MCO1 – Query Processing Technical Report

Rey Vincent Inocencio[1], Aaron Randall Jardenil[2], Enrique Rafael Lejano[3]
De La Salle University-Manila
[1]rey_vincent_inocencio@dlsu.edu.ph, [2]aaron_randall_jardenil@dlsu.edu.ph, [3]enrique_rafael_lejano@dlsu.edu.ph

ABSTRACT

This project aims to explore applications of Online Analytical Processing (OLAP) and data warehouse concepts through a clinic appointment dataset from SeriousMD. An Extract, Transform, Load (ETL) process was used to extract data from a data source, process it, and load it to a data warehouse which was used in reports generation that employed OLAP operations. From here, visualizations were made through a data visualization platform "Tableau. Through this study, the relevance of query structuring and optimization as well as OLAP operations was discovered to be more evident when dealing with a massive dataset.

Keywords

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

1. Introduction

This project involves creating a Query Processing Technical Report from a Data Warehouse. For this, the dataset used is the SeriousMD Appointment Anonymized Dataset. This dataset contains the .csv files "appointments", "clinics", "doctors", and "px" (patients). The columns/attributes of each file are as follows:

- Appointments: pxid, clinicid, doctored, apptid, status, TimeQueued, QueueDate, StartTime, EndTime, type, Virtual
- Clinics: clinicid, hospitalname, IsHospital, City, Province, RegionName
- Doctors: doctored, mainspecialty, Age
- Px (Patients): pxid, age, gender

The group used Python scripts to perform ETL processes and load the data into a Data Warehouse, following a Star Schema structure. This data warehouse was hosted on a MySQL Server, which was used as the main connection for the OLAP Application.

For this project, Tableau was the group's chosen OLAP application. It is a industry-standard visual analytics tool that specializes in querying relational databases, OLAP cubes, cloud databases, and spreadsheets to produce data visualizations and analyses (Tableau, 2024).

The group used Tableau to generate reports of the data stored inside the Data Warehouse. Through a combination of custom MySQL queries and chart manipulation through Tableau's built-in tools, the group successfully generated intuitive visualizations for better interpretation of the group's reports.

2. Data Warehouse Design

In the process of creating the Data Warehouse for the project, the group chose to design a data warehouse adhering to the design of a star schema and hosting it in a local MySQL database server. For this, the appointments table was treated as the Fact Table whereas the patients, doctors, and clinics tables were used as the denormalized dimension tables.

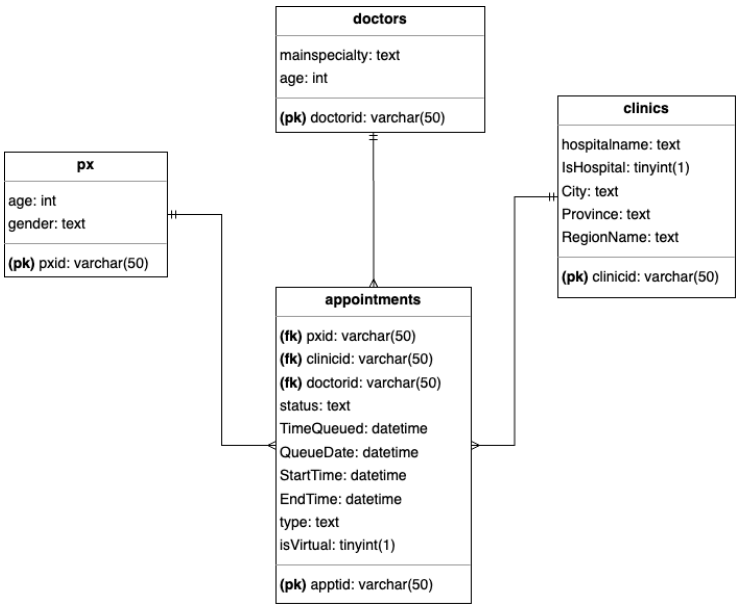


Figure 1. Dimension Table for Data Warehouse

Columns in each CSV file were assigned their respective data types, and foreign keys were put in place to declare the relationships between each table.

As appointments are the primary business focus of the queries, the group decided that appointments were the most suitable table to be used as the fact table. The CSV files of the other tables also contained fields that were present in the appointments table. As a result, the schema can be designed for the relationships of the appointments table to the other tables.

Appointments shall be used as the focus for the queries, and secondary information can be derived from the other tables. For example, for reports surrounding appointments and the patients of these appointments, the patients (px) table can be used to derive demographic information for the query such as their age and gender.

3. ETL Process

The SeriousMD dataset contains four CSV files, two of which contain over 5 million rows and 9 million rows, respectively. With this in mind, loading the data manually to a MySQL database would be inefficient. Thus, the group employed Python scripts to Extract, Process, and Load the data into the database. The CSV files were loaded and cleaned via a Jupyter Notebook before finally inserting them into the database.

Clinics Table

Starting with the *clinics* table, a data frame was initialized with the duplicate rows dropped. The group did not drop rows containing null values, as this table is one of the dimension tables, and removing any rows may cause errors pertaining to foreign key constraints. The group also checked for duplicates in the "*clinicID*" field to prevent referential errors between the Clinic table and the Appointments Table. Upon checking, there were no duplicate keys. The *clinics* table was also checked for null values. To prevent errors when loading the data to the data warehouse, the group set the null values under the "hospitalName" column to the string "noName."

Doctors Table

For the Doctors table, a similar procedure was followed. Here, no duplicates were found. Upon checking for null values, however, there were multiple rows with missing fields for "mainSpecialty" and "age". The null values for the *mainSpecialty column* summed up to 32,849 whereas the most frequently used term for general medicine only had 2,317. As such, the group decided to compromise by giving a default value "none" for specialty in order to prevent issues with importing the data to the warehouse. On the other hand, null values for *age* were set to a sentinel value of -1 to prevent any errors when loading the data into the data warehouse.

Patient Table

Lastly, for the Patients table, the dataframe was once again initialized with duplicates dropped, before being checked for duplicate values for the "pxid" column. Here, multiple instances of the same "pxid" value existed. These were removed accordingly in order to comply with Foreign Key Constraints. Another discrepancy found within the Patients table was the non-standard format for the "age" column. There were mixed data types in the table containing float, string, and integer types. To resolve this, rows were removed where the data type of the column was "string". Then the column was casted to the integer data type.

Appointments Table

Lastly, for the Appointments table, the data frame was initialized with the function .dropna() to remove rows with missing values. The dataframe was also initialized with the .drop_duplicates() function to avoid any duplicate rows. The group then checked for the unique values across the categorical columns "*status*", "*unique*", and "*virtual*".

The group found no instances of multiple representations of these columns. The last step in the transformation process was removing rows in the Fact Table "Appointments" where the fields "pxid", "doctorid", and "clinicid" did not match the dimension tables they were derived from (i.e., *Appointments.pxid* values having no matching value in *px.pxid*). This is to avoid discrepancies with the data and ensure that each foreign key in the Fact Table can be mapped to an entry in the dimension tables, allowing the group to derive meaningful information from each appointment.

Finally, with the use of the MySQL module in Python, each table was loaded into the database, and from there, the relationships between tables were set.

4. OLAP Application

The main purpose of the developed application is to provide relevant analysis of trends related to appointments along with their locations and times. These include the number of appointments across different locations and periods, which could provide insights into the performance of different clinics.

The application can help clinics improve operations and accommodate patients better by revealing insights on appointment timestamps and frequencies. The application can also help local government units and the national government by providing reports on medical facility and appointment location counts; hopefully prompting them to construct medical facilities in less developed areas.

Below are the four analytical reports generated for using the OLAP application.

4.1 Clinic and Hospital Location Statistics

Though appointments are the focus of this application, the clinics and hospitals where these appointments occur must be analyzed first. Initially, the group queried the database only for the number of clinics per region, as seen in Figure 2. However, this query lacked granularity—given that and could use additional information for better analysis.

```
SELECT      RegionName,
            COUNT(*) as ClinicCount
FROM        clinics
GROUP BY    RegionName;
```

Figure 2. Initial SQL Query for Clinic Location Count.

Thus, the group utilized the Drill-down OLAP Operation to obtain more useful information as seen in the queries in Figure 3. The query grouped the number of clinics according to each granularity available, from the least granular (Region) to the most granular (City). This was accomplished through the SQL GROUP BY statement. The number of recorded clinics that are also hospitals was included in the query as well by checking the IsHospital field per clinic via a combination of the aggregate COUNT() function and the conditional IF() function.

```
SELECT      City,
            Province,
            RegionName,
            COUNT(*) as ClinicCount,
            COUNT(IF(IsHospital = 1, 1,
            NULL)) as HospitalCount
FROM        clinics
GROUP BY    City, Province, RegionName
ORDER BY    RegionName, Province, City
```

Figure 3. SQL Query for Clinic and Hospital Location Count.

Thus, the group utilized the Drill-down OLAP Operation to obtain more useful information as seen in the query in **Figure 3**. The query grouped the number of clinics according to each granularity available, from the least granular (Region) to the most granular (City). This was accomplished through the SQL GROUP BY statement. The number of recorded clinics that are also hospitals was included in the query as well by checking the IsHospital field per clinic via a combination of the aggregate COUNT() function and the conditional IF() function.

The three following figures showcase snippets of data visualizations for each granularity. It is evident through these visualizations that there is a disproportionately high number of clinics and hospitals in the National Capital Region (NCR).

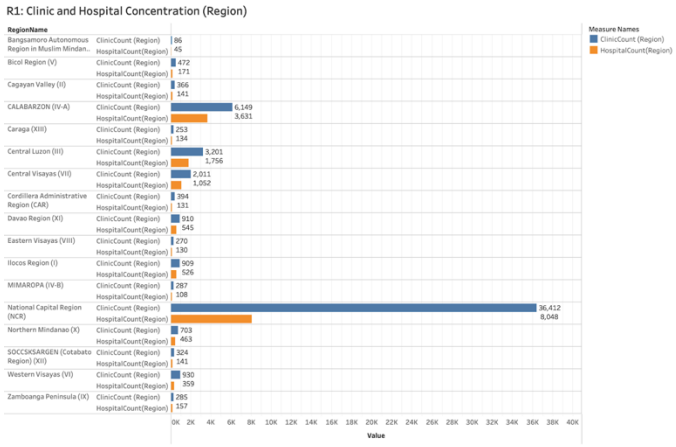


Figure 4. Clinic and Hospital Concentration by Region

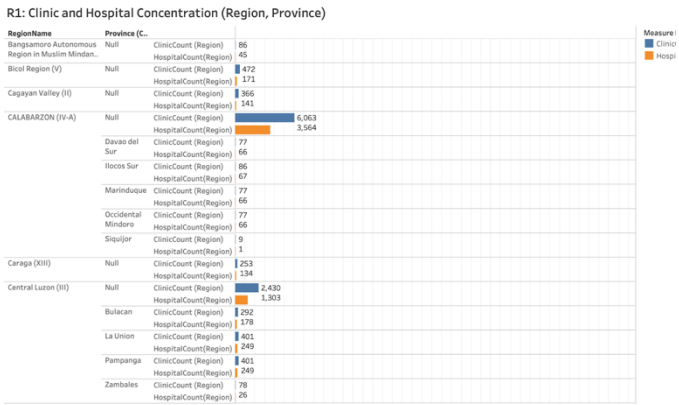


Figure 5. Clinic and Hospital Concentration by Region and Province

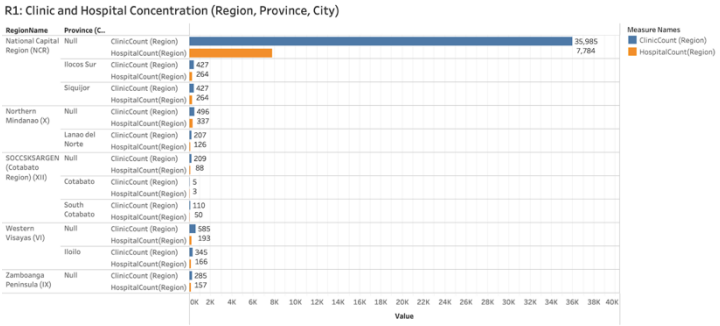


Figure 6. Clinic and Hospital Concentration by Region, Province, and City

4.2 Appointment Location Concentration

Following the report on clinic and hospital concentration throughout the country, the next step would be to analyze the number of appointments per location. To facilitate a more relevant analysis, the group also decided to obtain appointment quantities during the timeframe of the COVID-19 pandemic, which was declared a Public Health Emergency of International Concern (PHEIC) from January 30, 2020, to May 5, 2023 (World Health Organization, 2024).

The initial query for this report involves counting all the appointments per region and using a Slice OLAP operation by providing a new sub-cube using the aforementioned dates via the appointment StartTime field.

```
SELECT      c.RegionName, COUNT(*) as
            AppointmentCount
FROM        appointments as a
JOIN        clinics as c
ON          a.clinicid = c.clinicid
WHERE       a.StartTime BETWEEN '2020
            -01-30' AND '2023-05-05'
GROUP BY    c.RegionName
ORDER BY    AppointmentCount DESC;
```

Figure 7. SQL Query for Appointment Concentration

	RegionName	AppointmentCount
1	National Capital Region (NCR)	40250
2	Central Visayas (VII)	1360
3	CALABARZON (IV-A)	1345
4	Central Luzon (III)	291
5	SOCCSKSARGEN (Cotabato Region) (XII)	98
6	Eastern Visayas (VIII)	38
7	Western Visayas (VI)	14
8	Ilocos Region (I)	5
9	Northern Mindanao (X)	3

Figure 8. Output for Appointment Concentration Query

The query output shows that, once again, NCR has many more appointments than any other region. But in a real-world situation—especially as this query simulates the pandemic period—it is more common to analyze appointment counts per city. And given the low number of appointments in regions within Visayas and Mindanao, analyzing appointment counts specifically from Luzon is more beneficial in this report.

Thus, Figure 9 shows the group employing two OLAP operations to achieve this. First, the Drill-down operation narrowed down the

appointment location from regions to cities using the GROUP BY operator. Next, the Dice operation was employed along two dimensions: appointment date and location: exclusively selecting appointments with dates from January 30, 2020, to May 5, 2023, via the WHERE and BETWEEN, and cities only within NCR via the NOT and IN operators that check against a list of values. Figure 10 also showcases a map-based visualization of this query, where Quezon City tops the number of appointment counts during this timeframe.

```
SELECT      c.City, COUNT(*) as AppointmentCount
FROM        appointments as a
JOIN        clinics as c
ON          a.clinicid = c.clinicid
WHERE       a.StartTime BETWEEN '2020-01-30' AND '2023-
05-05'
AND         c.RegionName IN (
                            'National Capital Region
                            (NCR)',
                            'Cordillera
                           Administrative Region
                            (CAR)',
                            'Ilocos Region (I)',
                            'Cagayan Valley (II)',
                            'Central Luzon (III)',
                            'CALABARZON (IV-A)',
                            'MIMAROPA (IV-B)',
                            'Bicol Region (V)'
                        )
GROUP BY    c.City
ORDER BY    AppointmentCount DESC;
```

Figure 9. SQL Query for Appointment Concentration in Luzon

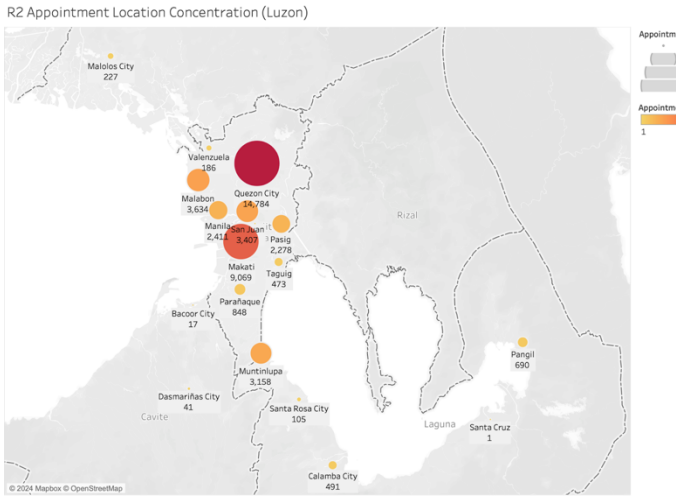


Figure 10. Data Visualization for Luzon Appointment Concentration

4.3 Appointment Quantity Trend Over Time

Continuing with the trend of using the pandemic timeframe, the third report aims to analyze the trend in appointment volume across the same period. This information will prove useful when conducting medical case studies in the years ahead, especially for studies concerning the COVID-19 pandemic. Spikes in appointment quantity could be correlated to COVID-19 strain outbreaks such as the Delta and Omicron variants (Katella, 2023). Meanwhile, drastic decreases could be correlated to the release of COVID-19 vaccines.

The initial query for this report uses the Slice OLAP operation to once again select the pandemic timeframe of January 2020 to May 2023. Appointments were grouped according to the year and specific week within the year that they occurred followed by a line chart visualization shown in Figure 11 and Figure 12, respectively.

```
SELECT      YEAR(a.StartTime) as AppointmentYear,
            WEEK(a.StartTime) as AppointmentWeek,
            COUNT(*) as AppointmentCount
FROM        appointments as a
JOIN        clinics as c
ON          a.clinicid = c.clinicid
WHERE       a.StartTime BETWEEN '2020-01-30' AND
            '2023-05-05'
GROUP BY    AppointmentWeek, AppointmentYear
ORDER BY    AppointmentYear, AppointmentWeek;
```

Figure 11. SQL Query for Appointment Quantity Trend by Week

	AppointmentYear	AppointmentWeek	AppointmentCount
1	2020	36	192
2	2020	37	379
3	2020	38	402
4	2020	39	353
5	2020	40	345
6	2020	41	333
7	2020	42	316
8	2020	43	304
9	2020	44	298
10	2020	45	344
11	2020	46	326

Figure 12. Output for Appointment Quantity Trend Query

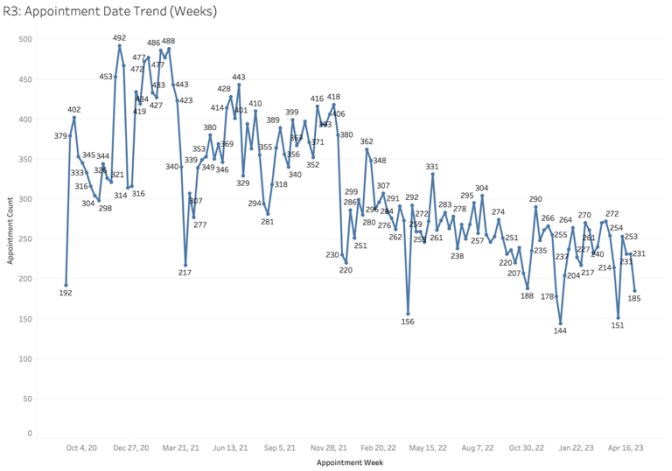


Figure 13. Data Visualization for Appointment Date Trend (Weeks)

However, the group quickly noticed that the high granularity of the initial query made the information cumbersome to interpret. When the query is executed on a SQL workbench, the output is easy to understand. But when the output is visualized, the trend is difficult to identify—not to mention how unintuitive it is to count the weeks in a year.

Thus, the group employed a Roll-up OLAP operation to decrease the output's granularity for easier interpretation. The query below

showcases a climb in the concept hierarchy of the appointment's StartTime, going from grouping the appointments by months instead of weeks. Additionally, the WITH ROLLUP modifier for the GROUP BY statement was used for easier interpretation of the SQL Workbench's output as well.

```
SELECT      YEAR(a.StartTime) as AppointmentYear,
            MONTH(a.StartTime) as
            AppointmentMonth,
            COUNT(*) as AppointmentCount
FROM        appointments as a
JOIN        clinics as c
ON          a.clinicid = c.clinicid
WHERE       a.StartTime BETWEEN '2020-01-30' AND
            '2023-05-05'
GROUP BY    AppointmentYear, AppointmentMonth WITH
            ROLLUP
ORDER BY    AppointmentYear, AppointmentMonth;
```

Figure 14. SQL Query for Appointment Date Trends by Month

	AppointmentYear	AppointmentMonth	AppointmentCount
1	<null>	<null>	43404
2	2020	<null>	5890
3	2020	9	1179
4	2020	10	1445
5	2020	11	1374
6	2020	12	1892
7	2021	<null>	19666
8	2021	1	1906
9	2021	2	1850
10	2021	3	1792
11	2021	4	1270
12	2021	5	1564

Figure 15. Output for Appointment Quantity Trend Query

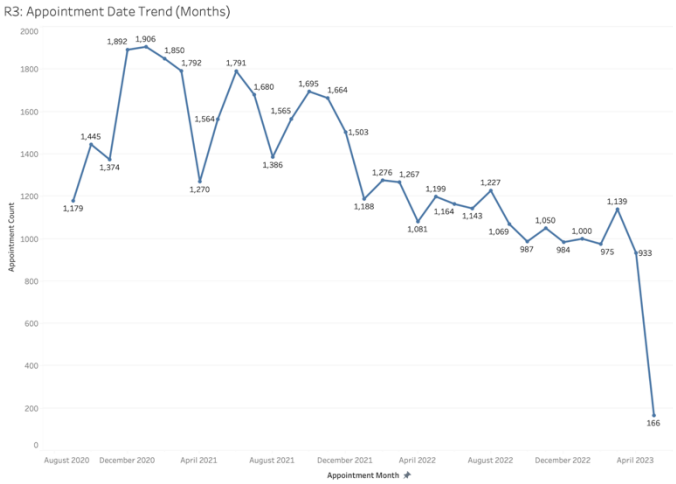


Figure 16. Data Visualization for Appointment Date Trend (Months)

With just a glance at Figure 16, it is evident that the output has become much easier to analyze because each point in the chart represents a single month.

But the group decided to also climb one more concept in the DateTime hierarchy by grouping the appointments in Quarters of the year. Out of the three visualizations presented in this report, this would be an optimal granularity for a high-level overview of the appointment volume trends across all the clinics and hospitals throughout the pandemic period.

```
SELECT      YEAR(a.StartTime) as AppointmentYear,
            QUARTER(a.StartTime) as
            AppointmentQuarter,
            COUNT(*) as AppointmentCount
FROM        appointments as a
JOIN        clinics as c
ON          a.clinicid = c.clinicid
WHERE       a.StartTime BETWEEN '2020-01-30' AND
            '2023-05-05'
GROUP BY    AppointmentYear, AppointmentQuarter
WITH ROLLUP
ORDER BY    AppointmentQuarter, AppointmentYear;
```

Figure 17. SQL Query for Appointment Date Trends by Quarter

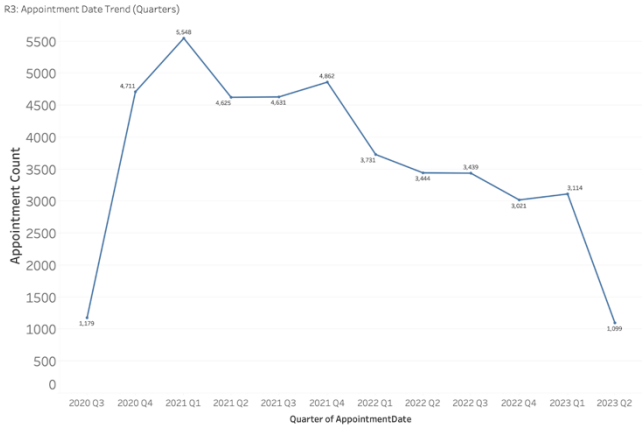


Figure 18. Data Visualization for Appointment Date Trend (Quarter)

4.4 Average Virtual & In-Person Appointments per Hour

Generating a report on the average number of appointments per hour could give some insight in which hours of the day can be considered peak hours for appointments. This would be useful in analyzing patient scheduling patterns and identifying time frames during the day when the demand for medical services is the highest. By pinpointing these hours, resources and working shifts for doctors, nurses, and other facilitators could be adjusted accordingly to more efficiently handle patient volumes.

```
SELECT
    Hour,
    AVG(`Virtual`) AS Avg_Virtual,
    AVG(InPerson) AS Avg_In_Person
FROM
    (SELECT
        c.City AS City,
        HOUR(a.StartTime) AS Hour,
        SUM(CASE WHEN a.isVirtual = 1 THEN 1 ELSE
        0 END) AS `Virtual`,
        SUM(CASE WHEN a.isVirtual = 0 THEN 1 ELSE
        0 END) AS InPerson
    FROM
        appointments a
    JOIN
        clinics c ON a.clinicid = c.clinicid
    WHERE
        c.RegionName = 'National Capital Region
        (NCR)'
        AND YEAR(a.StartTime) = 2023
    GROUP BY City, Hour) AS subquery
GROUP BY Hour
ORDER BY Hour;
```

Figure 19. SQL Query for Average Virtual & In-Person Appointments per Hour

The advanced SQL constructs used in this report were aggregate functions such as SUM, AVG, GROUP BY, subqueries, and pivoting. The query uses subquerying, the SUM function, and the CASE clause to count the total number of in-person and virtual appointments in cities across Metro Manila for the year 2023. Afterward, the sums were grouped by the hours of the day and the location in which the appointments took place. The subquery was then pivoted to get the average total of appointments across all the cities for each hour of the day.

Mapping the cities and hours of the day as the y-axis, the total number of appointments as the x-axis, and the time as the z-axis in an OLAP cube, drill-down, slice, and dice OLAP operations were utilized in this query. Initially, the data was rolled up to count and calculate the average total appointments across different regions, but since the volume of the data is very large and granular, it was drilled down to cities in order to generate more specific insights. The appointment times were also drilled down from dates to hours of the day. It was then diced to include only the cities within Metro Manila. Lastly, it was sliced to only include appointments from the year 2023.

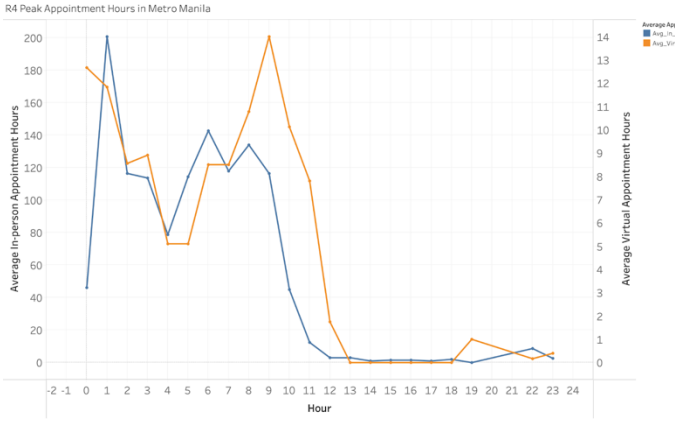


Figure 20. Data Visualization for Average Virtual & In-Person Appointments per Hour

After visualizing the query through a line graph in the OLAP application, with the x-axis being the hours of the day, and the y-axis being the number of appointments, the trend seems to show that both virtual and in-person appointments are generally at their peak at 1:00 - 2:00 AM and 10:00 AM in the morning. They then start to decline from 10:00 AM onwards, although it is worth noting that virtual appointments decline slower. Thus, a possible improvement would be to adjust working shifts to have more personnel available in the early morning rather than in the afternoon.

5. Query Processing and Optimization

5.1 Database Reprocessing and Further Data Cleanup

During the process of running tests on queries for Reports 1-3, discoveries were made regarding possible problems with the state of the database and data warehouse. Notably, there were a lot of appointments with patient ID's absent from the patient table. Initially, our data warehouse still included these rows. However, a reprocessing was performed to only include rows where candidate keys have primary keys (all rows are unique) in the dimension tables.

Table 1. Execution Times of Reports 1-3 Before Optimization

Test Case				
1	**2**	**3**	**4**	**5**
Report 1 Query				
20 s	20 s	21 s	20 s	20 s
Report 2 Query				
11 s	13 s	14 s	14 s	12 s
Report 3 Query				
11 s	11 s	12 s	12 s	12 s

Table 2. Execution Times of Reports 1-3 After Optimization

Test Case				
1	**2**	**3**	**4**	**5**
Report 1 Query				
419 ms	308 ms	276 ms	245 ms	260 ms
Report 2 Query				
118 ms	219 ms	175 ms	271 ms	180 ms
Report 3 Query				
195 ms	219 ms	186 ms	192 ms	203 ms

This was a necessary reprocessing because including appointments without registered patients could introduce gaps within the data and the queries in the long run given that information would be incomplete. This optimization is logical since a considerable number of unnecessary rows were truncated from the fact table, thereby reducing the workload during query processes. After applying the redesign and data cleaning, a substantial reduction in the total number of rows for the fact table was observed, dropping by over a million. This reprocessing proved to be beneficial, resulting in faster execution times. For instance, the queries saw a significant improvement from more than 10 seconds down to milliseconds.

5.2 Use of Primary and Foreign Keys and Indexing

Table 3. Execution Times of Report 4 Query

Test Case				
1	2	3	4	5
With Primary and Foreign Keys and Indexing				
54 s	44 s	44 s	41 s	41 s
Without Primary and Foreign Keys and Indexing				
24 s	24 s	24 s	24 s	24 s

Interestingly enough, while ensuring the use of indexes and primary keys is a common optimization strategy to improve query performance, after testing, it seems that omission from the use of these actually improved the query performance in terms of execution time for the query in Report 4. It is important to note, however, that these were the execution times before the first optimization was applied. Upon further review and investigation, it was concluded that it may have to do with the use of subqueries in the query.

```
SELECT
    Hour,
    AVG(`Virtual`) AS Avg_Virtual,
    AVG(InPerson) AS Avg_In_Person
FROM
    (SELECT
        c.City AS City,
        HOUR(a.StartTime) AS Hour,
        SUM(CASE WHEN a.isVirtual = 1 THEN 1 ELSE
        0 END) AS `Virtual`,
        SUM(CASE WHEN a.isVirtual = 0 THEN 1 ELSE
        0 END) AS InPerson
    FROM
        appointments a
    JOIN
        clinics c ON a.clinicid = c.clinicid
    WHERE
        c.RegionName = 'National Capital Region
(NCR)'
        AND YEAR(a.StartTime) = 2023
    GROUP BY
        City, Hour) AS subquery
GROUP BY Hour
ORDER BY Hour;
```

Figure 21. Subquery from Report 4 (Bold Portion)

According to the MYSQL 8.0 Reference Manual, indexes are used to find rows with specific column values quickly without iterating through every row in the entire table. While that makes READ operations significantly more efficient, it is not necessarily the same for WRITE operations such as INSERT and UPDATE. Indexing could cause performance loss for write operations, especially when dealing with massive datasets. More indexes must be updated; thus, increasing query processes and execution time. This has been experienced by multiple users on SQL forum sites, as seen in a discussion thread on Medium.com.

This property of MySQL applies particularly for Report 4 as it uses a subquery to count total appointments across different cities for a region which is then pivoted to get the average appointments for that region. The subquery effectively creates a temporary table with the total count of appointments per city, which could also mean that it updates the indexes for every row that is inserted in that temporary table, possibly being the cause of loss in performance when executing the query on the tables with the primary and foreign keys being properly set up versus without.

Table 4. Execution Times of Reports 1-3 Without Keys and Indexing

Test Case				
1	2	3	4	5
Report 1 Query				
447 ms	437 ms	426 ms	426 ms	452 ms
Report 2 Query				
414 ms	416 ms	414 ms	436 ms	397 ms
Report 3 Query				
493 ms	412 ms	395 ms	428 ms	494 ms

Table 5. Execution Times of Reports 1-3 With Keys and Indexing

Test Case				
1	2	3	4	5
Report 1 Query				
285 ms	296 ms	287 ms	286 ms	290 ms
Report 2 Query				
257 ms	234 ms	228 ms	240 ms	225 ms
Report 3 Query				
242 ms	256 ms	241 ms	235 ms	230 ms

On the other hand, using the keys greatly improved the performance for Reports 1-3. It reduced the execution time of the queries from insert data here. This could be because unlike Report 4, the queries for Reports 1-3 do not use any subqueries, making the indexing more effective, since they are mostly retrieving rows and only make use of READ operations.

Table 6. Execution Times of Report 4 With Both Optimizations Applied

Test Case				
1	2	3	4	5
With Primary and Foreign Keys and Indexing				
128 ms	158 ms	127 ms	131 ms	132 ms
Without Primary and Foreign Keys and Indexing				
363 ms	348 ms	347 ms	338 ms	366 ms

However, after the first optimization was introduced, this eliminated the increased execution times for Report 4 when using properly set up primary and foreign keys, thus, like the other reports, ultimately led to improved performance. This could be because the volume of the appointments table became significantly smaller after the first optimization, which could have negated the problem described in the previous paragraph.

6. Results and Analysis

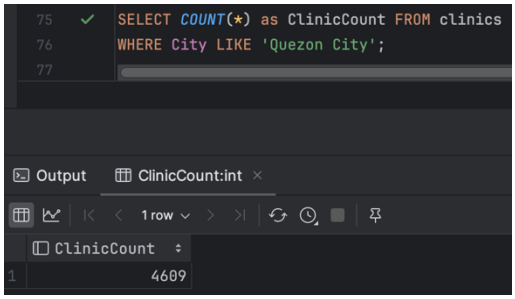
I. ETL Process Functional Testing

Function testing for the ETL process includes validating the consistency of data between the fact table and dimension tables. The validations were performed by constructing basic queries to examine sample rows. Various built-in tools from the Pandas Python library were also used to check for duplicates missing data points, and foreign key constraints.

II. OLAP Operations Functional Testing

Creating additional SQL queries with the intent of cross-checking outputs was the first method used to validate the group's OLAP operations. For example, in Report 1, separate queries were performed to check the number of clinics and hospitals for several cities. If the output of these queries matches the original OLAP query, then the process is correct. An example is shown in Figure 22 and Figure 23.

Figure 22. Output for OLAP Operation Verification



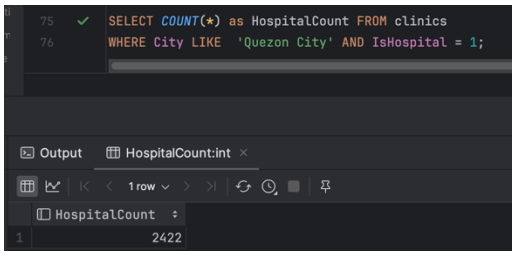


Figure 23. Hospital and Clinic Count Verification

A similar process was also used in Report 4, where simplified queries were derived from deconstructing the original queries. The subquery was isolated from the OLAP query and the resulting table was crosschecked against the final query.

Another strategy to validate the group's OLAP operations can be found in Report 3 with the use of the WITH ROLLUP SQL clause. The group ran queries that extracted the same information, just with varying granularity; thus, the total output row count and grouping counts should not change.

An example of this can be seen in Figure 25, where the number of appointments, both in total and per year, did not change despite the climb in the hierarchy from months to quarters.

	AppointmentYear	AppointmentMonth	AppointmentCount
1	<null>	<null>	43404
2	2020	<null>	5890
3	2020	9	1179
4	2020	10	1445
5	2020	11	1374
6	2020	12	1892
7	2021	<null>	19666
8	2021	1	1906
9	2021	2	1850
10	2021	3	1792
11	2021	4	1270
12	2021	5	1564
13	2021	6	1791
14	2021	7	1680
15	2021	8	1386
16	2021	9	1565
17	2021	10	1695
18	2021	11	1664
19	2021	12	1503

Figure 24. Report 3 Query (Months) using WITH ROLLUP clause

	AppointmentYear	AppointmentQuarter	AppointmentCount
1	<null>	<null>	43404
2	2020	<null>	5890
3	2021	<null>	19666
4	2022	<null>	13635
5	2023	<null>	4213
6	2021	1	5548
7	2022	1	3731
8	2023	1	3114
9	2021	2	4625
10	2022	2	3444
11	2023	2	1099
12	2020	3	1179
13	2021	3	4631
14	2022	3	3439
15	2020	4	4711
16	2021	4	4862
17	2022	4	3021

Figure 25. Report 3 Query (Quarters) using WITH ROLLUP clause

III. Performance Testing

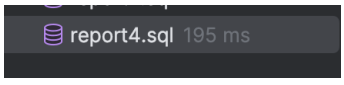


Figure 26. Query Execution Time in DataGrip

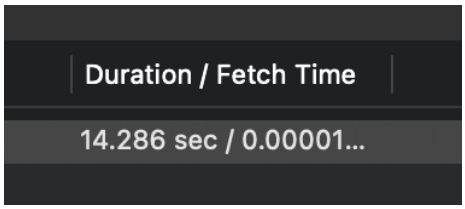


Figure 27. Query Execution Time in MySQL Workbench

In order to test the performance of the queries, each was run through 5 test cases under different conditions, mainly before and after each optimization process was applied. Afterward, the execution times, which were readily available through MySQL, were noted down, as presented in the execution time tables presented in the previous section.

Table 4. Execution Times of Reports 1-3 Without Keys and Indexing

Test Case	Intel Core i5 + 8GB RAM	Apple M1 Pro + 16GB RAM
1	184 ms	97 ms
2	187 ms	81 ms
3	189 ms	95 ms
4	177 ms	89 ms
5	195 ms	90 ms

Varying hardware was also used to test the performance across two different machines. The first machine had an Intel Core i5 Processor with 8 Gigabytes of RAM. The other machine had an Apple Silicon M1 Pro Processor with 16 Gigabytes of RAM. The same differences were observed when applying the optimization processes, however, the Apple Silicon machine notably was relatively faster.

A big factor was the volume of the data in the data warehouse, which greatly affected query performance. While the strategy of reprocessing the data warehouse to drop data that is incomplete was able to improve our queries, it is important to note that it would not apply to every situation, since there could be a situation where data is already thoroughly processed but is still very large.

7. Conclusion

The group gained several learnings through this project.

ETL enables not only physical loading and storing of data in the warehouse but also helps clean and pinpoint which data to load, thus maintaining the integrity and functionality of the data warehouse. In the real world, data is multi-dimensional, layered, and messy. But ETL allows everything to come together.

In OLTP, each transaction is often handled individually. In contrast, in OLAP, more advanced inquiries can be made from the available data, thereby producing deeper insights that could be used to make data-driven decisions. Put simply, OLTP allows for a microscopic view of data but OLAP allows for a macroscopic view.

This information will be useful for several stakeholders. SeriousMD themselves the data preprocessing and ETL section to improve their data collection process to ensure cleaner datasets in the future. LGUs can observe the appointment locations to propose locations for new public hospitals, especially in areas with fewer clinics. Researchers can use the appointment quantity trends during the COVID-19 timeframe to conduct case studies on the pandemic. Finally, the clinics and hospitals present in SeriousMD's database can access the appointment time trends to better allocate employee shifts to best accommodate their patients.

Overall, the opportunity to work with real-world data via the SeriousMD dataset has been a great learning experience for the group. Through understanding, cleaning, and querying information sourced from actual patients and clinics, the group garnered a larger appreciation for the importance of data analytics and visualization.

8. References

[1] Kathy Katella. 2023. Omicron, Delta, Alpha, and More: What To Know About the Coronavirus Variants. Yale Medicine. Retrieved from https://www.yalemedicine.org/news/covid-19-variants-of-concern-omicron

[2] Oracle Corporation. 2019. MySQL :: MySQL 8.0 Reference Manual. Mysql.com. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/

[3] Tableau. 2023. What is Tableau? Tableau. Retrieved from https://www.tableau.com/why-tableau/what-is-tableau

[4] World Health Organization. 2023. Coronavirus Disease (COVID-19) Pandemic. www.who.int. Retrieved from https://www.who.int/europe/emergencies/situations/covid-19

9. Declarations

During the preparation of this work the authors used ChatGPT by OpenAI in order to debug syntax errors encountered with the scripts used in the database. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.