure the speed of executing the SQL queries depending on the initial and final optimization. In this section, the performance testing was divided into two phases, both referring to the two optimization strategies discussed in the previous chapter 5.

For the first performance testing, the SQL queries' performance (already optimized from subqueries to JOINs) were tested based on two scenarios: the absence and the presence of foreign key relationships and indexing.

6.2.1 Performance of Queries using Foreign Key Relationships and Indexing

Table 7. Performance Results for all Queries.

Survey Item	No index nested-loop join	Indexed nested-loop join
Query #1	1.563s	0.687s
Query #2	0.5969s	0.375s
Query #3	0.250s	0.593s
Query #4	N/A	1.5s

Query #5	0.391s	0.343s

As per Table 7, queries 1, 2, and 4 showed the significant improvement of establishing primary indexing in each table and foreign key relationships between the dimension and fact tables. Specifically, for the 4th query, the MySQL workbench consistently lost connection due to the huge response delay. However, for queries 3 and 5, indexing and foreign key relationships establishment did not significantly impact their performance. It is worth noting that queries 3 and 5 both involved slice and dice operations involving either the columns *city, appointment type,* or *year of queue date*. This may be due to the fact that even though primary key indexes contributed to quicker JOIN operations, the time complexity of filtering rows based on the WHERE clause and specific columns not related to the primary key may have outweighed the time complexity of indexed nested loop joins. In the case queries 1, 2, and 4, they all did not have WHERE clauses and only involved JOINs and GROUP BYs. This observation highlights the importance of understanding both the type of indexing and when to use indexing based on the structure of one's SQL queries. Although 3 out of 5 original queries benefitted from primary key indexing and foreign key relationships, the benefits of data integrity from foreign key relationships and faster JOIN operations due to indexing generally outweigh the costs.

Table 8. Performance of Queries After Optimization

Q1	Q2	Q3	Q4	Q5
0.542	0.512	1.351	1.787	0.251
0.533	0.516	1.224	1.800	0.309
0.526	0.509	1.209	1.765	0.183
0.528	0.514	1.185	1.787	0.233
0.529	0.504	1.187	1.782	0.181

As it is seen in Table 8, Q1, Q2, and Q4, all had significant improvements in their times that resulted in sub 2 second query times. Q3 and Q5, however, showed improvements, although minor.

7. Conclusion

In this paper, we discussed the implementation of an application that utilizes OLAP operations to retrieve data from large data sets. It was vital to utilize a data warehouse that is populated from the SeriousMD dataset which contains various tables such as the appointments, px, clinics, and doctors to perform various operations such as the ETL solution for the pre-processing and the analytical reports. The application utilizes queries that apply different OLAP operations for various purposes, all concerning the amount of appointments with varying variables. Since the application focuses on the analytical reports that can be retrieved from the datasets, using OLAP helps in analyzing the data which could be used in decision making scenarios by using complex queries to gather specific data regarding the number of appointments.

ETL helps keep the warehouse updated by allowing periodical running of ETL jobs that refresh the data with the latest changes from the source. It can automatically insert values into the warehouse if new rows are created in the source.

In creating analytical reports, we realized that using OLAP applications in the data warehouse was the most efficient way to create the reports since it can handle large amounts of data at fast speeds, usually only taking a few seconds to generate the analytical reports. Applying typical SQL statements to run the queries was not enough even though it produced the same output. An application used for analyzing large datasets requires further optimization so that the query would be able to handle it even if the datasets get larger. Having optimized queries provides a sustainable and long-term analytical report application that can be used by medical analysts from different areas with varying data. Query optimization can be done by different strategies, some of which we incorporated such as primary key indexing and query restructuring utilizing JOINs.

Results showed that all five of the analytical reports showed faster running time after optimization, with the fourth report showing the most improvement of 17.021 seconds, meeting the criteria of running each query for under 5 seconds.

Overall, the utilization of the data warehouse and OLAP operations allowed us to create an application that adjusts to different sizes and values of a dataset. The simplicity of our chosen star schema provides the application with faster query performance as opposed to using a different schema design such as the snowflake schema.

8. References

[1] AWS. n.d.. B-Tree Indexes. https://docs.aws.amazon.com/dms/latest/oracle-to-aurora-mysql-migration-playbook/chap-oracle-aurora-mysql.tables.btree.html

[2] Confluent. n.d.. What is Apache NiFi?. https://www.confluent.io/learn/apache-nifi/

[3] Databricks. n.d.. What is Star Schema?. https://www.databricks.com/glossary/star-schema

[4] Erin M., SSGB, LSSBB, FMVA, M.S., AWS-CCP. 2023. 10 Reasons Data Scientists Love Jupyter Notebooks. Linkedin. https://www.linkedin.com/pulse/10-reasons-data-scientists-love-jupyter-notebooks-erin/

[5] GeeksforGeeks. 2023. Introduction of B-Tree. https://www.geeksforgeeks.org/introduction-of-b-tree-2/

[6] IBM. n.d.. What is OLAP?. https://www.ibm.com/topics/olap

[7] insightsoftware. 2023. Comparing Data Visualizations: Bar vs. Stacked, Icons vs. Shapes, and Line vs. Area. https://insightsoftware.com/blog/comparing-data-visualizations-bar-vs-stacked-icons-vs-shapes-and-line-vs-area/

[8] Jaspersoft. n.d..What is a Map Chart?. https://www.jaspersoft.com/articles/what-is-a-map-chart

[9] Linkedin. 2023. How can you use bar charts to compare sales performance by region?. https://www.linkedin.com/advice/0/how-can-you-use-bar-charts-compare-sales-performance-4wluf

[10] Luenendonk, M. 2019. What is OLAP?. Cleverism. https://www.cleverism.com/what-is-olap/

[11] MySQL. n.d.. 10.3.1 How MySQL Uses Indexes. https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html

[12] Oracle Philippines. n.d.. What Is a Database?. https://www.oracle.com/ph/database/what-is-database/

[13] Oracle Philippines. n.d.. What Is a Data Warehouse?. https://www.oracle.com/ph/database/what-is-a-data-warehouse/

[14] Prakash A. 2022. Query Optimization in DBMS. Scaler Topics. https://www.scaler.com/topics/query-optimization-in-dbms/

[15] Silberschatz, A., Korth, H. & Sudarshan, S. (2019). Database System Concepts, 7th Edition. McGraw-Hill Book Co.

[16] Yi, M. n.d.. Chartio. https://chartio.com/learn/charts/pie-chart-complete-guide/

9. Declarations

9.1 Declaration of Generative AI in Scientific Writing.

During the preparation of this work the authors used Bing AI and ChatGPT in order to correct grammar, provide better alternatives for words, and better sentence composition. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

9.2 Record of Contribution

Member	Task
Martinez	ETL Script, Query Optimization, Performance Testing, Technical Report
Rebano	Dimensional Model, OLAP Application, Functional Testing, Technical Report
Samson	ETL Script, Query Optimization, Performance Testing, ETL Functional Testing, Technical Report
Villanueva	Dimensional Model, OLAP Application, Functional Testing, Technical Report