A Data Warehouse for a Telehealth Appointment System

Tean Jeremy W. Concio[1], Matthew Adrian U. Chua[2], Nicole Daphne C. Ong[3], Raphael Jeremiah C. Tan Ai[4]

College of Computer Studies, De La Salle University

[1]tean_concio@dlsu.edu.ph, [2]matthew_adrian_u_chua@dlsu.edu.ph, [3]nicole_daphne_ong@dlsu.edu.ph, [4]raphael_tanai@dlsu.edu.ph

ABSTRACT

SeriousMD is a startup involving Telehealth Appointments that works with the “SeriousMD Appointment” dataset, which contains appointment, doctor, clinic, and patient information involving their telehealth appointments. As the dataset contained large numbers of records, it was difficult to derive insights from it at a glance, needing analytical reports to understand the data better. This paper presents a data pipeline for data analysis and query optimization of the SeriousMD Appointment dataset. A data warehouse was constructed to collect the data into one location, after which an ETL (Extact, Transform, Load) script was created to import the data into the data warehouse. Afterward, SQL queries using OLAP operations were created and implemented inside the OLAP Application in order to visualize the data. The paper also explores query optimization by conducting indexing and query restructuring to improve query performance. The Optimization methods employed proved to have improved execution time by comparing execution times before and after optimization. The optimized queries were found to have a 3.5% increase in querying speed.

Keywords

Distributed database design, Transaction isolation levels, Data fragmentation, data replication, Concurrency control, Update and recovery schema.

1. Introduction

The SeriousMD Appointment dataset contains anonymized telehealth entries featuring appointment details. It includes information about the logging or booking of patients’ visits to doctors and clinics in certain time slots. To effectively centralize, organize, and optimize the dataset for data analysis later on, the records were loaded into a data warehouse containing a schema mirroring the dataset appropriately. Afterward, several Online Analytical Processing (OLAP) operations, tools, and techniques were employed to analyze the different dimensions of the dataset and generate meaningful reports to help users understand the nature of the dataset.

This paper demonstrates how the members built a data pipeline for the SeriousMD Appointment dataset to perform data analysis and query optimization. The process started by using a Jupyter Notebook with the Pandas Python module to clean the dataset, removing inconsistencies and errors. Afterward, an Apache Nifi ETL script was used to import the data into a data warehouse fitted with the appropriate schema and data types hosted in a local MySQL server. With the environment setup, five reports were generated by applying various OLAP operation queries to the data warehouse. Their results were represented using the Tableau Desktop app, creating an interactive visual dashboard and allowing users to view each report's specific details. Lastly, optimizations were made to the report generation queries, allowing them to run faster in different environments. This was done by applying optimizations on the database level by adding secondary indexes and MySQL system configurations, along with query restructuring, with the optimized queries being exposed to functional and performance testing to compare their results to the original unoptimized ones.

Analysis of the results among the five queries showed that secondary indexing on the accessed columns frequently improved execution time by an average of 3.5%. However, it must be noted that adding an excessive number of indices may also prove detrimental to the execution time as seen in the report.

2. Data Warehouse

A data warehouse was used to centralize the different data from the SeriousMD Appointments dataset. A data warehouse is a system that aggregates different data sources into a central repository, allowing for data organization, analysis, and mining on larger volumes [6]. In this project, the data warehouse was made in a local MySQL database server, allowing for easier and faster connections through localhost. To initialize the data warehouse, a star schema mirroring the tables within the provided dataset was created, with the appointments table being made as the fact table because it contained foreign keys to the dimensional tables. This was chosen over using a snowflake schema, as star schemas often perform better in exchange for storage. Three dimension tables are connected to the fact table: px (patients), clinics, and doctors tables. The schema model can be seen below in Figure 2.1. After creating the schema model for the dataset, MySQL’s reverse engineer tool was used to declare and initialize the schema within the data warehouse’s database.

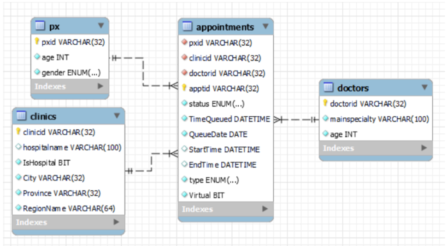


Figure 1. Schema Model of the SeriousMD Appointments Data Warehouse

2.1 Dimensional Models

Being made as the fact table, the appointments table contains a patient’s booking information, such as the time the appointment was made, appointment date, start and end time, current appointment status, type, and whether it is held virtually or not. It contains a unique primary key of “apptid” across all entries. Each appointment contains foreign keys connected to their respective dimension tables.

The dimensional tables linked to the fact table are the px (patients), clinics, and doctors tables. The px (patients) table contains a patient’s age and gender, with “pxid” as its primary key. The clinics table contains a clinic's location information, such as its hospital name, whether or not it is a hospital, and the City, Province, and Region it resides in, with "clinicid" as its primary key. Lastly, the doctors table contains information about their main specialty and age, with "doctorid" as its primary key.

Some notes about the column data types are: all primary keys are stored as VARCHAR(32) as they are all 32-bit hexadecimal values, all boolean values are stored as BIT since they are either True (1) or False (0), the ENUM data type was used for those with definite values, and VARCHARs used to store strings have variable lengths depending on the longest value of the column.

2.2 Issues Encountered

The first issue encountered while conceptualizing a model schema for the data warehouse was the size of the datasets, particularly of the px (patients) and appointments tables. The volume of both tables was too large to be displayed using common means, such as Excel and text editors. As such, the Pandas Python module had to be used to examine the datasets.

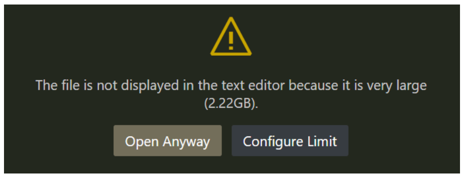

The file is not displayed in the text editor because it is very large (2.22GB).

Figure 2. Unable to Easily View the Data Due to the Large Volume of the Appointments Table

Another issue was understanding the timestamp values within the appointment table. The TimeQueued, QueueDate, StartTime, and EndTime columns had some contradicting information, such as null values being present in the columns, TimeQueued being after EndTime, QueueDate being after StartTime, and more. To resolve these issues, the entries were interpreted as logs of either previous or approaching appointments rather than future bookings, with null values representing the lack of data or that the action has not yet been finalized and that QueueDate is the check-in date of the patient for their appointment.

However, the biggest issue in the process was examining the values within the dataset itself. Many entries were filled with inconsistent, dirty, and incorrect data, making deducing the datatype for each column difficult. As such, designing the model schema had to be done in conjunction with the project's data wrangling and ETL phase.

3. ETL Script

ETL scripts, which stands for Extract, Transform, and Load scripts, is a procedure that integrates and merges data from various sources into a unified and consistent repository [4][7]. It extracts data from multiple sources, transforms them through cleaning and consolidation, and loads the tempered information into a singular system, such as a data warehouse. ETL is an essential procedure in streamlining the flow of data into the system, allowing optimizing data processing and facilitating its transformation and analysis. This is especially crucial when handling extensive datasets that undergo regular updates, such as new appointments. In the project context, ETL was used to clean and transform the data into a consistent format and then used to export the data into the data warehouse for data analysis. It happens in two steps: data wrangling using Jupyter Notebook with the Pandas Python Module and data loading using Apache Nifi to insert the data into the data warehouse.

3.1 Data Wrangling

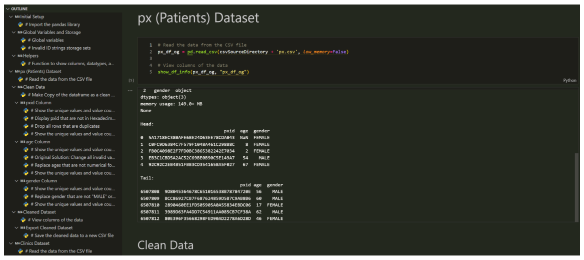
Data wrangling refers to the multitude of processes that convert raw data into usable formats for analysis, often involving merging data sources, handling missing values, and ensuring data reliability, with its ultimate goal being to prepare the data for further exploration and insights [2][13]. Given the untampered dataset containing inconsistent and dirty data, it would be infeasible to load them into the data warehouse or perform analysis without first performing data wrangling. Jupyter Notebook and the Pandas module were used for the procedure as they provided an environment to monitor and modify the current state of the data effectively as they are being processed. After data processing and cleaning, the modified datasets were exported to clean .CSV files to be loaded into the data warehouse.

The px (patients) dataset only contained 3 columns: pxid, age, and gender. After loading the dataset into the environment, functions were run to display the unique values per column, with the necessary cleaning being done for dirty columns. The pxid column contained duplicate values. As it is a primary key, only unique values are allowed; thus, the entries containing duplicate keys were dropped. The age column contained numbers below 0 and more than 1000. A patient's age can't be negative or be too high. As such, values below 0 or more than 122 had to be replaced with the -1 sentinel value. The 122 age limit was chosen based on the oldest recorded person [5]; thus, it would be highly improbable for someone to be older. However, there was an outlier entry that contained "pxid", "age", and "gender" for its respective columns. It was assumed this entry was an error and was dropped from the table.

The clinics table contained 6 columns: clinicid, hospitalname, isHospital, City, Province, and RegionName. While loading the dataset, an error occurred since it contained non-UTF8 characters such as "ñ" and "‐". This was resolved by specifying the encoding to latin1. After examining their unique values, each column was found to be relatively clean. Only some minor data representation changes had to be done to format the data to fit the database; these were: changing all non-UTF8 to their closest UTF8 counterparts to ensure correct representations ("ñ" -> "n" ; "‐ " -> "-"), and mapping the isHospital's True and False values to 1 and 0 to fit the BIT data type.

The doctors table contained 3 columns: dcotorid, mainspecialty, and age. Similar to the clinics table, it contained non-UTF8 characters and had to be loaded with latin1 encoding. The doctors table was the hardest to clean compared to the other tables due to the mainspecialty column containing over 2500+ unique values, ranging from non-medical specialties and misspellings to names and gibberish. It is not feasible to algorithmically clean mainspecialty. As such, a text file was made with the possible "valid" values, with the program replacing all non-valid entries with "General Practitioner". Afterwards, the age column was cleaned similarly to the px (patients) column's age column, but with the lower bound being 21 instead of 0, only allowing those old enough to become doctors.

Lastly, the appointments table was the largest dataset and contained 11 columns: pxid, clinicid, doctorid, apptid, status, TimeQueued, QueueDate, StartTime, EndTime, type, and Virtual. As the appointments table is the fact table and contained foreign keys, they had to be cross-checked to exist as primary keys within their respective columns. After running the cleaning, it was found that most of the appointment entries had invalid pxid keys that do not exist in the px (patients) table; the same goes for some clinicid. These rows had to be dropped as they would incur a foreign key reference error when loading into the database. After dropping the rows, the size of the appointments table shrunk dramatically, from 9752932 rows to 320140. Afterward, the timestamp columns had to be formatted to conform to MySQL's DATETIME and DATE data type, which follows the YYYY-MM-DD HH:MM:SS format [12]. This was done by removing the second's decimal field for DATETIME values (TimeQueued, StartTime, EndTime) and the time fields for DATE values (QueueDate). Following this, null TimeQueued and QueueDate rows had to be dropped as they are considered errors, and it would not make sense for them to be null if they were logged in the dataset. Meanwhile, for StartTime and EndTime, it was made valid for an entry not to have both StartTime and Endtime or only to have StartTime. This was to accommodate appointments that do not have a definite start and end time, such as for monitoring complex cases. Proceeding this, the cleaning for the rest of the columns were only minor changes for more consistent representation, such as mapping "Completed" to "Complete" for the status ENUM, and mapping True and False (null values are assumed to be False) to 1 and 0 in the Virtual column, similar to clinic table's isHospital.


OUTLINE
Initial Setup
# Import the pandas library
Global Variables and Storage
# Global variables
# Invalid ID strings storage sets
Helpers
# Function to show columns, datatypes, a...
px (Patients) Dataset
# Read the data from the CSV file
Clean Data
# Make Copy of the dataframe as a clean ...
pxid Column
# Show the unique values and value cou...
# Display pxid that are not in Hexadecim...
# Drop all rows that are duplicates
# Show the unique values and value cou...
age Column
# Show the unique values and value cou...
# Original Solution: Change all invalid va...
# Replace ages that are not numerical fo...
# Show the unique values and value cou...
gender Column
# Show the unique values and value cou...
# Replace gender that are not "MALE" or...
# Show the unique values and value cou...
Cleaned Dataset
# View columns of the data
Export Cleaned Dataset
# Save the cleaned data to a new CSV file
Clinics Dataset
# Read the data from the CSV file
px (Patients) Dataset
[5]
Python
Clean Data

```python
# Read the data from the CSV file
px_df_og = pd.read_csv(csvSourceDirectory + 'px.csv', low_memory=False)

# View columns of the data
show_df_info(px_df_og, "px_df_og")
```

```
 2   gender  object
dtypes: object(3)
memory usage: 149.0+ MB
None

Head:
                               pxid  age  gender
0  5A1718EC380AFE6BE24D63EE78CDA043  NaN  FEMALE
1  C0FC9D6384C7F579F1048A461C298B8C    8  FEMALE
2  FB0C4098E2F7FD0BC3865382242E7034    2  FEMALE
3  EB3C1CBD5A2AC52C69BE0B90C5E149A7   54    MALE
4  92C92C2EB4B51FBB3CD354165BA5F027   67  FEMALE

Tail:
                                     pxid age  gender
6507808  9D8045364678C651016538B78784720E  56    MALE
6507809  BCC86927CB7F687624859D5B7C9AB8B6  60    MALE
6507810  2890460EE1FD505905A0A55834EBDC06  17  FEMALE
6507811  3989D63FA4DD7C54911AA085CB7CF38A  62    MALE
6507812  80E396F35668298FED90AD2278A6D28D  46  FEMALE
```

Figure 3. Data Wrangling of the px (Patients) Dataset using Jupyter Notebook and Pandas.

3.2 Data Loading

Apache Nifi was used for the data loading portion of the project, loading the cleaned CSV data into the data warehouse in MySQL. The basic gist of the base process was to use the GetFile Processor to get the data from the cleaned CSV files, then use the PutDatabaseRecord Processor, configured to the MySQL local connection, to load the data into the data warehouse. Additionally, by adding the CSVReader controller, the PutDatabaseRecord processor was able to interpret the CSV data as table records directly, formatting the entries to fit the data warehouse and avoiding the use of SQL scripts, thus compacting and simplifying the overall flow. The base ETL script can be seen in Figure 3.1.

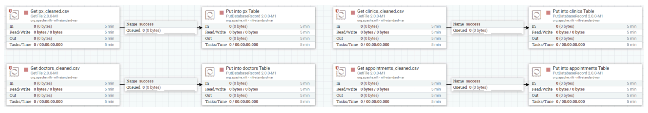


Figure 4. Basic ETL Script for Loading Data into the SeriousMD Appointments Data Warehouse

Running the base ETL script takes very long to execute due to only having a single processor load the large dataset volume into the data warehouse; this applies especially for the px (patients) table. To optimize the process, the solution was to have multiple process instances run concurrently, thus cutting the execution time by the number of PutDatabaseRecord processors currently running. This was done by splitting the records loaded from the cleaned datasets into batches of 50 records per FlowFile using the SplitRecords processor. Afterwards, they were given a route_to_path attribute using the UpdateAttribute processor, whose value is a random number between 0 and the number of paths - 1, 0 to 9 in this case. This lets the FlowFiles be evenly allocated among the paths. Lastly, the flow branched out to 10 separate RouteOnAttribute processors, each representing an individual path and only allowing FlowFiles with route_to_path values of their respective numbers. These are then connected to concurrently running PutDatabaseRecord processors, inserting the records into the data warehouse. Figure 3.2 below shows the optimized ETL script for the px (patients) table branching to 10 concurrently executing paths.

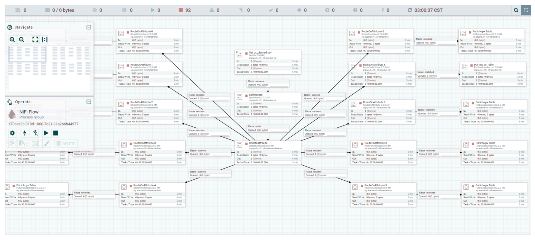


Figure 5. Optimized ETL Script for Loading the px (Patients) Dataset into the SeriousMD Appointments Data Warehouse

3.3 Issues Encountered

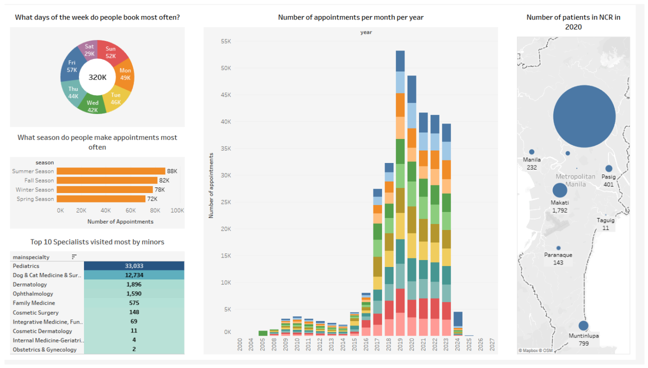
As stated before, an issue with the dataset was its sheer volume. This also applies to data wrangling, as each step to either view or modify its contents took several minutes to execute, thus delaying the cleaning process, especially for those that required trial and error. Additionally, the datasets themselves contained a lot of inconsistent and unexplained data, such as the timestamp values in appointments. This made it hard to contextualize the data for different scenarios and deduce whether a value was correct, mis-inputted, or invalid. Lastly, are the fields that are hard to clean by themselves, such as mainspecialty. There was almost no method to clean it algorithmically, and the group had to scour through its unique values and manually remove invalid ones.

Due to the thorough cleaning done during data wrangling, the ETL script phrase received no importing errors while loading data into the data warehouse. However, despite optimizations, data loading took over 20 minutes due to the dataset size. This was also why the flow's processor queues kept filling, thus bottlenecking some processors and introducing further inefficiencies in the system. Nevertheless, this was still faster than using MySQL's table import wizard, which took over 10 hours to complete.

4. OLAP Application

Online Analytical Processing (OLAP) encompasses techniques crucial for analyzing data from diverse perspectives. It involves collecting data from various sources, storing them in a data warehouse, and representing them as OLAP Cubes [1]. These cubes allow for multidimensional storage of data, enabling operations such as drill down to add more dimensions, roll up to reduce dimensions, dice to select multiple dimensions for a new sub-cube, slice to select a single dimension for a sub-cube, and pivot to rotate views for different representations [3]. OLAP facilitates insightful analysis by providing flexible ways to interact with data, making it an indispensable tool for decision-makers seeking comprehensive insights.

In the context of the application, a Tableau-based dashboard was developed. Tailored specifically for the SeriousMD Appointment dataset, the dashboard is designed to offer detailed insights into patient appointment dynamics, including the timing of appointments and the specialization areas patients visit, to make better decisions on allocating healthcare resources. Using SQL queries to perform OLAP operations, the dashboard gives insight to users regarding the dataset, allowing them to make decisions based on the knowledge gained from the reports.



mainspecialty	
Pediatrics	33,033
Dog & Cat Medicine & Sur..	12,734
Dermatology	1,896
Ophthalmology	1,590
Family Medicine	575
Cosmetic Surgery	148
Integrative Medicine, Fun..	69
Cosmetic Dermatology	11
Internal Medicine-Geriatri..	4
Obstetrics & Gynecology	2

Figure 6. Tableau OLAP Application Dashboard

The first report analyzes appointment frequencies across different days of the week, focusing on those with the highest occurrence. This analysis involves a roll-up from the queue date to the specific days of the week. A GROUP BY clause is used to group the days, while the COUNT(*) function computes the frequency of appointments for each day. The resulting data is visualized through a donut chart chosen for its ability to illustrate the relative density of appointments across different weekdays. The hole in the center denotes the total number of appointments, highlighting the distribution pattern. The report aids in identifying differences in appointment density among different days, which could assist in making staffing decisions to accommodate busier periods effectively.

Listing 1. SQL Query to Generate Which Days of the Week Do Appointments Occur Most Frequently.

```
SELECT
        DAYNAME(a.QueueDate) AS 'dayOfTheWeek',
        COUNT(*) as 'total'
FROM appointments a
GROUP BY dayOfTheWeek
ORDER BY total desc;
```

The second report depicts the number of appointments per month within each year. This analysis involves a drill-down operation on the queue date attribute to month and year. The query utilizes a GROUP BY to group the months and years, combined with the aggregate function COUNT(*), to obtain the appointments per month and per year. The presentation format chosen for this report is a stacked bar chart, chosen for its ability to visually represent the cumulative number of appointments across years while also depicting the distribution of appointments across individual months. This visualization aids in identifying patterns such as peak months, potentially attributed to seasonal illnesses, and discerning years characterized by heightened appointment activity, which may coincide with prolonged health crises such as the COVID-19 pandemic.

Listing 2. SQL Query to Generate Number of Appointments Per Month Per Year.

```
SELECT
        MONTH(a.QueueDate) AS 'month',
        YEAR(a.QueueDate) AS 'year',
        COUNT(*) AS 'appointments'
FROM appointments a
GROUP BY
        year,
        month
GROUP BY total desc;
```

The third query shows the top 10 specialists most visited by minors (age lower than 18). A slice is performed on age to obtain the sub-cube of just minors. The query utilizes a GROUP BY to group the specialties, used in combination with the aggregate function COUNT(*) to obtain the frequency of visits to doctors of certain specialties. The resulting report adopts a highlight table format, effectively representing the data akin to a ranking system. This format is preferred over a bar chart due to the significant disparities between values, which makes smaller frequencies less identifiable on a bar chart. Notably, the highlight table employs varying shades to highlight different counts of specialties, with darker hues denoting specialties with more appointments. This visualization aids in identifying specialties frequented by minors, offering insights that could inform training initiatives within these disciplines to cater to the specific healthcare needs of this demographic more effectively.

Listing 3. SQL Query to Generate Top 10 Specialists Visited Most By Minors

```
SELECT
        d.mainspecialty,
        COUNT(a.apptid) AS  'appointments'
FROM doctors d
JOIN appointments a ON d.doctorid = a.doctorid
JOIN px p ON a.pxid = p.pxid
WHERE
p.age < 18 AND p.age != -1
GROUP BY d.mainspecialty
ORDER BY appointments desc;
```

The fourth report presents data regarding the number of patients in the National Capital Region (NCR) by city during the year 2020. A dice operation is performed from the region to specifically NCR, and queue date to specifically 2020. Additionally, a drill-down operation was conducted from the region city. The query utilizes a GROUP BY to group the region names, provinces, and cities, combined with the aggregate function COUNT(*), to obtain the appointments per day of the appointments within NCR per city in 2020. The result is visualized through a symbol map, indicating the precise geographical location of each city. Each city is represented by a circle, with variations in size corresponding to the number of appointments per city. The significance of focusing on appointments in the year 2020 lies in its association with the widespread impact of the COVID-19 pandemic. By examining healthcare demand during this critical period, the report offers insights into the potential strain on healthcare systems during global health crises. Such analysis aids in better preparedness for future pandemics, enabling healthcare systems to effectively anticipate and address heightened demand.

Listing 4. SQL Query to Generate Number of patients in NCR during 2020

```
SELECT
        c.RegionName,
        c.Province,
        City,
        COUNT(*) AS 'appointments'
FROM clinics c
JOIN appointments a ON c.clinicid = a.clinicid
WHERE
        c.RegionName = 'National Capital Region (NCR)'
        AND
        YEAR(a.QueueDate) = 2020
GROUP BY
        c.RegionName,
        c.Province,
        City
ORDER BY appointments desc;
```

The final report shows the seasons where people make appointments most often. A roll-up on queue date is performed by climbing up the concept hierarchy from date to season. The query utilizes a GROUP BY to group the seasons, combined with the aggregate function COUNT(*), to obtain the count of appointments per season. CASE is also used in order to categorize the appointment dates into seasons. This is represented as a horizontal bar to make it easier to determine which seasons have more appointments or less appointments and can be ordered from most to least or least to most. While the Philippines does not have the traditional four seasons, blocking months into seasons can help identify which seasons or quarters need more attention in healthcare (such as summer and dengue).

Listing 5. SQL Query to Generate What Season Do People Make Appointments Most Often

```
SELECT
        CASE
                WHEN MONTH(a.QueueDate) BETWEEN 3
AND 5 THEN 'Spring Season'
                WHEN MONTH(a.QueueDate) BETWEEN 6
AND 8 THEN 'Summer Season'
                WHEN MONTH(a.QueueDate) BETWEEN 9
AND 11 THEN 'Fall Season'
            ELSE 'Winter Season'
        END
            AS season,
        COUNT(*) AS 'appointments'
FROM appointments a
GROUP BY
        CASE
                WHEN MONTH(a.QueueDate) BETWEEN 3
AND 5 THEN 'Spring Season'
                WHEN MONTH(a.QueueDate) BETWEEN 6
AND 8 THEN 'Summer Season'
                WHEN MONTH(a.QueueDate) BETWEEN 9
AND 11 THEN 'Fall Season'
            ELSE 'Winter Season'
        END
ORDER BY appointments desc;
```

5. Query Processing and Optimization

Query processing and optimization are the methods used to improve the performance of queries so that users can receive their desired data or results promptly by guiding the database's optimizer to select the most efficient execution plan. Its necessity is largely due to the requirement for databases to respond as quickly as possible at any moment, time being the main factor. [9][14] There are a variety of strategies that can be employed to achieve the desired results of increasing the performance of queries on the database.

Optimization strategies outside of query optimization can be broken down into 2 main categories: Hardware-level and Database-level [10]. Database-level optimizations include database design and configurations to the MySQL server to improve the way that MySQL itself executes the queries [11]. On the other hand, hardware-level optimisations are expensive, making them impractical for this occasion; some examples of this are improving the CPU to handle heavier workloads or using a modern SSD and HDD to speed up the reads from disk. [10]

In terms of queries, the first method used in improving performance is indexing, as indexes allow MySQL to quickly locate the data during queries, making it so that the query does not need to go over much of the dataset, indexes also get used in the GROUP BY and ORDER BY clauses, and on JOINs these indexes allow the server to immediately locate the required data granted the correct indexes are used in the query. [14][15].

A different optimization strategy is query restructuring, where a query is rewritten to achieve the same result with improved performance, ideally requiring fewer resources and running in a shorter timeframe. One of the notable applications of query restructuring is in the usage of subqueries or joins, as restructuring the query to use the other variant, whether changing a join into a subquery or a subquery into a join, may improve the performance greatly. Besides that, minor changes to the WHERE, GROUP BY, and ORDER BY clauses may also greatly improve. [8][15].

The database's design was covered during the ETL portion of the report. On the other hand, hardware-level optimisations are difficult, as they require a budget to implement upgrades to the system that the DBMS is running on. This leaves indexing, query restructuring, and database-level configurations as the feasible options for improving query processing and optimization.

5.1 Optimizing via Indexing

Each table in the database already has a primary index. However, introducing a secondary index may improve query speed. Therefore, indexes were created based on the needs of each query. The first query is a roll-up operation, increasing the scope of the query from individual days to days of the week. Since the query initially did not use any index during look-up, an index was added based on the QueueDate column, which is what the query is filtering for.

Listing 1. Create Indices on appointments.QueueDate

```
CREATE INDEX idx_date ON appointments (QueueDate);
```

The second query is a drill-down operation where the query is grouped by year and month, working on the same time dimension as the first query, meaning that it will now be using the created index to filter through the data.

The third query is a slice operation that acquires what specialists are most often visited by *minors*. The query is filtered using the age of people from the px table, making it the best option to add an index.

Listing 2. Create Indices on px.age

```
CREATE INDEX idx_age ON px (age);
```

The fourth query is a dice operation, where the desired regionName is specifically filtered to get the NCR region, as well as using the QueueDate to filter to the specific year 2020. This query uses the RegionName, Province, and City columns to filter the data. An index was added to each column as each has a decent amount of cardinality, indicating a potential for optimization, even though there are dangers in using too many indexes or over-indexing.

Listing 3. Create Indices on clinics.(RegionName, Province, City)

```
CREATE INDEX idx_RegionName ON clinics (RegionName);
CREATE INDEX idx_Province ON clinics (Province);
CREATE INDEX idx_City ON clinics (City);
```

The fifth and final query is a roll-up operation on QueueDate, which has already been modified to have indices from the initial Listing 1.

5.2 Optimizing via System Configurations

The MySQL server that the database is running on can be configured in a variety of ways, one of the modifications made to increase query performance was to change the *optimizer_search_depth* to 0 so the query plan would automatically pick the most reasonable plan, which could speed up the query execution time. [11]

5.3 Optimizing via Query Restructuring

For Query 1, outside of the application of indices, a subquery may be applied to create a temporary table to more easily filter values, the data values being numbers also allows for much easier indexing when applying the GROUP BY and ORDER BY clauses.

Listing 4. Query 1 Optimized Subquery

```
FROM (
        SELECT
        DAYOFWEEK(a.QueueDate) AS dayOfTheWeek
        FROM
        appointments a
) AS derived_table
```

6. Results and Analysis

Given the presented dataset, patterns of medical appointments based on different time frames were observed and analyzed. There are numerous justifications for conducting such analysis, such as:

- Resource Allocation: Understanding the distribution of appointments across different days, seasons, and time frames can help healthcare providers in allocating resources such as employees and medical supplies on peak time frames, such as in certain seasons where there is a surge of patients that can help healthcare providers prepare for an influx of patients.
- Scheduling efficiency: Analyzing appointment patterns can help optimize scheduling practices to minimize gaps between appointments and ensure that every patient can be accommodated.
- Identifying Trends and Patterns: Analyzing appointment data can help predict time frames where medical services are needed, such as seasons where different variations of diseases are most frequent

The Analysis was conducted using the SeriousMD appointment anonymized dataset imported into MySQL to analyze and perform OLAP operations on the data to draw conclusions from the information the dataset provided

The conclusions that were drawn as a result of the analysis were as follows:

- Throughout the week appointments maintained a consistent range of 42-52k appointments per day except Saturday which had an average of 29k appointments per day, almost half of the daily average of other days.
- Appointments peaked in 2019 and 2020. This could be attributed to the COVID-19 Pandemic. Since then, the number of appointments has declined in the succeeding years by approximately 35% since its peak.
- Among minors (Ages below 18), the most common doctor speciality was Pediatrics, making up roughly 70% of appointments, followed by dog and cat veterinarian services, making up roughly 25% of visits, and the remaining consisting of other services such as dermatology, cosmetic surgery etc.
- During the height of COVID-19 in 2020 in the National Capital Region (NCR) most of the appointments were from Quezon City. These made up roughly 85% of appointments in NCR for 2020, followed by Makati, making up roughly 7%. The remaining percentage comprises the remaining NCR cities, such as Muntinlupa and Manila.
- Among the four seasons, summer was found to have had the most appointments, having 88k appointments, followed by Fall (81k), then Winter(77k), and finally, Spring(72k) had the least appointments, roughly a 20% decrease from the peak season (summer).

In conclusion, the findings from this analysis provide valuable insights into healthcare utilization patterns, resource allocation strategies, and the impact of external factors such as pandemics on healthcare delivery. This research underscores the importance of data-driven decision-making in healthcare management and the ongoing need for adaptive strategies to meet evolving healthcare needs effectively.

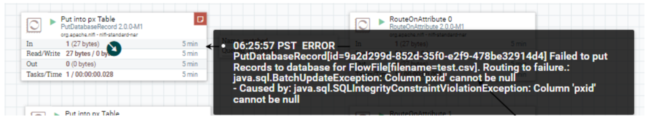
6.1 Functional Testing

To validate the correctness of the ETL process, the following functional tests were conducted:

Data Integrity Testing: To ensure that the transfer of data kept the data constraints such as uniqueness, referential integrity and quality, a test CSV file was created containing a sample of rows mirroring the format of the actual SeriousMD datasets and its tables; however, it had incorrect values with respect to the attributes set in the data warehouse. From here, various test cases were performed to test its data integrity:

Table 1. Test Case 1

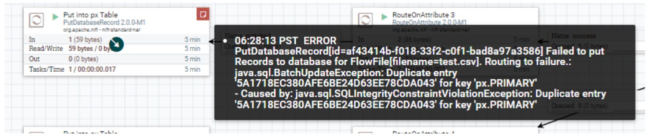
Test Description	Expected Result	Actual Result
Test of primary key uniqueness and non-nullable attribute	Throw an error and not load the value in the data warehouse	Threw an error and did not load the value in the data warehouse



```
06:25:57 PST  ERROR
PutDatabaseRecord[id=9a2d299d-852d-35f0-e2f9-478be32914d4] Failed to put
Records to database for FlowFile[filename=test.csv]. Routing to failure.:
java.sql.BatchUpdateException: Column 'pxid' cannot be null
- Caused by: java.sql.SQLIntegrityConstraintViolationException: Column 'pxid'
cannot be null
```

Put into px Table
PutDatabaseRecord 2.0.0-M1
org.apache.nifi - nifi-standard-nar
In 1 (27 bytes) 5 min
Read/Write 27 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 1 / 00:00:00.028 5 min
RouteOnAttribute 0
RouteOnAttribute 2.0.0-M1

Figure 6. Failing test case for null primary key in PX table



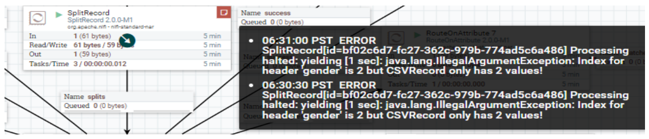
```
06:28:13 PST  ERROR
PutDatabaseRecord[id=af43414b-f018-33f2-c0f1-bad8a97a3586] Failed to put
Records to database for FlowFile[filename=test.csv]. Routing to failure.:
java.sql.BatchUpdateException: Duplicate entry
'5A1718EC380AFE6BE24D63EE78CDA043' for key 'px.PRIMARY'
- Caused by: java.sql.SQLIntegrityConstraintViolationException: Duplicate entry
'5A1718EC380AFE6BE24D63EE78CDA043' for key 'px.PRIMARY'
```

Put into px Table
PutDatabaseRecord 2.0.0-M1
org.apache.nifi - nifi-standard-nar
In 1 (59 bytes) 5 min
Read/Write 59 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 1 / 00:00:00.017 5 min
RouteOnAttribute 3
RouteOnAttribute 2.0.0-M1
Name success

Figure 7. Failing test case for duplicate primary key in PX table

Table 2. Test Case 2

Test Description	Expected Result	Actual Result
Test for non-integer value in an integer column	Throw an error and not load the value in the data warehouse	Threw an error and did not load the value in the data warehouse

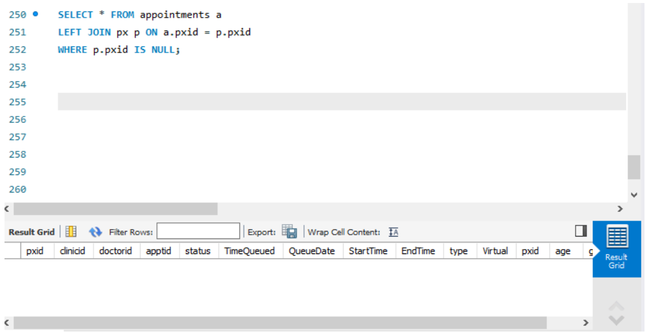


```
06:31:00 PST  ERROR
SplitRecord[id=bf02c6d7-fc27-362c-979b-774ad5c6a486] Processing
halted: yielding [1 sec]: java.lang.IllegalArgumentException: Index for
header 'gender' is 2 but CSVRecord only has 2 values!
06:30:30 PST  ERROR
SplitRecord[id=bf02c6d7-fc27-362c-979b-774ad5c6a486] Processing
halted: yielding [1 sec]: java.lang.IllegalArgumentException: Index for
header 'gender' is 2 but CSVRecord only has 2 values!
```

Figure 8. failing test case for non-integer value in age column in PX table

Table 3. Test Case 3

Test Description	Expected Result	Actual Result
Test for referential integrity between tables (foreign key), using an SQL script to check for any primary keys tha	Give an empty query result because all foreign keys exist in its referencing table	Gave an empty query result



```
SELECT * FROM appointments a
LEFT JOIN px p ON a.pxid = p.pxid
WHERE p.pxid IS NULL;
```

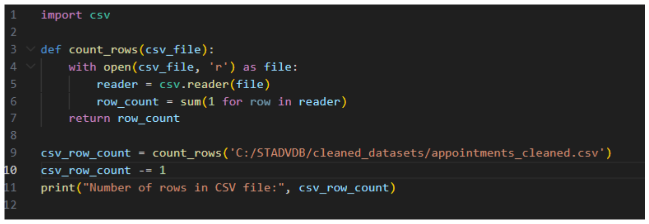
pxid	clinicid	doctorid	apptid	status	TimeQueued	QueueDate	StartTime	EndTime	type	Virtual	pxid	age

Figure 9. Referential Integrity test query and its result from joining px and appointments table on its foreign key pxid

Data Transfer Validation: To ensure that data was transferred correctly from the CSV files to the data warehouse without missing values, data comparison was conducted to validate the transfer process of the ETL.

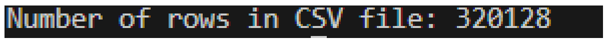
Table 4. Test Case 4

Test Description	Expected Result	Actual Result
Run row counter test file for CSV file, then run SQL query to count rows, then compare both results.	Both values should be the same from the CSV and MySQL results	Both values from the CSV and MySQL results were identical



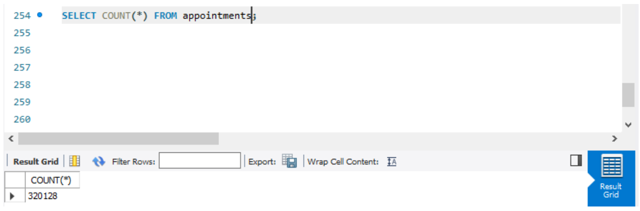
```python
import csv

def count_rows(csv_file):
    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        row_count = sum(1 for row in reader)
    return row_count

csv_row_count = count_rows('C:/STADVDB/cleaned_datasets/appointments_cleaned.csv')
csv_row_count -= 1
print("Number of rows in CSV file:", csv_row_count)
```

Figure 10. Python code to count the number of rows in CSV file



```
Number of rows in CSV file: 320128
```

Figure 11.: Result of Python code on the CSV file appointments



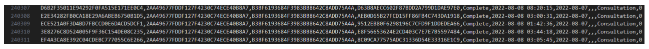
```
SELECT COUNT(*) FROM appointments;
```

```
COUNT(*)
320128
```

Figure 12. SQL query to count the rows of a table from the data warehouse and its result

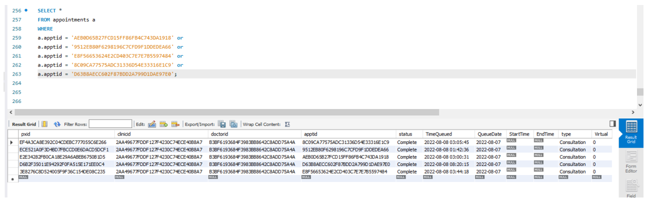
Table 5. Test Case 5

Test Description	Expected Result	Actual Result
Randomly select a sample of rows in CSV file and compare the selected rows to its corresponding rows in MySQL using a script to retrieve the specified rows	Both rows should be the same from the CSV and MySQL results	Both rows from the CSV and MySQL results were identical



```
D6B2F35011E94292F0FA515E171EE0C4,2AA49677FDDF127F4230C74ECE40B8A7,B3BF6193684F3983BB8642C8ADD75A4A,D63B8AECC602F87BDD2A799D1DAE97E0,Complete,2022-08-08 08:20:15,2022-08-07,,,Consultation,0
E2E34282FB0CA18E29A6ABEB6750B1D5,2AA49677FDDF127F4230C74ECE40B8A7,B3BF6193684F3983BB8642C8ADD75A4A,AEB0D65B27FCD15FF86FB4C743DA1918,Complete,2022-08-08 03:00:31,2022-08-07,,,Consultation,0
ECE521A0F3D4BD7FBCCD0E6DACD5DCF1,2AA49677FDDF127F4230C74ECE40B8A7,B3BF6193684F3983BB8642C8ADD75A4A,9512EB80F6298196C7CFD9F1DDEDEA66,Complete,2022-08-08 01:42:36,2022-08-07,,,Consultation,0
3E8276C8D524005F9F36C154DE08C235,2AA49677FDDF127F4230C74ECE40B8A7,B3BF6193684F3983BB8642C8ADD75A4A,E8F56653624E2CD403C7E7E7B5597484,Complete,2022-08-08 03:44:18,2022-08-07,,,Consultation,0
EF4A3CA8E392C04CDEBC777055C6E266,2AA49677FDDF127F4230C74ECE40B8A7,B3BF6193684F3983BB8642C8ADD75A4A,8C09CA77575ADC31336D54E33316E1C9,Complete,2022-08-08 03:05:45,2022-08-07,,,Consultation,0
```

Figure 13. Sample of randomly selected rows in appointments CSV file



```
SELECT *
FROM appointments a
WHERE
a.apptid = 'AEB0D65B27FCD15FF86FB4C743DA1918' or
a.apptid = '9512EB80F6298196C7CFD9F1DDEDEA66' or
a.apptid = 'E8F56653624E2CD403C7E7E7B5597484' or
a.apptid = '8C09CA77575ADC31336D54E33316E1C9' or
a.apptid = 'D63B8AECC602F87BDD2A799D1DAE97E0';
```

```
pxid                              clinicid                          doctorid                          apptid                            status    TimeQueued           QueueDate   StartTime  EndTime  type          Virtual
EF4A3CA8E392C04CDEBC777055C6E266  2AA49677FDDF127F4230C74ECE40B8A7  B3BF6193684F3983BB8642C8ADD75A4A  8C09CA77575ADC31336D54E33316E1C9  Complete  2022-08-08 03:05:45  2022-08-07  NULL       NULL     Consultation  0
ECE521A0F3D4BD7FBCCD0E6DACD5DCF1  2AA49677FDDF127F4230C74ECE40B8A7  B3BF6193684F3983BB8642C8ADD75A4A  9512EB80F6298196C7CFD9F1DDEDEA66  Complete  2022-08-08 01:42:36  2022-08-07  NULL       NULL     Consultation  0
E2E34282FB0CA18E29A6ABEB6750B1D5  2AA49677FDDF127F4230C74ECE40B8A7  B3BF6193684F3983BB8642C8ADD75A4A  AEB0D65B27FCD15FF86FB4C743DA1918  Complete  2022-08-08 03:00:31  2022-08-07  NULL       NULL     Consultation  0
D6B2F35011E94292F0FA515E171EE0C4  2AA49677FDDF127F4230C74ECE40B8A7  B3BF6193684F3983BB8642C8ADD75A4A  D63B8AECC602F87BDD2A799D1DAE97E0  Complete  2022-08-08 08:20:15  2022-08-07  NULL       NULL     Consultation  0
3E8276C8D524005F9F36C154DE08C235  2AA49677FDDF127F4230C74ECE40B8A7  B3BF6193684F3983BB8642C8ADD75A4A  E8F56653624E2CD403C7E7E7B5597484  Complete  2022-08-08 03:44:18  2022-08-07  NULL       NULL     Consultation  0
NULL                              NULL                              NULL                              NULL                              NULL      NULL                 NULL        NULL       NULL     NULL          NULL
```

Figure 14. SQL query to retrieve the specified rows using its primary key (appt.id) and its query result

6.2 Performance Testing

These tests were performed to ascertain the performance of the queries in their pre-optimized and post-optimized states. The tests should show the significance of the differences between the queries.

6.2.1 Performance of All Queries Post Optimization

Each query version was run 5 times, with each table containing the average of the 5 trials starting from cold cache, ensuring no data was in the buffer pool.

For the first query, it can be seen that introducing a secondary index helped immensely, increasing performance by approximately 11.3% compared to the original query. This improvement can be credited to the added index, which helped MySQL look up the required data, as well as the optimized subquery.

Table 1. Performance Results for all Queries.

Survey Item	Base Query	Optimized	Optimization %
Query #1	0.106 sec	0.094 sec	88.7%
Query #2	0.172 sec	0.147 sec	85.5 %
Query #3	35.975 sec	35.809 sec	99.5%
Query #4	0.194 sec	0.219 sec	112.9%
Query #5	0.178 seec	0.172 sec	96.6%

Query 2 also showed a striking improvement, increasing performance by 14.5%, hinting at the fact that the indices on QueueDate were used to filter through the rows in the database more easily.

On the other hand, Query 3 showed negligible improvement, with a measly 0.5% increase in performance. This is most likely due to the requirement for a full table scan during the execution of the query, which is burdensome as the optimizer does not use an index for the lookup. A possible way to improve the query's performance would be to add another condition in the WHERE clause to reference a possible index, such as a specific pxid or doctorid.

Query 4 is a different case where the optimized query performed worse than the original query by -12.9%. The increase in query time is due to adding extra indices used to filter and compare the rows, despite the PRIMARY index that already existed before being sufficient enough to filter through the data. Removal of some of the added indices may benefit this query.

Query 5 saw a minimal increase in performance, with only a 3.4% increase in speed. The difference is so minimal that it could have been only due to the device's state when the tests were performed, making the increase extremely negligible.

7. Conclusion

The paper described the process of building an OLAP application that leverages a data warehouse populated from the anonymized SeriousMD Appointment Dataset using MySQL queries, with the dataset first being cleaned and imported using ETL scripts. The OLAP application was designed to analyze and perform OLAP operations on the appointment data to draw insights into healthcare utilization patterns, resource allocation strategies, and the impact of external factors such as pandemics on healthcare delivery.

Results showed that the OLAP application provided valuable insights into appointment distribution across different days, seasons, and time frames. It facilitated the identification of peak appointment periods, such as during the COVID-19 pandemic, enabling healthcare providers to allocate resources effectively and optimize scheduling practices to accommodate patient needs efficiently.

The analysis helped the group gain insights into the importance of building and maintaining a data warehouse in healthcare management. A well-designed data warehouse serves as a central repository for structured data, facilitating efficient data analysis and decision-making processes. The Extract, Transform, Load

(ETL) process plays a crucial role in keeping the warehouse updated by extracting data from multiple sources, transforming it into a consistent format, and loading it into the warehouse for analysis.

Aside from OLTP, the primary reason for conducting OLAP is to analyze historical data and identify trends, patterns, and relationships that can inform decision-making processes. OLAP enables users to perform complex analytical queries across large datasets efficiently.

Optimizing queries is essential for improving the performance of the OLAP application. Strategies such as indexing, query optimization techniques, and database schema design optimization can help enhance query performance and reduce response times, ensuring timely access to insights.

In some situations, creating custom indexes may be necessary to optimize query performance further. This could occur when certain queries require access to specific subsets of data that are not adequately addressed by the indexes automatically generated by the database management system. In such cases, creating custom indexes tailored to the query requirements can significantly improve performance.

8. References

[1] Biscobing, J. (2023, October). What is OLAP (online analytical processing)?: Definition from TechTarget. Data Management. https://www.techtarget.com/searchdatamanagement/definition/OLAP

[2] Coursera. (2023, December 2). What is data wrangling? definition, steps, and why it matters. https://www.coursera.org/articles/data-wrangling

[3] GeeksforGeeks. (2019, August 19). OLAP operations in DBMS. https://www.geeksforgeeks.org/olap-operations-in-dbms/

[4] GeeksforGeeks. (2023, February 2). ETL process in Data Warehouse. https://www.geeksforgeeks.org/etl-process-in-data-warehouse/

[5] Guinness World Records. (n.d.). Oldest person ever. https://www.guinnessworldrecords.com/world-records/oldest-person/

[6] IBM. (n.d.-a). What is a data warehouse? https://www.ibm.com/topics/data-warehouse

[7] IBM. (n.d.-b). What is ETL (extract, transform, load)? https://www.ibm.com/topics/etl

[8] Lee, A. (2022, December 19). SQL: Subqueries VS join. Medium. https://adamtlee.medium.com/sql-subqueries-vs-join-9bcb921a5b2e

[9] Macrometa. (n.d.). How does query optimization work? https://www.macrometa.com/articles/how-does-query-optimization-work

[10] MySQL. (n.d.-a). MySQL 8.0 Reference Manual :: 10.1 optimization overview. https://dev.MySQL.com/doc/refman/8.0/en/optimize-overview.html

[11] MySQL. (n.d.-b). MySQL 8.0 Reference Manual :: 10.9.1 controlling query plan evaluation. https://dev.MySQL.com/doc/refman/8.0/en/controlling-query-plan-evaluation.html

[12] MySQL. (n.d.-c). MySQL 8.0 Reference Manual :: 13.2.2 the date, DATETIME, and timestamp types. https://dev.MySQL.com/doc/refman/8.0/en/datetime.html

[13] Stobierski, T. (2021, January 19). Data wrangling: What it is & why it’s important. Harvard Business School Online. https://online.hbs.edu/blog/post/data-wrangling

[14] Tutorialspoint. (n.d.). What is query optimization? https://www.tutorialspoint.com/what-is-query-optimization

[15] Zawodny, J. D., Balling, D. J., Schwartz, B., Zaitsev, P., Lentz, A., & Tkachenko, V. (n.d.). High performance MySQL, 2nd edition. O’Reilly Online Learning. https://www.oreilly.com/library/view/high-performance-MySQL/9780596101718/ch04.html

[16] Oakley, V. Nisi (eds). ICIDS 2017. Lecture Notes in Computer Science, 10690. Springer, Cham, 290-294.

9. Declarations

9.1 Declaration of Generative AI in Scientific Writing

During the preparation of this work the author(s) used ChatGPT in order to structure compositions together. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

9.2 Record of Contribution

Concio, Tean Jeremy Concio.

- Modeled the SeriousMD data warehouse schema
- Cleaned the dataset with Jupyter Notebook and Pandas
- Created the Apache Nifi ETL Script
- Wrote the corresponding parts in the report
- Wrote the Introduction part of the report
- Formatted the report

Ong, Nicole Daphne

- Developed the OLAP application
- Wrote the corresponding part in the report
- Formatted the references

Raphael, Tan Ai Jeremiah

- Wrote the Query Processing and Optimization
- Wrote the performance testing section

Chua, Matthew Adrian

- Created all SQL queries
- Wrote the Results and Analysis section
- Conducted functional testing on the ETL process and the OLAP application
- Wrote the functional testing of ETL process and OLAP application section
- Wrote the conclusion section