STADVDB MCO1 Technical Paper

Johndayll Lewis D. Arizala[1], John Kovie L. Niño[2], Zach Matthew D. Noche[3], and Donnald Miguel L. Robles[4]
[1]johndayll_arizala@dlsu.edu.ph, [2]john_kovie_nino@dlsu.edu.ph, [3]zach_noche@dlsu.edu.ph, [4]donnald_robles@dlsu.edu.ph

ABSTRACT

This paper investigates the development and optimization of a data warehousing and Online Analytical Processing (OLAP) framework aimed at enhancing healthcare data analytics. Utilizing anonymized datasets from the SeriousMD platform, the researchers engineered a star-schema-based data warehouse and implemented a comprehensive Extract, Transform, Load (ETL) process pipeline to ensure the integrity and utility of the data. The creation of an OLAP application enables healthcare administrators and professionals to conduct sophisticated analytical queries, thereby facilitating informed decision-making processes. The application leverages advanced SQL constructs and implements OLAP operations such as roll-up, drill-down, slice, and dice to produce actionable insights. Through rigorous query processing and optimization techniques, the study showcases significant improvements in query execution times, underscoring the critical impact of database design, the ETL process, and query optimization on system performance. The findings of this paper validate the effectiveness of the OLAP system in supporting healthcare management, provide valuable insights into database design and optimization literature, and demonstrate the practical application of OLAP systems in improving healthcare outcomes through data-driven analysis.

Keywords

Data Cleaning, ETL, OLAP, Query Processing, Query Optimization, Amazon Cloud Services, MySQL, Apache Nifi, Tableau

1. Introduction

The advent of technology has made it possible to store vast amounts of data, addressing past data management challenges. The health industry has greatly benefited from this, as it has improved the storage of patient information. However, a new problem has arisen: the ability to store excessive amounts of data simultaneously, making it difficult to manage and analyze. This has necessitated the development of efficient systems for data storage, processing, and analysis. One such system is the data warehouse, a pivotal component of business intelligence that enables comprehensive data analysis and reporting. Therefore, this paper presents an in-depth study and implementation of a data warehouse system, with a particular emphasis on Online Analytical Processing (OLAP) operations and optimized Query Processing.

The dataset used for this project came from anonymized and randomized data from a service application called SeriousMD. It comprises information about doctors, patients, clinics, and appointments. It provides a rich source of data for analysis, offering insights into various aspects of healthcare service delivery.

The data warehouse constructed for this project follows a star schema design, with the Appointments table serving as the fact table, and the Doctors, Px, and Clinics tables as dimension tables. This design facilitates efficient data retrieval and simplifies the structure of the database, making it more understandable for end-users.

An OLAP application was developed to enable users to perform complex analytical queries in the data warehouse. The application supports various OLAP operations, including roll-up, drill-down, slice, and dice, allowing users to view data from multiple perspectives and at varying levels of granularity.

The intended users of this system are healthcare administrators, analysts, and researchers who need to analyze healthcare service delivery data and the relation of geolocation for decision-making and research purposes. The system allows these users to gain meaningful insights from the data, such as appointment trends across regions, and explore geospatial relationships and subsequent correlations, which can be used to improve service delivery and patient outcomes.

Lastly, this paper also explores query processing and optimization strategies to enhance the performance of the OLAP application. These strategies include creating secondary indexes, rewriting query statements, and redesigning tables. It will then be analyzed whether the optimization strategies improved the query performance or not via the cost estimation and time metric.

2. Data Warehouse

As mentioned in the introduction, the data warehouse follows the star schema design, with the 'appointments' table serving as the fact table, and the 'doctors', 'px', and 'clinics' tables serving as dimension tables.

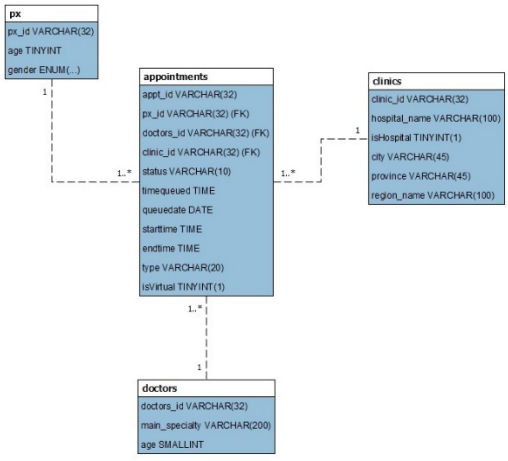

Figure 1. Data Warehouse Model of the Dataset

Interestingly, the structure of the data warehouse closely resembles the structure of the original data source. This is because the original data was already well-organized and followed a structure that closely resembled a star schema [2]. This not only simplified the process of designing the data warehouse but also ensured a high degree of consistency between the source data and the data in the warehouse.

The fact table 'Appointment' contains 11 columns, 1 being its primary key, 3 being a foreign key to the dimension tables, and the rest are measures related to appointments. The following are the columns:

- **appt_id:** The primary identifier for each appointment.
- **px_id:** The foreign identifier for each patient.
- **doctors_id:** The foreign identifier for each doctor.
- **clinic_id:** The foreign identifier for each clinic.
- **status:** The status of the appointment.
- **timequeued:** The time the patient was queued.
- **queuedate:** The date the patient was queued.
- **starttime:** The start time of the appointment.
- **endtime:** The end time of the appointment.
- **type:** The type of appointment (Consultation or Inpatient).
- **isVirtual:** Whether the appointment was virtual or not.

The design done for the fact table was focused on ensuring proper data types and constraints. Mainly, the 4 identifiers were given constraints. A primary key constraint was implemented on the 'appt_id' column. This was done to facilitate efficient indexing [1], which is crucial for rapid data retrieval in large-scale data environments.

Foreign key constraints were established on the 'px_id', 'doctors_id', and 'clinic_id' columns. These foreign keys link the fact table to the respective dimension tables, ensuring relational integrity. This design choice prevents orphan records and maintains a consistent view of the data.

Regarding data types, exact lengths were assigned to string data types. This decision optimizes storage by not allocating more space than necessary. For time-related data, the data type was chosen based on the nature of the data. For instance, the 'queuedate' column uses the '*date*' data type, while the 'timequeued', 'starttime', and 'endtime' columns use the '*time*' data type. This not only reflects the nature of the data but also allows for more efficient storage and querying.

The dimensions in this model are represented by the 'doctors', 'px', and 'clinics' tables. Each table contains attributes describing a specific aspect of the data.

- **doctors table**: This contains attributes related to doctors, such as their unique identifier, main specialty, and age.
- **px table**: Contains attributes related to patients, such as their unique identifier, age, and gender.
- **clinics table**: This table contains attributes related to clinics, such as their unique identifier, hospital name, whether it's a hospital, and location (city, province, region).

The hierarchy in each dimension is simple, as each dimension table contains only one level of detail. For example, the Doctors table contains details about individual doctors but does not contain any higher-level groupings of doctors.

During the model design process, a few issues were encountered. One issue was the naming of the columns. To address this, the columns were renamed to follow a consistent naming convention. Another issue was the use of the reserved keyword 'virtual' as a column name in the Appointments table. This was resolved by renaming the column to isVirtual. Lastly, design choices were significantly influenced by the data exploration and cleaning process. For instance, duplicate patient data in the 'px' table, likely due to data gathering errors, were dropped to maintain data integrity.

In the 'clinic' dimension table, despite missing hospital names for non-hospital entries, data was retained due to the completeness of geolocation information, enabling valuable insights.

The 'doctors' table presented a challenge with the 'main_specialty' column, which contained numerous variations of the same specialty. Efforts were made to standardize these entries, but due to the sheer number of variances, it was not fully resolved.

In the 'appointments' table, a surprising discovery was that only a small percentage of foreign keys were used in the large dataset. Consequently, unused data from other tables was dropped. This decision was justified as the OLAP application's visual dashboard primarily uses the 'appointments' table to connect dimension tables based on specific query requirements.

These actions ensured the data's cleanliness, integrity, and optimization while also addressing the challenges encountered during the data cleaning process.

3. ETL Script

One process pivotal to data warehousing is the Extract, Transform, Load (ETL) process. It's a complex process that involves extracting data from outside, multiple sources, transforming it to fit business needs, and finally loading it into the end target - the data warehouse [3]. Several challenges were present during this process, particularly due to the large volume and unclean nature of the source data and the limited capability of the member's work machine. However, a strategy was devised to overcome these hurdles and ensure a smooth ETL process.

To tackle the challenges, three major steps were done:

1. Set up a Virtual Private Server (VPS) to host the Apache NiFi application, which eliminated the need for constant setup.
2. Utilize AWS MySQL to ensure the data warehouse was accessible to all team members and to maintain synchronization of the local databases.
3. Clean the dataset using a separate program (Jupyter), where duplicate data and data violating foreign key constraints were removed (mentioned previously).

These prerequisites smoothened the ETL process and facilitated its setup. Consequently, the ETL script only needed a light setup to make it work.

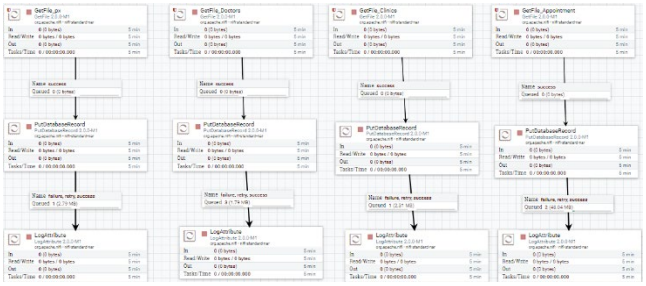

Figure 2. ETL Script in Apache Nifi

The **Extraction** process involved using pandas in Jupyter Notebook to process and output a cleaned copy of the data source. This cleaned data was then placed into the local folder of the hosted app. Four "GetFile" jobs were created, one for each table.

The **Transformation and Loading** processes were performed simultaneously using the "PutDatabaseRecord" job processor. The CSV reader was used to transform the data from CSV format to a format suitable for insertion/loading into the data warehouse hosted in AWS.

There were concerns during the ETL process, such as constraint violations, incorrect data type assertions, and other potential errors. These were significant considerations in the data cleaning process because if left unaddressed, it would have complicated the ETL process and hindered the smooth flow of data from the source to the warehouse.

To address these concerns, a rigorous data-cleaning process was implemented. This process involved several steps:

1. **Constraint Violations**: Ensured that all foreign key relationships were valid, which prevented constraint violations when loading the data into the warehouse. This was achieved by cross-referencing the keys in the fact table with those in the dimension tables and removing any orphan records.
2. **Data Type Assertions**: Verified that the data type of each column in the cleaned data matched the corresponding column in the data warehouse schema. This prevented data type assertion errors during the loading process. For instance, numeric (integer) types were used for numeric data, date types for dates, and string types for textual data.
3. **Duplicate Data**: Identified and removed duplicate records in the data source. This was particularly relevant for the 'px' (patients) table, where duplicate patient data was likely due to errors in the data gathering process.
4. **Missing Data**: Ensured that attributes that are meant to not be null (keys and some attributes such as location) do not have null values.

Addressing these challenges through meticulous data cleaning and strategic infrastructure choices ensured a streamlined and efficient ETL script. Thus, this script leveraged Apache NiFi's capabilities, coupled with pre-processing steps, to facilitate a smooth and effective data integration process into the data warehouse.

4. OLAP Application

Once the data warehouse has been set up and filled with the necessary data. A visual dashboard application, utilizing software called "Tableau", was made to easily visualize the query result from the data warehouse [4]. This was done to provide medical analysts and administrators with a tool to generate analytical report results in a user-friendly interface. The application is primarily intended for decision-making tasks related to the geographical distribution of healthcare services.

Four types of analytical reports were created to correspond to different OLAP operations: roll-up, drill-down, slice, and dice. These reports were made to provide valuable insights into the demand for different medical specialties across various provinces, the demand for different medical specialties across various cities, comparing the number of appointments in hospital clinics versus non-hospital clinics in a specific province, and the comparison of appointments of a doctor's specialty in two or more regions.

4.1 Roll-up Analysis Report

The analytical report intended for the roll-up operation provides a broad overview of the demand for different medical specialties across various provinces in the Philippines. Essentially, the operation is performed on the geographic dimension of the dataset. Say a medical analyst wants to know the number of appointments for each doctor's specialty and the total appointments across provinces. The result of this query would help to understand the needs of the locals per province. The SQL statement to achieve this would utilize the use of the "WITH ROLLUP" operation to show the aggregated total of appointments per province.

```
SELECT c.province, d.main_specialty, COUNT(*) AS num_appointments
FROM appointments a
JOIN doctors d ON a.doctors_id = d.doctors_id
JOIN clinics c ON a.clinic_id = c.clinic_id
WHERE d.doctors_id IN (SELECT doctors_id FROM appointments)
AND c.clinic_id IN (SELECT clinic_id FROM appointments)
GROUP BY c.province, d.main_specialty WITH ROLLUP;
```

Figure 3. Roll-up SQL Query

The SQL statement in question combines several operations to extract meaningful insights from the data. It begins with a "JOIN" operation, which merges rows from the 'appointments', 'doctors', and 'clinics' tables based on related columns. The "WHERE" clause, used with subqueries, filters the data to only include doctors and clinics that have at least one appointment. The "SELECT" statement is used to display the province, main specialty, and the count of appointments. The "COUNT" function is used to calculate the total number of appointments for each combination of province and medical specialty. Lastly, the "GROUP BY" clause, combined with the "WITH ROLLUP" modifier, is used to create appointment subtotals per group of doctor specialty and grand totals per province, providing a hierarchical, multi-level summary of the data.

This simple query provides the end-user with a powerful tool for analyzing geographic trends in doctor specialist demands.

Table 1. Sample Query Result of Roll-up SQL Statement

province	main_specialty	num_appointments
Laguna	Dermatology	1308

Laguna	Pediatrics	977
Laguna	null	2285

This query result would then be visualized via a heatmap in the application dashboard. The application also allows users to filter by a specific province.

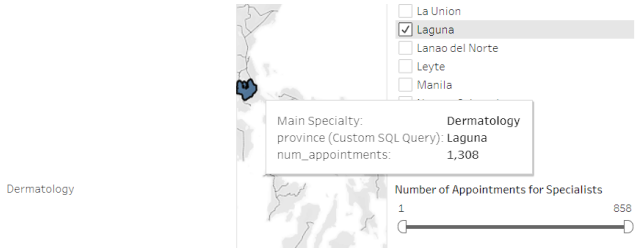

Figure 4. Heat Map Visualization of Roll-up Query Result

4.2 Drill-down Analysis Report

The analytical report for the drill-down operation provides a more granular view of the demand for different medical specialties in a specific city. A medical analyst could query this report, and the result would then be analyzed to get the general data on the common needs of locals in a specific city.

```
SELECT
    (SELECT city FROM clinics WHERE clinic_id = a.clinic_id) AS city,
    (SELECT main_specialty FROM doctors WHERE doctors_id = a.doctors_id) AS main_specialty,
    COUNT(a.appt_id) AS num_appointments
FROM
    appointments a
GROUP BY
    city, main_specialty;
```

Figure 5. Drill-down SQL Query

This SQL query counts the number of appointments for each doctor's specialty in various cities by employing subqueries within the "SELECT" clause to associate appointments with cities and specialties. For each appointment in the appointments table, the city is identified by matching "clinic_id" with the clinics table, and the specialty is determined by matching "doctors_id" with the "doctors" table. The query then aggregates these appointments by city and specialty.

Like the roll-up SQL query, this query provides end-users a tool to visualize the demand of doctor specialists across cities. The application also allows filtering by specific cities.

Table 2. Sample Query Result of Drill-down SQL Statement

city	main_specialty	num_appointments
Quezon City	Dog & Cat Medicine & Surgery	33868
General Santos City	Pediatrics	16191
Iloilo City	INTERNAL MEDICINE-Geriatrics	10325

This query result would then be visualized via a treemap in the application dashboard.

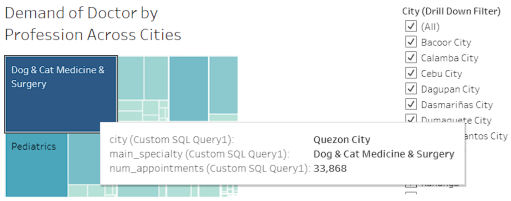

Figure 6. Treemap Visualization of Drill-down Query Result

4.3 Slice Analysis Report

The analytical report that utilizes the slice operation focuses on the healthcare facility dimension of the dataset, showing a comparison that helps to understand the landscape of healthcare service provision. For example, a healthcare administrator might use this operation to analyze the balance between appointments conducted in hospitals versus non-hospital clinics within a particular province, providing insights into infrastructure utilization and patient preference trends.

```
SELECT
    (SELECT province FROM clinics WHERE clinic_id = a.clinic_id) AS province,
    (SELECT isHospital FROM clinics WHERE clinic_id = a.clinic_id) AS isHospital,
    COUNT(a.appt_id) AS num_appointments
FROM
    appointments a
GROUP BY
    province, isHospital;
```

Figure 7. Slice SQL Query

This SQL query is designed to retrieve information about the number of appointments made at different clinics, grouped by the province and whether the clinic is a hospital or not.

The "SELECT" statement fetch "province" and "isHospital" from the clinics table and counts "appt_id" from the appointments table. The fields from clinics are fetched using subqueries, which are nested queries that fetch related data from different tables, done to fetch the required data which is spread across multiple tables. The "FROM" clause specifies the appointments table as the source of data as this is the fact table connecting the dimension tables needed for data. The "GROUP BY" clause groups the selected rows based on the values in the "province" and "isHospital" columns. This allows the query to count the number of appointments for each unique combination of "province" and "isHospital"

This query provides end-users a tool to easily visualize the number of appointments done in hospitals vs non-hospitals within a specific province. Say the healthcare administrator is interested in comparing appointments in the province of Manila.

Table 3. Sample Query Result of Slice SQL Statement

province	isHospital	num_appointments
Manila	0	62620
Manila	1	11386

This query result would then be visualized via a pie chart in the application dashboard.

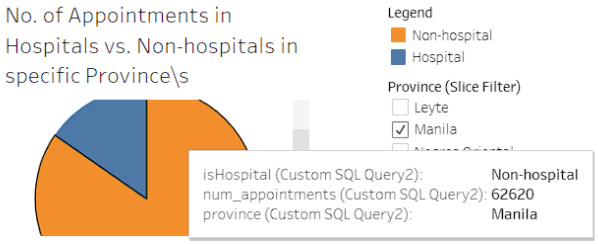

Figure 8. Pie chart Visualization of Slice Query Result

4.4 Dice Analysis Report

The analytical report specific to the dice operation is focused on understanding the appointment modalities across different regions. Say a medical analyst or administrator is interested in knowing the conduct of appointments by a doctor's specialty in regions like the NCR or CALABARZON. The result of the query would aid in analyzing regional preferences for medical consultations and the accessibility of technological solutions in healthcare delivery.

```
SELECT
    (SELECT region_name FROM clinics WHERE clinic_id = a.clinic_id) AS region_name,
    (SELECT main_specialty FROM doctors WHERE doctors_id = a.doctors_id) AS main_specialty,
    a.isVirtual,
    COUNT(a.appt_id) AS num_appointments
FROM
    appointments a
GROUP BY
    region_name, main_specialty, a.isVirtual;
```

Figure 9. Dice SQL Query

The employed SQL query slices the dataset by leveraging subqueries within the **SELECT** clause to extract the "region_name**"** and "main_specialty**"**, while also considering whether appointments are virtual or in-person ("isVirtual**"**).

Grouping by "region_name", "main_specialty", and "isVirtual" is an essential dice operation that allows for granular analysis of how different specialties are approached in terms of appointment modality across selected regions, providing a multidimensional view of the data.

Like the slice operation query, the use of "WHERE" clause is not used to allow for a general set of query results. This is due to the application that allows the data to be filtered. Essentially, it makes it a powerful tool for users to easily see the visualization of data they want.

Table 4. Sample Query Result of Dice SQL Statement

region_ name	main_specialty	isVirutal	num_app ointments
National Capital Region (NCR)	Dermatology	0	11130
National Capital Region (NCR)	Dermatology	1	836
CALAB ARZON (IV-A)	Dermatology	0	1306
CALAB ARZON (IV-A)	Dermatology	1	2

This query result would then be visualized via a stacked bar chart in the application dashboard.

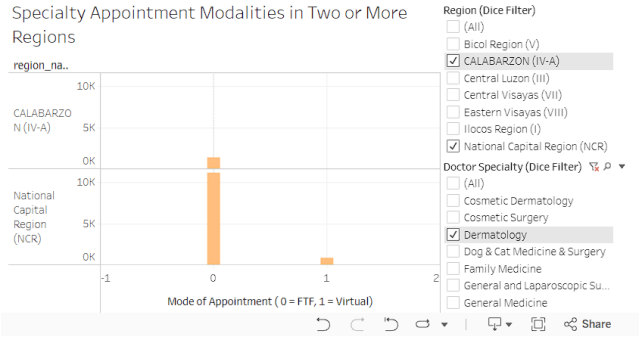

Figure 10. Stacked Bar Chart Visualization of Dice Query Result

5. Query Processing and Optimization

In the context of developing an OLAP application, the importance of query optimization is simple, user experience. Website loading statistics show that 40% of consumers will wait no longer than three (3) seconds [6]. Anything more than that would give a high chance of users clicking out of the website. With this, it is easy to see the importance of query speeds, knowing that it is the bridge between the data and the user interface. Additionally, system resources are finite, and slow queries may consume a hefty number of resources [7]. Scaling an application up to a multitude of users will cause some problems. Thus, it is essential to ensure that queries are carried out as efficiently as possible to increase speeds. The strategies to accomplish this can be divided into the following [5]:

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

The application-level tackles formulating the proper queries, the database level tackles the schema, the DBMS level configures the optimizer, lastly, the hardware level handles the physical infrastructure.

Analyzing the speed of a query will be done simply by measuring the duration of its execution. Moreover, knowing that the duration varies per execution, the query will be run ten times, and its average will be its metric to measure its speed. The current SQL statements present in the OLAP application have the following speeds in seconds, with the average being in the last row:

Table 5. Duration per Execution in the OLAP Application (in seconds)

Roll-Up	Drill-Down	Slice	Dice
16.593	1.078	0.766	1.125
16.782	1.031	0.641	1.016
16.282	1.031	0.671	1.140
16.282	0.922	0.672	1.015
17.063	0.953	0.656	1.125

15.984	1.188	0.641	1.094
16.125	0.954	0.610	1.390
16.531	0.969	0.641	1.140
16.500	0.985	0.625	1.125
16.078	1.092	0.718	1.109
16.422	1.02	0.664	1.128

It is seen that the roll-up query is the most inefficient, taking an average of 16.422 seconds. The rest of the queries take roughly a second to complete on average, with the slice being the fastest, taking only 0.664 seconds to execute on average. Looking back at the existing queries, they all utilize sub-queries. From the nature of what the OLAP application performs, it is possible to create a new query that produces the same results but eliminates the use of sub-queries.

5.1 Optimizing Roll-Up

The roll-up functionality displays all medical specialties under a particular province and the number of appointments. Rewriting the query to avoid subqueries results into the following:

```
SELECT c.province, d.main_specialty, COUNT(a.appt_id) AS num_appointments
FROM appointments a
JOIN doctors d ON a.doctors_id = d.doctors_id
JOIN clinics c ON a.clinic_id = c.clinic_id
GROUP BY c.province, d.main_specialty WITH ROLLUP;
```

Table 6. Sample Query Result of Roll-Up SQL Statement

province	main_specialty	num_appointments
Albay	General Medicine	1
Albay	null	1
Bulacan	Obstetrics and Gynecology	228

The WHERE clause, alongside its subqueries, was redundant and inefficient, as it checks whether the doctor's and clinic's ID are in the list of all doctors and clinic IDs respectively.

5.2 Optimizing Drill-Down

The drill-down functionality is a more detailed view compared to the roll-up, showing the city instead of the provinces. The query can be rewritten into:

```
SELECT c.city, d.main_specialty, COUNT(a.appt_id) AS num_appointments
FROM appointments a
JOIN doctors d ON a.doctors_id = d.doctors_id
JOIN clinics c ON a.clinic_id = c.clinic_id
GROUP BY c.city , d.main_specialty;
```

Table 7. Sample Query Result of Drill-Down SQL Statement

city	main_specialty	num_appointments
Makati	Cosmetic Dermatology	4558
San Juan	Integrative Medicine, Functional Medicine	4395
Quezon City	Dog & Cat Medicine & Surgery	33868

The two sub-queries can be translated into two joins, thus removing the redundant select statements from the two sub-queries.

5.3 Optimizing Slice

The slice operation displays the number of appointments for non-hospitals and hospitals under a certain province. Like the previous optimization, the query can be rewritten into:

```
SELECT c.province, c.isHospital, COUNT(a.appt_id) AS num_appointments
FROM appointments a
JOIN px p ON a.px_id = p.px_id
JOIN doctors d ON a.doctors_id = d.doctors_id
JOIN clinics c ON a.clinic_id = c.clinic_id
GROUP BY c.province, c.isHospital;
```

Table 8. Sample Query Result of Slice SQL Statement

province	isHospital	num_appointments
Manila	0	62620
South Cotabato	1	14820
South Cotabato	0	1371

5.4 Optimizing Dice

The dice operation displays the number of appointments that are virtual and non-virtual under each main specialty in each region. Once again, similar to the previous optimization, subqueries could be eliminated and be translated into two join clauses.

```
SELECT c.region_name, d.main_specialty, a.isVirtual, COUNT(a.appt_id) AS num_appointments
FROM appointments a
JOIN doctors d ON a.doctors_id = d.doctors_id
JOIN clinics c ON a.clinic_id = c.clinic_id
GROUP BY c.region_name, d.main_specialty, a.isVirtual;
```

Table 9. Sample Query Result of Dice SQL Statement

Region_name	main_specialty	isVirtual	Num_appointments
Bicol Region (V)	General Medicine	1	1
CALABARZON(IV-A)	Cosmetic Dermatology	0	63
CALABARZON(IV-A)	Cosmetic Surgery	0	17

5.5 Cost Estimation

Before running the actual test on the new queries, examining the cost estimation of both the new and the old queries could provide crucial insights regarding the nature of these operations. Theoretical analysis via cost estimation paves the way for the actual analysis where the actual time executed is measured.

5.5.1 Roll-Up Estimation

```
SELECT c.province, d.main_specialty, COUNT(*) AS num_appointments
FROM appointments a
JOIN doctors d ON a.doctors_id = d.doctors_id
JOIN clinics c ON a.clinic_id = c.clinic_id
WHERE d.doctors_id IN (SELECT doctors_id FROM appointments)
AND c.clinic_id IN (SELECT clinic_id FROM appointments)
GROUP BY c.province, d.main_specialty WITH ROLLUP;
```

$N_a = Number\ of\ rows\ in\ the\ appointments\ table$
$N_c = Number\ of\ rows\ in\ the\ clinics\ table$
$N_d = Number\ of\ rows\ in\ the\ doctors\ table$
N_{acd}
$= Number\ of\ rows\ in\ the\ table\ resulting\ from\ the\ join\ operation$

The notable operations within this query are the two join statements, the two sub-queries, and the group by. Thus, the cost estimation would be:

$$Join\ Operation\ for\ Doctors = N_a * O(logN_d)$$
$$Join\ Operation\ for\ Clinics = N_a * O(logN_c)$$

Knowing that MySQL uses a B-tree for indexes [8], the time complexity for searching via a B-tree is log(n). Thus, for each row within the appointments table, joining via the id is done with the help of the index in its respective table.

$$Where\ Subquery\ for\ doctors\ id = O(logN_d)$$
$$Where\ subquery\ for\ clinic\ id\ = O(logN_c)$$

The "WHERE IN" statement presents yet another search problem, and thus, MySQL should utilize indexes as seen here via the "EXPLAIN" clause:

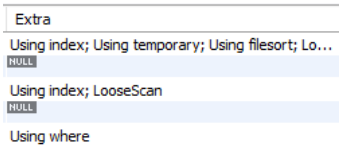

Figure 11. EXPLAIN statement for the Slice Operation

$$Group\ By\ Operation = O(N_{acd})$$

Lastly, the group by operation is done in one pass for non-sorted rows [9] [10], therefore performing a linear scan. The equation for the total cost would be:

$$RollUp\ Cost\ = N_a * O(logN_d) + N_a * O(logN_c) + O(logN_d) + O(logN_c) + O(N_{acd})$$

Going by with the same idea but for the optimized query, the total cost for the new query should be:

$$Join\ Operation\ for\ Doctors = N_a * O(logN_d)$$
$$Join\ Operation\ for\ Clinics = N_a * O(logN_c)$$
$$Group\ By\ Operation = O(N_{acd})$$
$$RollUp\ Cost\ = N_a * O(logN_d) + N_a * O(logN_c) + O(N_{acd})$$

The WHERE subquery has been eliminated, effectively shortening the equation. Theoretically, the new query should perform better, which is to be proven in part 6.

5.5.2 Drill-Down, Slice, and Dice Estimation

The structure of the drill-down, slice, and dice queries is fundamentally the same. The old queries contain two sub queries and one group by operation. Meanwhile, the new queries contain two join operations and one group by operation.

$N_1 = Number\ of\ rows\ in\ the\ appointment\ table$
$N_2 = Number\ of\ rows\ in\ the\ clinics$
$N_3 = Number\ of\ rows\ in\ the\ doctor's table$
N_{123}
$= Number\ of\ rows\ of\ the\ table\ resulting\ from\ the\ join\ operation$

$$Select\ Statement\ with\ Subqueries = N_1 * (O(logN_2) + O(logN_3))$$

For the old queries, it consists of two sub queries. Thus, for each row in the appointments table, the database executes two subqueries. Add the group by operation defined in the previous section, the total cost will result into:

$$Total\ Cost\ = N_1 * (O(logN_2) + O(logN_3)) + O(N_{123})$$

The structure of all the new queries, from the roll-up to the dice, are fundamentally the same, containing two joins and one group by. It is possible to derive the cost from part 5.5.1, which is:

$$RollUp\ Cost\ = N_1 * O(logN_3) + N_1 * O(logN_2) + O(N_{123})$$

Simplified into:

$$RollUp\ Cost\ = N_1 * (O(logN_2) + O(logN_3)) + O(N_{123})$$

This is the same as the total cost for the old queries of the drill-down, slice, and dice. Thus, the cost estimation for the last three operations is the same, even if it is optimized. Theoretically, the improvements should marginal (due to the optimizer), or possibly nonexistent, all of which is to be shown in part 6.

6. Result and Analysis

Similar to how the old queries were analyzed, the new queries will be run a total of ten times, and the average will be its metric to measure its speed.

Roll-Up	Drill-Down	Slice	Dice
2.078	0.485	0.453	0.625
2.046	0.578	0.422	0.672
0.906	0.844	0.406	0.625
0.969	0.531	0.516	0.641
0.968	0.500	0.454	0.656
0.890	0.515	0.422	0.609
1.141	0.515	0.468	0.610
0.844	0.500	0.469	0.610
0.860	0.484	0.594	0.640
0.844	0.469	0.516	0.672
1.155	0.542	0.472	0.636

All but the slice operation has made significant improvement. The roll-up is 15.627 seconds faster, the drill-down being 0.478 seconds faster, the slice being 0.192 seconds faster, and the dice being 0.492 seconds faster. Bridging the estimated cost from the

previous section and the actual time shows that the roll-up operation indeed had significant improvements, while the rest were marginal.

6.1 Indexing

When appropriately used, indexing is a powerful strategy to help boost performance. As mentioned previously, indexing could turn linear scans into a much faster operation. There exists one query where a compound index can be utilized, which is the slice operation, where the data is grouped by the province and the isHospital column. The rest of the queries are grouped by different columns from separate tables. Thus, indexes cannot be fully utilized [9].

A compound index containing (province, isHospital) was created named province_isHospital_COMPOUND. The guide to optimize a GROUP BY provided by the MySQL documentation states that the preconditions to utilize an index in GROUP BY statements are: 1) All the columns group by must be in one index, and 2) The index must sort these columns [9]. Thus, the compound index should be able to satisfy these preconditions, as the columns are contained within the index, and it is sorted by default via a B-tree. Despite this, it seems that the optimizer does not utilize the compound index.

	table	partitions	type	possible_keys	key	k	ref	rows	Extra
1	a	NULL	ALL	clinic_id_idx,doctors_id_idx	NULL	NULL	NULL	110323	Using temporary
1	d	NULL	eq_ref	PRIMARY	PRIMARY	9.	my...	1	Using index
1	c	NULL	eq_ref	PRIMARY,province_isHospital_COMPOUND	PRIMARY	9.	my...	1	NULL

Figure 12. EXPLAIN Clause for The Slice Operation

Looking at the first row, the "extra" column shows that it solely uses a temporary table to perform the group by despite having the compound index. This is a decision made by the optimizer; therefore, the existence of the compound index would not make any difference.

7. Conclusion

The project delved deeply into creating an OLAP application intended to help end users have a tool to visualize medical data easily for easier analysis. Concepts of data warehouse, schema model design, ETL pipeline, query performance, and query optimizations were all explored and used to develop the OLAP application fully. Intensive time, research, and experience were undergone for each concept and once the OLAP application was fully developed. It is clear that each of these concepts plays a crucial role in maintaining a smooth process from the source data to the data warehouse and finally queried by end users. Unlike OLTP systems, where the applications are focused more on transactional queries or actions that manipulate data in some way, OLAP is a robust methodology for uncovering insights from multidimensional data.

The data warehouse is the core artifact of OLAP operations. It is where data from various sources is pre-processed, transformed, and stored for analytical querying. The model schema design of the warehouse is essential as it impacts its performance and efficiency in terms of the query construction and the data retrieval process. Thus, the data warehouse design for the OLAP application was designed to meet the needs of the application users while maintaining good query performance.

While the data warehouse is important, the ETL pipeline is equally important. This process is important as it involves the extraction of source data, transforming it into a format suitable for analytical processing, and finally loading it to the data warehouse. This process takes time to set up as sometimes the original data source can be too dirty, which might need further pre-processing, and the script needs meticulous tinkering. However, once the process is smooth, it ensures that data is available, cleaned, and up to date, directly influencing the OLAP applications analytical capabilities.

Finally, the performance of the OLAP application itself. For end users to have good user experience, the performance of the application must be able to show the expected results as instantaneously as possible. Thus, performance optimization necessitated exploration of strategies to reduce query execution times. Initial queries were made using sub-queries but as explored in the paper, using JOIN and optimizing group by operations were pivotal to reduce query execution times. Strategies that involved modifying the data warehouse were also explored and employed. While the data warehouse was designed to be simple and complete with the proper data types and key constraints to ensure automatic indexing by MySQL, strategies like custom index proved to be crucial in under specific conditions, such as for the slice operation which gained some performance.

In conclusion, developing an OLAP application for medical data visualization and exploring relevant OLAP concepts contributes significantly to the field of database management, offering insights into the practical application of advanced database concepts. By facilitating more accessible medical data, the application aids in improving healthcare decision-making, highlighting the societal benefits of accessible and analyzable data. For database developers, the exploration of query optimization and custom indexing presents valuable methodologies for enhancing database performance. The findings from this paper thus serve as a resource for furthering database technology advancements and show the importance of database systems in addressing real-world challenges.

8. Reference

[1] Oracle, 2024. 10.3.2 Primary Key Optimization, https://dev.mysql.com/doc/refman/8.0/en/primary-key-optimization.html

[2] Microsoft, 2023. Understand star schema and the importance for Power BI, https://learn.microsoft.com/en-us/power-bi/guidance/star-schema

[3] IBM, 2022. What is ETL? https://www.ibm.com/topics/etl

[4] Salesforce, 2024. What is Tableau? https://www.tableau.com/why-tableau/what-is-tableau

[5] Oracle, Chapter 10 Optimization, https://dev.mysql.com/doc/refman/8.0/en/optimization.html

[6] Ryan E., 2023, Website Load Time Statistics: Why Speed Matters in 2024, https://www.websitebuilderexpert.com/building-websites/website-load-time-statistics/

[7] Apriorit, 2023, SQL Query Optimization: Benefits and Expert Tips, https://medium.com/that-feeling-when-it-is-

compiler-fault/sql-query-optimization-benefits-and-expert-tips-eae2c72837e3

[8] Oracle, 8.3.8, Comparison of B-Tree and Hash Indexes, https://dev.mysql.com/doc/refman/5.7/en/index-btree-hash.html

[9] Oracle, 10.2.1.17 GROUP BY Optimization, https://dev.mysql.com/doc/refman/8.0/en/group-by-optimization.html

[10] "What's the Asymptotic Complexity of Group By Operation?" Stack Overflow, https://stackoverflow.com/questions/4889669/whats-the-asymptotic-complexity-of-groupby-operation.

9. Declaration

9.1 Declaration of Generative AI in Scientific Writing.

During the preparation of this work, the authors used Bing AI to help enhance the writing flow of the paper by fixing grammar, suggestion of suitable words, and verify facts found in several documents related to MySQL. 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

Name	Contribution
Arizala, Johndayll Lewis D.	• ETL script • Query Optimization • Section 5 and 6 of the Technical Paper
Niño, John Kovie L.	• OLAP application • Query statement • Section 4 of the Technical Paper
Noche, Zach Matthew D.	• Data Cleaning • Data Warehouse design • Section 1, 2, 3 of the Technical Paper
Robles, Donnald Miguel L.	• OLAP application • Section 7 of the Technical Paper