A Technical Report on Query Processing in a Data Warehouse

Sealtiel B. Dy
sealtiel_dy@dlsu.edu.ph
De La Salle University
Manila, Philippines

Kyle Carlo C. Lasala
kyle_lasala@dlsu.edu.ph
De La Salle University
Manila, Philippines

Maria Monica Manlises
maria_monica_manlises@dlsu.edu.ph
De La Salle University
Manila, Philippines

Camron Evan C. Ong
camron_evan_ong@dlsu.edu.ph
De La Salle University
Manila, Philippines

ABSTRACT

This paper explores the integration of digital technology in the healthcare industry, focusing on the analysis of electronic health records (EHRs) from SeriousMD, a telemedicine company. A data warehouse was designed having a dimensional model of a star schema with one fact table and three dimension tables. The dataset underwent Extract, Transform, Load (ETL) processes, where Python scripts, addressed data quality issues and involved rule-based cleaning. Python scripts were also used to load the data to the designed data warehouse. To explore trends in the EHRs, queries were created via SQL and visualized using Tableau as the Online Analaytical Processing (OLAP) application. The study also delves into query processing optimization, employing horizontal partitioning based on key criteria like appointment time and clinic location. Additionally, indexing was implemented to enhance query performance for specific `WHERE` clauses. Performance testing revealed improved query speeds, demonstrating the significance of these optimization techniques. The paper contributes insights into leveraging digital data analyses tools for healthcare data management, providing practical benefits for decision-makers in the industry.

KEYWORDS

data warehouse, OLAP, ETL, query processing

1 INTRODUCTION

The healthcare business is undergoing a significant shift by incorporating digital technology to improve patient care and streamline medical procedures. Electronic health records (EHRs) are essential for improving the efficiency and comprehensiveness of healthcare management [5]. SeriousMD is a telemedicine firm leading the way in revolutionizing healthcare by offering a digital platform to help doctors efficiently manage patient records [6].

The research involves analyzing a large anonymized dataset from SeriousMD, which includes EHRs from various healthcare settings. The dataset was subjected to a number of Extract, Transform, Load (ETL) processes to prepare it for analysis, enabling the extraction of significant insights on trends, patterns, and critical aspects of patient care.

Tableau was used to construct an Online Analytical Processing (OLAP) application to help users explore the dataset easily and gain practical insights. This application functions as a potent tool for healthcare professionals and decision-makers, allowing them to visually examine the available records. The OLAP program uses data visualizations to offer a detailed overview of trends in appointment times, appointment locations, and more, enabling healthcare practitioners to make well-informed decisions.

2 DATA WAREHOUSE

The design of the warehouse largely followed the structure of the source files. This resulted in the dimensional model of a star schema. The fact table of Appointments contains the reference IDs of the other dimensions. This includes `pxid`—patients' identifiers, `clinicid`—clinics' identifiers, `doctorid`—doctors' identifiers. Along with those referential identifiers are attributes related to the appointment itself. This includes the status of the appointment, which can be of values `queued`, `complete`, `serving`, `cancel`, `noshow`, or `skip`. It also includes `datetime` data on the time queued, queue date, start time, and end time. The type of appointment is also indicated as either `consultation` or `inpatient`. Lastly, this fact table includes `boolean` information on whether it was held virtually or not.

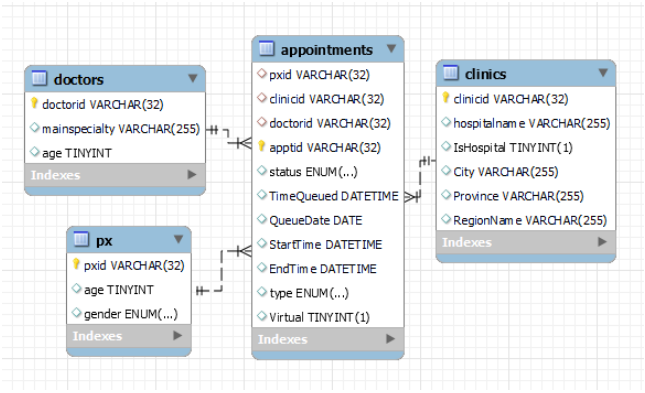


Figure 1: Database Model

The dimension table `doctors` contains the age of the doctor and their main specialty. Similarly, the dimension table `px` contains the age of the patient and their gender. Finally, the `clinics` dimension table holds data related to where the appointment took place, including the name of said clinic or hospital, its location, and whether the clinic is a hospital or not. The location is separated into the city, province, and region, which can be used for online analytical processing (OLAP) operations since it contains both the more specific city location and broader region location. This information

can help analyze potential geographical hot spots. The star schema is shown in Figure 1.

It should also be noted that a patient can have multiple appointments, doctors can handle multiple appointments, and clinics can be the venue for multiple appointments as well. Conversely, an appointment can only have one of each (patient, doctor, and clinic).

As such, `appointments` was chosen as the fact table as each row is equivalent to one transaction that references the other tables. Furthermore, the majority of the analysis will be done on information about the appointments.

3 ETL SCRIPT

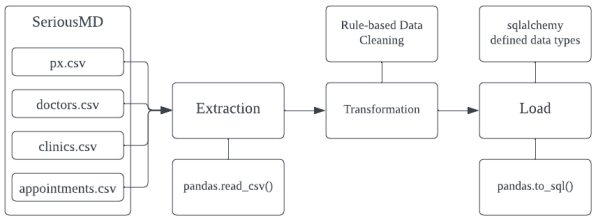


Figure 2: ETL Pipeline

ETL scripts were made on a Jupyter Notebook using Python libraries. One ETL pipeline shown in Figure 2 was created and is subdivided into the processes of extracting, transforming, and loading each table.

3.1 Extraction

Importing the `pandas` library is essential prior to extraction. The `appointments.csv`, `clinics.csv`, `doctors.csv`, and `px.csv` files are the source dataset. The contents of the four files are read and saved in separate `pandas.DataFrames` using the `pandas.read_csv` function. To read the `clinics.csv`, `doctors.csv`, and `px.csv` the `encoding` parameter was set to `unicode_escape` as not all characters in these files could be read using the default `encoding="utf-8"`. The code snippet for extraction can be seen in Listing 1.

Listing 1: Data Extraction from CSV Files

```
appointments =
     pd.read_csv('smd-ns-appointments/appointments.csv')
clinics = pd.read_csv('smd-ns-appointments/clinics.csv',
     encoding='unicode_escape')
doctors = pd.read_csv('smd-ns-appointments/doctors.csv',
     encoding='unicode_escape')
px = pd.read_csv('smd-ns-appointments/px.csv',
     encoding='unicode_escape')
```

3.2 Transformation

After extracting the data, it was found that the DataFrame corresponding to the fact table, `appointments`, initially contained 9 million rows. As such, pre-processing was done across all tables as part of the transformation process before loading the data into the destination data warehouse. This was done on the dimensions tables first which will be used as reference for the fact table.

3.2.1 Clinics Dimension Table. Previewing the information about `clinics`, it was found that there were six attributes as follows:

- `clinicid` - `string` containing no `null` values
- `hospitalname` - `string`
- `IsHospital` - `boolean` containing no `null` values
- `City` - `string` containing no `null` values
- `Province` - `string` containing no `null` values
- `RegionName` - `string` containing no `null` values

The `clinicid` column contained no duplicate values and were all 32 characters long. There were duplicate values found under `hospitalname` but were not removed as they had different corresponding `clinicid` values. The rest of the attributes had all valid values. As such, no cleaning was done on `clinics`.

3.2.2 Patients Dimension Table. The px dataframe originally contained the following three attributes:

- `pxid` - `string` containing no `null` values
- `age` - `string`
- `gender` - `string` containing no `null` values

One of the values under the `gender` column was `gender`. As such, the row containing this value was dropped.

Since age is numerical, the `age` column was converted to `float`. It was not converted to `integer` as this data type is not compatible with `null` values. Aside from this, another issue with the `age` column was that there were negative values and really large values. Since the oldest person alive is 115 years old as of January 2023 [2], it was decided to keep the range of values between 0 and 120, inclusive. All negative values were set to 0 while all values greater than 120 were set to 120.

After cleaning the `age` column, the values were used as reference for cleaning the `pxids`. Since there were duplicate values detected under `pxid`, the row with the largest age for that `pxid` was kept. This assumes that a patient has consulted multiple times over the span of years and their oldest age is kept in the database. It is also important to note that each `pxid` was 32 characters long.

3.2.3 Doctors Dimension Table. The `doctors` dataframe also contained three attributes:

- `doctorid` - `string` containing no `null` values
- `mainspecialty` - `string`
- `age` - `float`

Similar to `px`, the `age` under `doctors` were limited to values between 0 and 120. No duplicate `doctorids` were found and each were 32 characters long. The `specialty` required the most cleaning as similar specialties had inconsistent wording and formatting. Some specialties were very specific while some were very broad. Furthermore, some rows contained multiple specialties. As such, only the first specialty of each row was considered and rule-based data cleaning was done to make the labels uniform. Initially, the following rules were applied:

- All values were set to lowercase
- All '&'s were replaced with 'and's
- Extra spaces in the beginning and end were removed

Afterwards, rules were used to label similar specialties with one another. These were based on online medical references on doctor specialties [7] and hospital departments[1]. One set of these rules for "internal medicine" are available in Table 1. The rest can be seen in Appendix B. The values of the specialties that could not be inferred or were ambiguous were set to none.

Table 1: Rules for Internal Medicine

Internal Medicine	- strings that contain "internal med" in the first 12 characters - strings equal to "internal" - strings that contain {"im ", "im-", "im -"} in the first 4 characters but not including the strings {"im not", "im trained"}

3.2.4 Appointments Fact Table. The appointments contains 11 attributes as follows:

- pxid - string containing no null values
- clinicid - string containing no null values
- doctorid - string containing no null values
- apptid - string containing no null values
- status - string containing no null values
- TimeQueued - string containing no null values
- QueueDate - string
- StartTime - string
- EndTime - string
- type - string containing no null values
- Virtual - boolean

The TimeQueued, QueueDate, StartTime, and EndTime were converted to datetime. Meanwhile, status was converted to an enum() with values "queued", "complete", "serving", "cancel", "noshow", and "skip". Similarly, type was converted to an enum() with values "consultation", and "inpatient".

The referenced foreign keys pxid, clinicid, and doctorid were then cross-referenced to the corresponding dimension tables. The rows with foreign keys not present in the dimension tables were then removed.

3.2.5 Issues Encountered During Transformation. In preprocessing, data quality issues were a significant concern. The checking of anomalous data, null values, and appropriate datatypes took a few days because the values were nonstandardized and inconsistent. The usage of rule-based data cleaning sped up the process as Python scripts were utilized to clean the data based on inferred rules.

3.3 Loading

After pre-processing all of the data, the data was then imported into the Data Warehouse in MySQL via the Python library SQLAlchemy. Similar to how this would be done in Apache NiFi, the first step would be connecting the Notebook to MySQL, which can be done using an engine containing the connection string.

The cleaned data was then all loaded back into their respective DataFrames, namely appointments, px, doctors, and clinics.

The next step was creating the SQL schemas in Python. To define the SQL schema, a key-value pair is used where the key is the column name and the value is the MySQL data type. In defining the MySQL data type, the library SQLAlchemy contains ready-made MySQL data types. This turned out to be relatively simple as the syntax for creating schemas in SQLAlchemy was similar to that of MySQL.

Primary Keys (PK) and Foreign Keys (FK) were defined using a Python script after creating the schemas, based on the ID attributes in each column. As mentioned in the Data Warehouse Section, the keys are as follows:

- Px: pxid (PK)
- Doctors: doctorid (PK)
- Clinics: clinicid (PK)
- Appointments: apptid (PK), pxid (FK), doctorid (FK), and clinicid (FK)

Following this was importing all of the already-structured data into the warehouse through the use of the Pandas DataFrame to_sql method. This converts the entire DataFrame into SQL syntax based on the schema that was indicated in the function parameters. The Python script can be seen in Listing 2.

Listing 2: Loading Data to Warehouse

```
appointments.to_sql('appointments', engine,
     if_exists='replace', index=False,
     dtype=appointments_schema)
doctors.to_sql('doctors', engine, if_exists='replace',
     index=False, dtype=doctors_schema)
clinics.to_sql('clinics', engine, if_exists='replace',
     index=False, dtype=clinics_schema)
px.to_sql('px', engine, if_exists='replace', index=False,
     dtype=px_schema)
```

After importing, the number of rows in each table in the warehouse matched the originals in the DataFrames and was presumed to have been imported correctly. Python would also warn if any errors or incorrect data types were being imported, hence the decision.

3.3.1 Issues Encountered During Loading. In loading the data to the warehouse, defining the schema was an issue. Since there was no well-defined metadata from the extraction, metadata was assumed based on data cleaning. The data also had to be consistent prior to loading to the warehouse because small inconsistencies would throw an error and disrupt the inserting process. The utilization of the Pandas and SQLAlchemy libraries simplified the process. In total, inserting took under 5 minutes as the MySQL server needed to process a large volume of data.

4 OLAP APPLICATION

The OLAP Application utilized in this project is Tableau, which is a data visualization application that has database integration for data retrieval. Tableau is capable of generating various types of graphs and illustrations based on provided data. The data may be imported from different types of sources, such as files and database queries. However, some integrations, such as the MySQL Database Server

integration, require the installation of their MySQL Connector Driver in order to function properly.

For this project, data was imported into Tableau from a local MySQL Database Server by using multiple data sources with custom queries. The main purpose of the application is to visualize trends concerning the demographics pertaining to location, time, age, and virtual consultations associated with the appointments. The includes the following analyses: Amount of clinics per city and per province; Quarterly and monthly morning appointments; Appointments concerning internal medicine; And virtual appointments of senior citizens. The visualizations of these analytical reports are in Appendix A.

The first analytical report concerns the amount of clinics per province which is achieved with the query in Listing 3.

Listing 3: Amount of Clinics per Province

```
SELECT     COUNT(c.clinicid), c.Province
FROM       appointments a
           JOIN clinics c ON a.clinicid = c.clinicid
GROUP BY   c.Province
ORDER BY   c.Province;
```

In order to obtain the amount of clinics per city while maintaining the count for provinces, the query may be modified with the ROLLUP keyword to perform a roll up OLAP operation, shown in Listing 4. The results of these queries were visualized in Tableau as a map with colored regions for the provinces and colored dots for the cities.

Listing 4: Amount of Clinics per Province per City

```
SELECT     COUNT(c.clinicid), c.City, c.Province
FROM       appointments a
           JOIN clinics c ON a.clinicid = c.clinicid
GROUP BY   c.Province, c.City WITH ROLLUP
ORDER BY   c.Province;
```

The next analytical report is about morning appointments, where the query obtains quarterly data about morning appointments. The query is stated in Listing 5.

Listing 5: Morning Appointments per Quarter

```
SELECT     COUNT(a.apptid) AS apptCount,
           YEAR(a.TimeQueued) AS Yr,
           QUARTER(a.TimeQueued) AS Qtr
FROM       appointments a
           JOIN clinics c ON a.clinicid = c.clinicid
WHERE      TIME(a.TimeQueued) < '12:00:00'
           AND c.IsHospital = 1
GROUP BY   YEAR(a.TimeQueued), QUARTER(a.TimeQueued)
ORDER BY   Yr, Qtr;
```

This query can be further refined using a drill down OLAP operation in order to show monthly series of results instead of quarterly, as shown in the query in Listing 6. Both of these time series data were visualized using a line graph in Tableau.

Listing 6: Morning Appointments per Month

```
SELECT     COUNT(a.apptid) AS apptCount,
           YEAR(a.TimeQueued) AS Yr,
           MONTH(a.TimeQueued) AS Mnth
FROM       appointments a
           JOIN clinics c ON a.clinicid = c.clinicid
WHERE      TIME(a.TimeQueued) < '12:00:00'
           AND c.IsHospital = 1
GROUP BY   YEAR(a.TimeQueued), MONTH(a.TimeQueued)
ORDER BY   Yr, Mnth;
```

The third analytical report shows the number of appointments that are categorized under internal medicine. The SQL query for this is available in Listing 7. This is a slice OLAP operation where only internal medicine appointments are needed apart from all types of appointments. The results of this query are depicted in Tableau as a pie chart where the amount of appointments in a hospital is segregated from non-hospital appointments.

Listing 7: Internal Medicine

```
SELECT     COUNT(a.apptid) AS apptCount,
           a.doctorid AS did,
           d.mainspecialty,
           c.IsHospital
FROM       appointments a
           JOIN clinics c ON a.clinicid = c.clinicid
           JOIN doctors d ON a.doctorid = d.doctorid
WHERE      d.mainspecialty = 'internal medicine'
GROUP BY   a.doctorid, d.mainspecialty, c.isHospital;
```

The final analytical report is about the number of senior citizens that had virtual appointments, per city. This is a dice OLAP operation since this is a smaller selection of results from all patients and appointments, in all cities. The SQL query for this report is seen in Listing 8. Similar to the first analytical report, the visualization for the results of this report is a Map in Tableau, where each city is a dot, with appointment counts as its label.

Listing 8: Virtual Appointments of Senior Citizen

```
SELECT     ID, City
FROM   (   SELECT     px.pxid AS ID,
                      px.age AS Age,
                      COUNT(px.pxid) AS ConsultTimes,
                      c.city AS City
           FROM       appointments a
                      JOIN px ON a.pxid = px.pxid
                      JOIN  clinics c ON a.clinicid =
                           c.clinicid
           WHERE   px.age > 60 AND a.Virtual = 1
           GROUP BY   a.pxid, c.city, px.age
           ) as SubQ
WHERE      ConsultTimes = 1
ORDER BY   ID;
```

5 QUERY PROCESSING AND OPTIMIZATION

The performance of the base queries may be further improved through a variety of means. Particularly, Partitioning and Indexing were chosen as the optimization strategies for this project.

5.1 Partitioning

Partitioning is a data de-normalization technique that can refer to two things: splitting vertically and horizontally. Vertical Partitioning is when a table is split up into multiple tables based on the column, such as splitting less commonly accessed attributes across multiple tables. On the other hand, Horizontal Partitioning is splitting the rows (tuples) based on a certain condition, and grouping them into a smaller 'sub-table' or 'partition' [3]. Given the structure of the tables in the data warehouse, specifically the number of attributes and tuples per table, horizontal partitioning was chosen as it appeared to fit the requirements better than vertical.

Partitioning can improve the performance of queries, particularly in MySQL, because it allows for computing techniques such as parallel processing and pruning and also allows for reductions in data volume. Parallel processing refers to how multiple subprocesses can be handled at the same time in order to complete a single, larger process faster. Pruning is ignoring the irrelevant sub-tables as the DBMS would already identify them to not fit in the constraint. Finally, reduced data volume is achieved since the tables being worked on are inherently smaller, simply due to the definition of this optimization strategy [4].

Given that the SQL query performance could potentially be improved, the tables were partitioned based on the main requirements of the query. Before that, it was found that MySQL had restrictions on partitioning, namely 1) Primary Keys must be included as the partitioning key, but was also bugging when attempted, and 2) Partitioning is actually unavailable for tables with Foreign Keys. Given this, after checking for permission, the PK and FK constraints were removed to allow partitioning. The partition criteria are as follows:

- Appointments: `TimeQueued` year
- Clinics: `Province` where the clinic is located
- Doctors: `MainSpecialty` of the doctor as identified and binned in the data pre-processing stage
- Patients: `Age` of the patient, by range of 20

A visualization of how the patients' table was partitioned is shown in Figure 3.

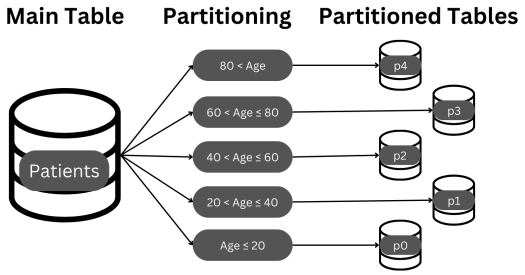


Figure 3: Patient Table Partitioning

After these partitions were identified and created for the other tables, the data had to be re-imported to new tables for temporary storage. These temporary tables were then used to import the data, divided based on the criteria mentioned earlier, into the newly partitioned tables. Python scripts were used to create the SQL syntax for the MainSpecialty and Province attributes. Finally, the SQL queries did not require any changes, as MySQL automatically self-optimizes the most efficient execution of the query, but all of the said queries can be improved by the partitioning done.

5.2 Indexing

Indexing was the other optimization strategy that was used, but not concurrently with Partitioning. Indexing improves query performance by assigning rows with unique indices, which helps the DBMS locate specific rows faster. It is a special kind of lookup table/reference table, similar to how a phone book would store the names and phone numbers of people to help humans find a specific person's phone number quickly.

Indices were created based only on the requirements of the queries, specifically those related to the `WHERE` clauses. These are separated by the queries as follows:

- Q1: `Province` of Clinic
- Q2: `TimeQueued` of appointment, and `isHospital` of Clinic
- Q3: `MainSpecialty` of doctor
- Q4: `Age` of patient

This was done using the sample syntax in Listing 9, wherein the indices are created, the query is conducted, and then the indices are removed for comparison as a baseline:

Listing 9: Index Creation and Dropping

```
-- Index Creation
create index IX_time_queued on appointments (TimeQueued);
create index IX_IsHospital on clinics (IsHospital);


-- Index Dropping
alter table appointments drop index IX_time_queued;
alter table clinics drop index IX_IsHospital;
```

Similar to the partitioning process, the actual SQL statements were not changed, as MySQL automatically optimizes based on its calculations without needing explicit instruction to do so.

5.3 Other Techniques

Apart from partitioning and indexing, other techniques to improve query speeds include structuring the database schema properly, creating correct but also efficient queries, and reducing the amount of data through cleaning.

The initial design had the same attributes as the current one but sported no FK and PK constraints on the tables. This led to slower read speeds and potential data integrity issues. Thus the said keys were added, as a good model also implies improved performance on all SQL queries.

Other than changing the database design/tables themselves, the base SQL also saw some improvements. Specifically, query number two, which retrieves appointments in the morning based on whether it was conducted in a hospital or not, was also restructured. For instance, an unnecessary `WHERE` argument that involved a `TIME()` conversion was removed after realizing that it was not contributing anything to the necessary filtering.

6 RESULTS AND ANALYSIS

Testing involved both functional and performance testing. Functional testing was primarily limited to the MySQL Workbench and whether it would run and query as intended. As such, working queries were considered functional. After performing OLAP operations, the number of rows also served as the basis for checking whether they worked as intended.

On the other hand, Performance testing was also conducted using MySQL's duration window. This involved testing each, namely the base queries with the optimizations under 'Other Techniques', then with partitioning, then finally with indexing. Note again that the partitioning test had no PKs and FKs as these were not available.

Each statement was first run to allow caching and was not recorded. Each was also tested one at a time and was conducted on a workstation machine. The specifications of the machine include an Intel i7-8700 @ 3.2Ghz and 32GB of RAM, the warehouse is stored on an SSD, and a constant load was kept throughout the test. Finally, the time recorded was then averaged to four decimal places and compared using percent difference. The results are shown in the following tables: Table 2 for base queries, Table 3 for partitioned, Table 4 for indexed, and finally summarized in Table 5.

Table 2: Speed of Base Queries in Seconds

Query	T1	T2	T3	T4	T5	AVE
1	1.188	1.266	1.203	1.235	1.187	1.216
2	0.704	0.703	0.719	0.703	0.688	0.703
3	1.125	1.172	1.187	1.172	1.094	1.150
4	0.703	0.735	0.687	0.625	0.609	0.672

As seen in Table 2, After the five conducted runs of the SQL command, Query number 1 had an average speed of 1.216, Query 2 at 0.703, Query 3 with 1.150, and finally Query 4 at 0.672. These are considerably fast, which is mostly attributed to not only the query structure but also the proper design of the database. In particular, the PKs allowing for some simple indexing help the queries to perform decently well at base. After partitioning and removing PK and FK due to constraints, some of the queries saw improvement. Specifically, Table 3 shows that Query 1's average speed dropped to 0.884; Query 2 sped up to 0.587, and Query 3 at 0.791. This is equivalent to an increase in query speed of 27.27%, 16.49%, and 31.25% respectively. On the other hand, Query 4 slowed to 3.028 seconds, which is roughly an increase of query time of 350.76%, which was much slower than the base query.

Table 3: Speed with Partitioning in Seconds

Query	T1	T2	T3	T4	T5	AVE	% Diff.
1	0.875	0.891	0.875	0.89	0.89	0.884	27.27%
2	0.562	0.594	0.594	0.609	0.578	0.587	16.49%
3	0.797	0.828	0.797	0.766	0.765	0.791	31.25%
4	3.0	3.125	3.11	2.912	2.994	3.028	-350.76%

Faster speeds are attributed to the Partitioning function working as intended and improving MySQL's workflow. This works by limiting the amount of data that needs to be scanned or processed for a given query. With this, MySQL can optimize itself to execute the queries better and quicker.

However, the much slower average speed for Query 4 was alarming. Upon closer inspection, it was found that full table scans were being done. As shown in Figure 4, the three tables in question are fully scanned and joined together, resulting in much overhead. On the other hand, Figure 5 shows that the PKs and FKs and their built-in indexing helped to improve the performance of the query. This simply means that the Partitioning was not used optimally, which led to poorer performance in comparison with the original base query.

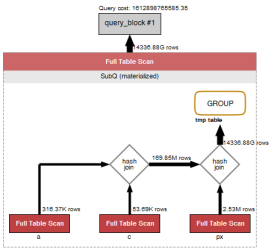


Figure 4: Patient Table Partitioning

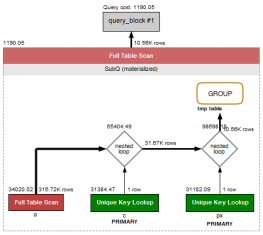


Figure 5: Patient Table Partitioning

The data warehouse was restored to its previous configuration, including primary keys (PKs) and foreign keys (FKs). After applying indexing to certain attributes. The speeds of all the inquiries were enhanced. In Table 4, Query 1 now has an average execution time of 1.094 seconds, which is a 10.03% increase in speed. Query 2 was at 0.572, representing an 18.68% rise. Query 3 took 0.472 seconds, showing a 58.97% increase. Query 4's response time decreased to 0.160 seconds, making it 76.24% faster.

Table 4: Speed of Base Queries in Seconds

Query	T1	T2	T3	T4	T5	AVE	% Diff.
1	1.078	1.078	1.063	1.125	1.125	1.094	10.03%
2	0.578	0.563	0.594	0.563	0.562	0.572	18.68%
3	0.453	0.5	0.469	0.469	0.468	0.472	58.97%
4	0.156	0.157	0.141	0.172	0.172	0.160	76.24%

In addition to the primary keys and foreign keys fulfilling their intended functions, the additional indexing also contributed significantly to the workload. Indexing can decrease the data scanned and rows combined, hence reducing the data filtered. MySQL's built-in optimizer determines and utilizes the most efficient route to execute a query.

The data is summarized once more in Table 5, showing that Partitioning marginally enhances query time but also has negative effects, while Indexing typically improves all queries, with some showing more significant improvements than others.

Table 5: Summary

Query N.	Base	Partition	% Diff	Index	% Diff
1	1.008	0.884	27.27%	1.094	10.03%
2	0.525	0.587	16.49%	0.572	18.68%
3	0.897	0.791	31.25%	0.472	58.97%
4	0.160	3.028	-350.76%	0.160	76.24%

7 CONCLUSION

This research involved a thorough investigation of data analysis on appointment data obtained from SeriousMD. The document outlines processes for analysis, which involve creating a data warehouse dimensional model. Subsequently, ETL procedures were carried out for data extraction, cleansing, and loading utilizing Python. Afterward, queries were generated on the data warehouse to examine patterns in appointment location, timing, and other factors. OLAP operations were conducted to analyze data based on broader or more particular features. The data was visualized using Tableau and improved through partitioning and indexing. Partitioning can enhance query performance based on the type of data being processed, as indicated by the results. Indexing led to quicker query speeds in all tests.

These methods revealed that establishing and upkeeping a data warehouse is an effective method to centralize and ensure the relational accuracy of data. The design of the dimensional model is influenced by the sort of analysis that will be conducted on the data. The ETL procedure facilitated the extraction of data from different sources including databases and standalone files, cleaning and formatting them to a specific structure, and finally importing the data into the destination data warehouse. Occasionally, the ETL scripts or pipelines can be executed to insert data into and refresh the warehouse as required.

Utilizing OLAP procedures in queries can yield a wider range of comprehensive results. OLAP can offer fresh insights into requested data that would have been obscured by the database tables and schema designs, or even by the basic Online Transaction Processing. while discussing queries, it is crucial to optimize them while working with databases. Enhancing query speed is essential in application development to prevent user dissatisfaction caused by longer loading times. Therefore, there are several methods to enhance inquiries and boost performance. These tasks involve ensuring proper database design, implementing necessary restrictions, denormalizing data and pre-calculating specific attributes, partitioning data and adding indexes based on queries, optimizing query construction, and enhancing physical hardware components like processors, storage devices, and memory.

Nevertheless, it is crucial to consider certain limitations when trying to enhance query performance. Storage may be compromised when designing denormalized tables that contain pre-calculated data. Similarly, indices should only be generated when essential, particularly for characteristics frequently utilized as restrictions in `WHERE` clauses.

Finding the equilibrium between performance and other variables such as storage is crucial in creating and managing databases. This equilibrium is expected to vary between different devices and databases. With the exponential growth of data volume on a daily basis, acquiring expertise in Big Data and understanding how to utilize it is a crucial ability to develop. This research demonstrates that information alone is insufficient as distinct and diverse procedures are required to generate meaningful analyses.

REFERENCES

[1] Asian Hospital and Medical Center. [n. d.]. *Departments.* Retrieved February 18, 2024 from https://www.asianhospital.com/departments/

[2] Lianne Kolirin and Arnaud Siad. 2023. US-born Spanish woman is now the world's oldest person, at age 115. *CNN* (2023). https://edition.cnn.com/2023/01/26/europe/maria-branyas-morera-oldest-person-scli-intl/index.html

[3] Milica Medic. 2014. *Database table partitioning in SQL Server.* Retrieved February 18, 2024 from https://www.sqlshack.com/database-table-partitioning-sql-server/

[4] Microsoft. 2023. *Partitioned tables and indexes.* Retrieved February 18, 2024 from https://www.ibm.com/docs/en/db2/11.1?topic=schemes-partitioned-tables

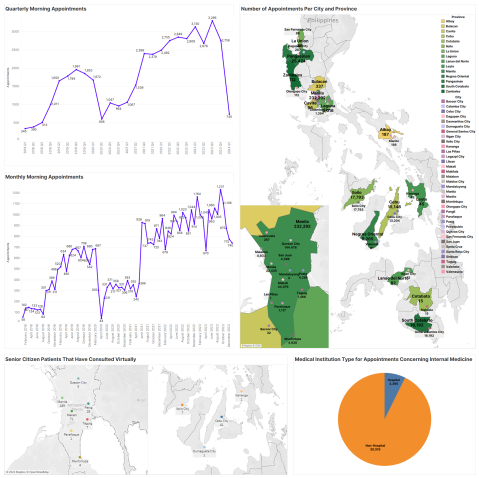
[5] David Pfaehler, H. Woo, and Saim Kashmiri. 2020. Electronic Health Records and Health Information Exchange and Their Impact on International Healthcare System Efficiency.

[6] SeriousMD. 2024. *Introduction to SeriousMD.* Retrieved February 18, 2024 from https://help.seriousmd.com/en/articles/1080179-introduction-to-seriousmd

[7] Kathryn Whitbourne. 2022. What Are the Different Types of Doctors? *WebMD* (2022). Retrieved February 18, 2024 from https://www.webmd.com/health-insurance/insurance-doctor-types

8 APPENDIX

8.1 Appendix A: Tableau Dashboard



8.2 Appendix B: Rules for Doctor Specialties

Rule for	Rule
Internal Medicine	- strings that contain "internal med" in the first 12 characters - strings equal to "internal" - strings that contain {"im ", "im-", "im –"} in the first 4 characters but not including the strings {"im not", "im trained"}
General Medicine	- strings that contain "general medicine" in the first 16 characters - strings equal to "general" - strings that contain "general phy" but not including the strings that contain {"psy", "rad"}
Pediatrician	- strings that contain "general pedia" in the first 13 characters - strings equal to "pedia"
Family Medicine	- strings that contain "family med" in the first 10 characters
Obstetrics and Gynecology	- strings that contain "ob" in the first 2 characters but not including the string "obesity"
Neurology	- strings that contain "neuro" in the first 5 characters or "adult neuro" in the first 11 characters - strings equal to "adult and pediatric neurology"
Cardiology	- strings that contain "cardio" but not including the string "pulmonologist, cardiologist, and internal medicine"
Orthopedics	- strings that contain "orthop"
Immunology	- strings that contain "aller" but not including the string "infectious disease, allergology"
Dermatology	- strings that contain "derma" but not including the strings that contain "dental"
Occupational Medicine	- strings that contain "occupational" in the first 12 characters
General Surgery	- strings that contain "general" in the first 7 characters and also contain "surgery"
Psychology	- strings that contain "psychol" in the first 7 characters
Otorhinolaryngology	- strings that contain {"otol","otor"} in the first 4 characters
string length is 2	- {"im","in"} → Internal Medicine - {"gp","md"} → General Medicine - {"rn"} → Nursing - {"fm"} → Family Medicine - {"pt"} → Physical Medicine and Rehabilitation - {"on"} → Medical Oncology - {"ob","gy"} → Obstetrics and Gynecology - {"gs"} → General Surgery - {"er"} → Emergency Medicine - {"id"} → Infectious Disease

9 DECLARATIONS

9.1 Declaration of Generative AI in Scientific Writing

During the preparation of this work the author(s) used ChatGPT and Quillbot in order to write drafts of non-documentation sections of the paper. 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

All authors contributed to the creation of the technical report. Contributions to the different processes involved in this study are as follows:

- Sealtiel B. Dy - Query Optimization, OLAP Application
- Kyle Carlo C. Lasala - Data Pre-processing, Data Warehouse Design, ETL Process and Pipeline
- Maria Monica Manlises - Data Pre-processing, OLAP Application
- Camron Evan C. Ong - Query Processing and Optimization