Query Processing and Optimization in a Data Warehouse

Chua, Edric[1], Lim, Rydel[2], Ngo, Rendell[3], Santiago, Nikolai[4]

College of Computer Studies, De La Salle University

[1]edric_jarvis_chua@dlsu.edu.ph [2]rydel_ridley_lim@dlsu.edu.ph, [3]rendell_christian_ngo@dlsu.edu.ph, [4]nikolai_santiago@dlsu.edu.ph,

ABSTRACT

This paper examines the use of query optimization techniques (specifically indexing, query restructuring, and table denormalization) to lower the execution time of OLAP related queries. This process includes the use of multiple ETL pipelines to properly clean and utilize the large medical dataset provided. The data utilized concerns medical appointments between patients and doctors, as well as where these were located and when each individual appointment occurred. The analytical reports generated were intended for use by professionals in the medical industry, as well as those managing different medical organizations across the Philippines.

Across the 4 OLAP queries, applying indexing, we found improvements of 16%, 2%, 87%, and 8% when compared to the unoptimized reports. Each query was tested 18 times under different levels of optimization in MySQL workbench to record their execution times. Furthermore, query restructuring/ optimization on top of indexing yielded improvements of 35%, 38%, while table denormalization yielded improvements of 99.3%, and 99.7% compared to the unoptimized reports. Following optimization, the reports were ported into a Microsoft Power Bi dashboard application to provide a user interface and visualize the data provided by the reports.

Keywords

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

1. Introduction

Our target dataset consists of a set of medical appointments in the Philippines, primarily occurring from 2005 to 2024. This dataset contains records of anonymized patients and doctors, as well as the set of clinics wherein appointments occurred. The files consisted of 4 CSV files: appointments, px (patients), clinics, and doctors. Notably, the files “px” and “appointments” were significantly larger than previous data samples we had previously processed, with the appointments CSV file containing 2.2 GB of data or nearly 10 million lines.

A data warehouse model was generated based on the data needed to be stored. We then utilized ETL tools to clean, split, transfer, and load the data into a MySQL server, where 4 OLAP queries were performed before and after optimization techniques, including the addition of secondary indices, restructuring of SQL queries, and denormalization of tables.

The queries were collated and presented in an OLAP application to provide valuable information regarding current and historical trends in the medical industry in the Philippines, aimed at medical professionals and healthcare managers to allow them to make more informed decisions regarding the current medical landscape in the Philippines.

2. Data Warehouse

Our data warehouse schema was designed as a star schema as seen in Figure 1. We selected this schema for multiple reasons, these being simplicity, efficiency, and ease of use. As this table features a single central fact table and multiple connected dimension tables that do not have further sub-dimension tables, making our selected design classified as a star schema rather than a snowflake schema [1].

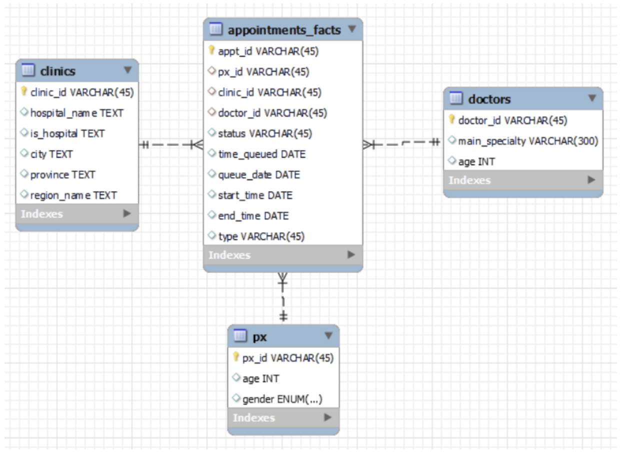


Figure 1. Dimensional Model

The schema utilizes 4 tables. A primary appointment fact table, containing all the data of a given appointment (this includes the start and ending time, when the appointment was initially requested/queued, whether the appointment was ever completed, and whether the appointment was a consultation or inpatient/hospital treatment), as well as 3 foreign key IDs belonging to the 3 other dimension tables - these being doctors, px, and clinics. These IDs indicate which anonymized doctor, patient, and clinic was involved in the appointment.

The px table simply contains the patient’s biological gender and their recorded age, as well as their assigned anonymized ID. The clinics table contains the clinic ID, regional information such as the city, province, and region of the clinic, as well as the hospital it is a branch of, if any at all. The doctors table contains their ID, age, and non-standardized specialty. This property specifically caused severe issues during the ETL process, requiring rigorous cleaning. Initially, the model did not contain any foreign key restraints/secondary indices to aid in data loading speed, as recommended by [6], [7].

We found that this data warehouse schema design provided the correct mix of usability, flexibility, simplicity, and data segregation, while also providing the additional benefit of closely correlating to the existing CSV files, simplifying the ETL pipeline. As such, we found no reason to further divide the tables beyond this point.

3. ETL Script

We utilized multiple ETL tools (Apache Nifi supplemented with Pandas) to extract the raw data from the files, split them, clean out unviable data/normalize text to a safe form, and remerge them when being loaded into our data warehouse hosted on a local MySQL server.

Initially, we simply utilized the extremely simple ETL script we utilized in the previous hands-on exercise designed to extract CSV data from a local file and load it into our database record using "GetFile -> PutDatabaseRecord". As there was no need for progressive updates as the data came from a local file, we simply utilized the same configuration for all CSVs. However, we ran into the issue of missing/invalid entries, sizes too large to be processed in a timely manner, invalid data types, and entries that disrupted Nifi's CSV reader due to special characters.

In order to solve the latter 2 issues, we implemented a string of "TextReplacement" processors set to "literal replacement mode" in order to remove all invalid characters (such as those found in doctors' specialties) and a ValidateCSV node to ensure all data was correct and in a workable format (such as ensuring age was an integer and not a float or unusable string). Our final Nifi model may be seen in Figure 2.

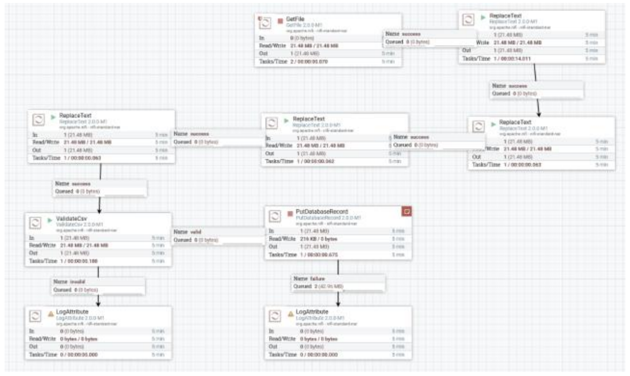
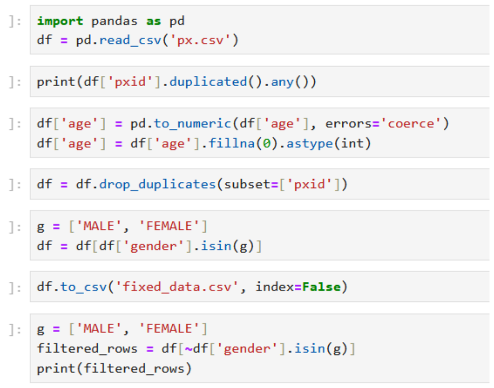


Figure 2. Apache Nifi Pipeline

While we were aware of Nifi's ability to split CSVs into smaller files, we were unable to make use of it in the first place, as our machine would begin to slow down as the file was gotten in the first place. As such, we utilized Jupyter Notebook and Pandas to examine, clean, and split both px and appointments before utilizing our Nifi model in Figure 2 to load it into our data warehouse.

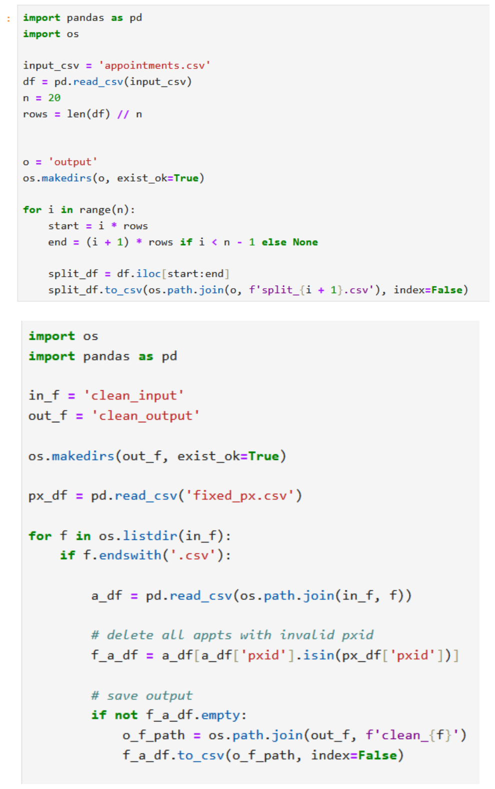
Using Pandas, we found that px contained multiple inconsistencies, including duplicate primary keys, invalid keys, and lines that simply were not valid entries. These rows were cleaned utilizing Pandas and the aforementioned Nifi ReplaceText processors.



```
]: import pandas as pd
   df = pd.read_csv('px.csv')

]: print(df['pxid'].duplicated().any())

]: df['age'] = pd.to_numeric(df['age'], errors='coerce')
   df['age'] = df['age'].fillna(0).astype(int)

]: df = df.drop_duplicates(subset=['pxid'])

]: g = ['MALE', 'FEMALE']
   df = df[df['gender'].isin(g)]

]: df.to_csv('fixed_data.csv', index=False)

]: g = ['MALE', 'FEMALE']
   filtered_rows = df[~df['gender'].isin(g)]
   print(filtered_rows)
```

Figure 3. JP Notebook Pandas PX Data Cleaning Pipeline

Lastly, we discovered that not all appointments contained valid foreign IDs. We utilized Pandas to split appointments.csv into 20 smaller files to be filtered. We removed all entries within each file which did not contain a valid px_id. Out of nearly 10 million entries, only ~300,000 were valid within our database. This allowed us to easily remerge the files and load it into our database using Apache Nifi with no further complications.

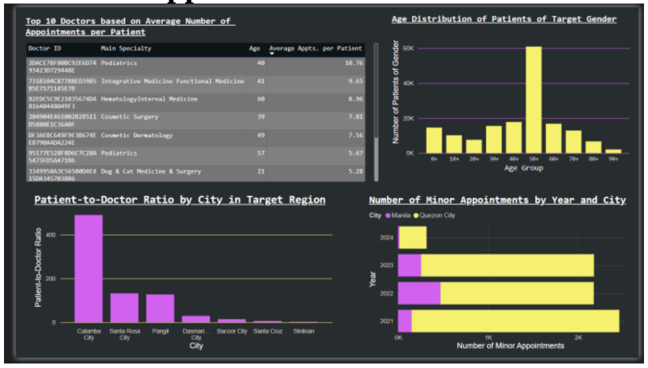


```
: import pandas as pd
  import os

  input_csv = 'appointments.csv'
  df = pd.read_csv(input_csv)
  n = 20
  rows = len(df) // n


  o = 'output'
  os.makedirs(o, exist_ok=True)

  for i in range(n):
      start = i * rows
      end = (i + 1) * rows if i < n - 1 else None

      split_df = df.iloc[start:end]
      split_df.to_csv(os.path.join(o, f'split_{i + 1}.csv'), index=False)
```

```
import os
import pandas as pd

in_f = 'clean_input'
out_f = 'clean_output'

os.makedirs(out_f, exist_ok=True)

px_df = pd.read_csv('fixed_px.csv')

for f in os.listdir(in_f):
    if f.endswith('.csv'):

        a_df = pd.read_csv(os.path.join(in_f, f))

        # delete all appts with invalid pxid
        f_a_df = a_df[a_df['pxid'].isin(px_df['pxid'])]

        # save output
        if not f_a_df.empty:
            o_f_path = os.path.join(out_f, f'clean_{f}')
            f_a_df.to_csv(o_f_path, index=False)
```

Figure 4. Jupyter Notebook Pandas Appointments Splitter & Appointments Cleaning Pipeline

4. OLAP Application

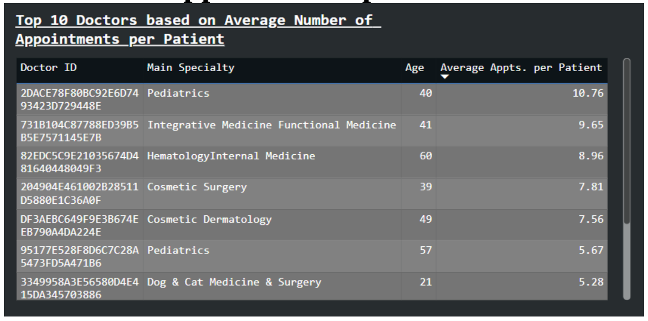

Top 10 Doctors based on Average Number of Appointments per Patient
Age Distribution of Patients of Target Gender
Number of Patients of Gender
60K
40K
20K
0K
0+
10+
20+
30+
40+
50+
60+
70+
80+
90+
Age Group
Patient-to-Doctor Ratio by City in Target Region
Patient-to-Doctor Ratio
400
200
0
Calamba City
Santa Rosa City
Pangil
Dasmari... City
Bacoor City
Santa Cruz
Siniloan
City
Number of Minor Appointments by Year and City
City ● Manila ● Quezon City
Year
2024
2023
2022
2021
0K
1K
2K
Number of Minor Appointments

Doctor ID	Main Specialty	Age	Average Appts. per Patient
2DACE78F80BC92E6D74 93423D729448E	Pediatrics	40	10.76
731B104C87788ED39B5 B5E7571145E7B	Integrative Medicine Functional Medicine	41	9.65
82EDC5C9E21035674D4 81640448049F3	HematologyInternal Medicine	60	8.96
204904E461002B28511 D5880E1C36A0F	Cosmetic Surgery	39	7.81
DF3AEBC649F9E3B674E EB790A4DA224E	Cosmetic Dermatology	49	7.56
95177E528F8D6C7C28A 5473FD5A471B6	Pediatrics	57	5.67
3349958A3E56580D4E4 15DA345703886	Dog & Cat Medicine & Surgery	21	5.28

Figure 5. OLAP Application Dashboard

Our OLAP application is intended for use by both medical professionals and those supervising and managing them, providing them with analytical reports regarding the appointments, clinics, patients, and individual doctors in the data warehouse. The application and the analytical reports composing it were designed to provide users with information about large-scale trends within the medical industry as well as performance statistics of specific regions, clinics, and individual medical professionals. These analytical reports are intended to provide important insights and allow users to make informed and data-driven decisions, such as allowing doctors to compare their statistics to the wider medical community and view which locations in the Philippines have the densest/most saturated population of other medical professionals. Our OLAP application consists of 4 primary reports, each utilizing an OLAP operation to produce new and meaningful information.

4.1 Rollup – Doctors with Highest Average Number of Appointments per Patient



Top 10 Doctors based on Average Number of Appointments per Patient

Doctor ID	Main Specialty	Age	Average Appts. per Patient
2DACE78F80BC92E6D74 93423D729448E	Pediatrics	40	10.76
731B104C87788ED39B5 B5E7571145E7B	Integrative Medicine Functional Medicine	41	9.65
82EDC5C9E21035674D4 81640448049F3	HematologyInternal Medicine	60	8.96
204904E461002B28511 D5880E1C36A0F	Cosmetic Surgery	39	7.81
DF3AEBC649F9E3B674E EB790A4DA224E	Cosmetic Dermatology	49	7.56
95177E528F8D6C7C28A 5473FD5A471B6	Pediatrics	57	5.67
3349958A3E56580D4E4 15DA345703886	Dog & Cat Medicine & Surgery	21	5.28

Figure 6. OLAP Application Rollup Query

We chose to provide the “Top 10 Doctors based on Average Number of Appointments per Patient”. This statistic essentially represents how likely a patient is to return to a doctor after receiving initial treatment. It could also indicate medical professionals who have successfully nurtured a continuous and ongoing relationship and trust amongst their clients. This information can be utilized by doctors to gauge whether their own metric is above or below average and act accordingly.

```sql
SELECT d.doctor_id, d.main_specialty, d.age,
    AVG(appt_per_pat_sq.appts_per_pat) AS avg_appointments_per_patient
FROM doctors d
JOIN(
    SELECT af.doctor_id, COUNT(appt_id) AS appts_per_pat
    FROM appointments_facts af
    GROUP BY af.doctor_id, af.px_id
)  AS appt_per_pat_sq
ON d.doctor_id = appt_per_pat_sq.doctor_id
GROUP BY d.doctor_id
ORDER BY avg_appointments_per_patient DESC
LIMIT 10;
```

Figure 7. Rollup SQL Query

This report utilizes a rollup operation in its subquery, firstly aggregating the total number of appointments for each existing pair of doctor IDs and patient IDs. This represents the number of appointments a single patient has had with a specific doctor. Then for each doctor in the doctors table, it then joins onto this subquery and returns the average number of appointments per patient for each doctor.

4.2 Drilldown – Patient to Doctor Ratio per City for a Given Region

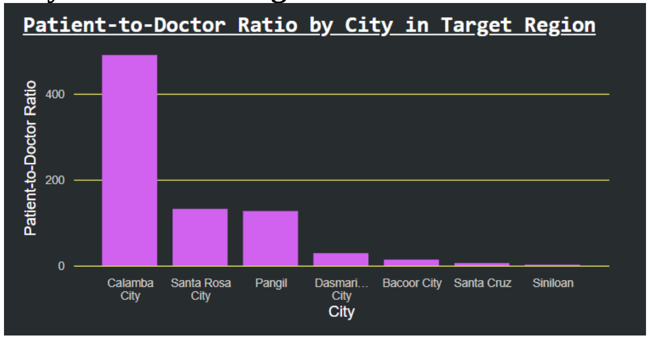


Figure 8. OLAP Application Drilldown Query

This report provides the Patient-to-Doctor ratio (calculated as the number of unique patients in a clinic divided by the number of unique doctors active in that clinic) per city within a given region. This statistic is essentially a measure of which areas are undersaturated and oversaturated with medical professionals. An area with an extremely high patient-to-doctor ratio is likely to have more job openings for doctors, and hospital managers may use this information to transfer doctors in/out of a given location due to demand or a lack of it.

```sql
SET @targetRegion = 'National Capital Region (NCR)';

SELECT p1.city, AVG(p1.nPatients / p2.nDoctors) AS patient_to_doctor_ratio FROM
    (SELECT c.city, c.clinic_id, COUNT(DISTINCT p.px_id) AS nPatients
    FROM clinics c JOIN appointments_facts af ON c.clinic_id = af.clinic_id AND c.region_name = @targetRegion
    JOIN px p ON af.px_id = p.px_id
    GROUP BY c.city, c.clinic_id) as p1
JOIN
    (SELECT c.city, c.clinic_id, COUNT(DISTINCT d.doctor_id) AS nDoctors
    FROM clinics c JOIN appointments_facts af ON c.clinic_id = af.clinic_id AND c.region_name = @targetRegion
    JOIN doctors d ON af.doctor_id = d.doctor_id
    GROUP BY c.region_name, c.city, c.clinic_id) as p2
ON p1.city = p2.city AND p1.clinic_id = p2.clinic_id
GROUP BY p1.city
ORDER BY patient_to_doctor_ratio DESC;
```

Figure 9. Drilldown SQL Query

This report is primarily a drilldown operation, as it takes a higher-level dimension (given region) and provides a more specific look into the city-level granular statistics. It functions by utilizing 2 subqueries. The first subquery finds the total amount of unique patients per clinic per city within the target region. The second does the same but counts the total amount of unique doctors instead. Both subqueries are then joined together using their related columns (city, clinic).

This resulting table contains 6 columns, with redundant city and clinic columns and two aggregate columns: nPatients and nDoctors. Finally, to find the average patient-doctor ratio, we grouped the table by city, and calculated the average of nPatients / nDoctors.

4.3 Slice – Age Distribution of Patients of a Given Gender

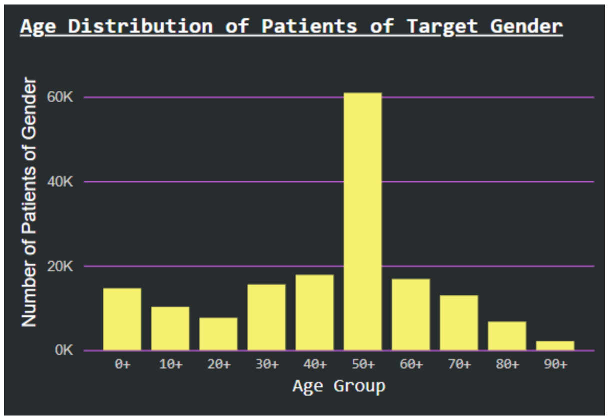
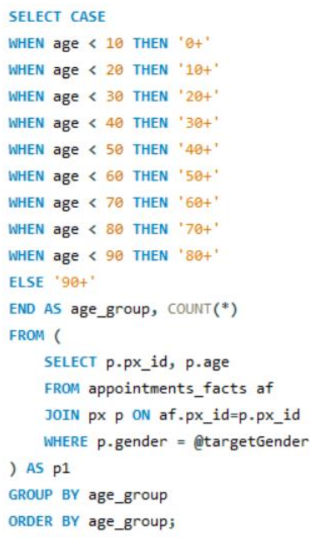


Figure 10. OLAP Application Slice Query

This report provides an age distribution in 10-year segregations for the number of all active patients (meaning only those who have appointments) of a target gender. This information provides a clinic and the medical professionals in it with insight on what demographics they are most likely to serve. It highlights certain trends, such as how a vast majority of both male and female patients are within 50-60 years old. This allows them to better plan and allocate their resources towards services, staff, and facilities that cater towards issues common within that age range for either gender.



```
SELECT CASE
WHEN age < 10 THEN '0+'
WHEN age < 20 THEN '10+'
WHEN age < 30 THEN '20+'
WHEN age < 40 THEN '30+'
WHEN age < 50 THEN '40+'
WHEN age < 60 THEN '50+'
WHEN age < 70 THEN '60+'
WHEN age < 80 THEN '70+'
WHEN age < 90 THEN '80+'
ELSE '90+'
END AS age_group, COUNT(*)
FROM (
    SELECT p.px_id, p.age
    FROM appointments_facts af
    JOIN px p ON af.px_id=p.px_id
    WHERE p.gender = @targetGender
) AS p1
GROUP BY age_group
ORDER BY age_group;
```

Figure 11. Slice SQL Query

This query utilizes a slice operation, defined as an operation that removes a dimension, returning values that only match a given value for that dimension. In this case, we utilize a subquery to return all patient IDs and age within appointments that match the target gender. We then utilize a CASE statement to sort them into their respective age ranges and aggregate them by age group.

4.4 Dice – Number of Minor Appts. By City Within Time Span

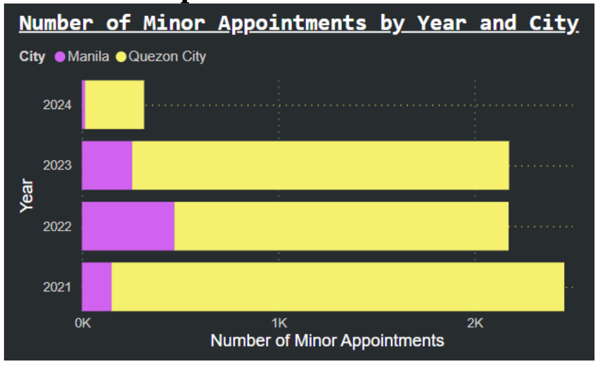
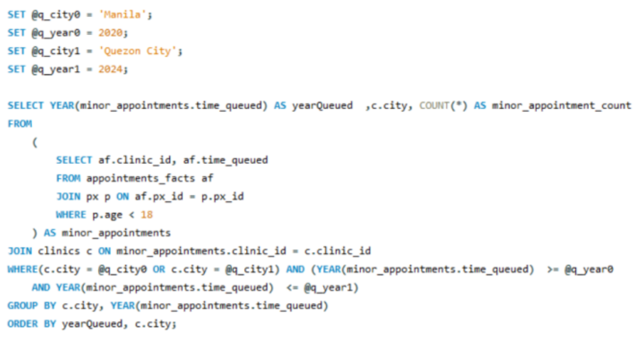


Figure 12. OLAP Application Dice Query

We defined “minor appointments” as appointments wherein the patient was below the age of 18. This report presents the number of minor appointments occurring within 2 cities across a given period of time. It compares the number of appointments between the 2 on a yearly basis and charts it to see rising/falling trends. This information is valuable towards healthcare professionals, especially those dealing with pediatrics and adolescent patients. It provides insight into which cities the services of pediatricians are in most high demand and indicates which cities have unique requirements and when that trend might have begun. This data can also be used by city-level officials when making policies that affect the healthcare and treatment of young citizens.



```
SET @q_city0 = 'Manila';
SET @q_year0 = 2020;
SET @q_city1 = 'Quezon City';
SET @q_year1 = 2024;

SELECT YEAR(minor_appointments.time_queued) AS yearQueued  ,c.city, COUNT(*) AS minor_appointment_count
FROM
    (
        SELECT af.clinic_id, af.time_queued
        FROM appointments_facts af
        JOIN px p ON af.px_id = p.px_id
        WHERE p.age < 18
    ) AS minor_appointments
JOIN clinics c ON minor_appointments.clinic_id = c.clinic_id
WHERE(c.city = @q_city0 OR c.city = @q_city1) AND (YEAR(minor_appointments.time_queued)  >= @q_year0
    AND YEAR(minor_appointments.time_queued)  <= @q_year1)
GROUP BY c.city, YEAR(minor_appointments.time_queued)
ORDER BY yearQueued, c.city;
```

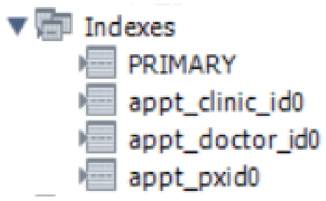
Figure 13. Dice SQL Query

This report can be considered an OLAP dice operation, as it produces a much smaller subcube than the primary OLAP cube, containing only 2 cities and a small number of years. It functions by firstly selecting all minor appointments within a subquery, regardless of time or location in which it occured. It then joins onto the clinic table to access location information. It uses a set of where statements to filter out all appointments not within either city and not within the time span. It then aggregates this data based on city and year.

5. Query Processing and Optimization

Data Optimization refers to the process of improving the speed at which reports are generated and queries are executed through database design, optimization of the SQL query itself, and implementation of hardware-level optimizations [4].

We chose to implement 3 optimization methods to our reports. Specifically, indexing, query restructuring, and table denormalization. Indexing refers to the creation of secondary indices within columns commonly used in queries – specifically, foreign keys utilized in the appointments facts table [5].



```
Indexes
    PRIMARY
    appt_clinic_id0
    appt_doctor_id0
    appt_pxid0
```

Figure 14. Primary & Secondary Indices in appointments_facts

Query restructuring would be utilized to reform the query to produce the same outputs while improving efficiency, such as by removing problematic SQL constructs (such as subqueries) and minimizing the number of operations the server must perform to lower the overall disk access cost and execution time for a given query [8]. Examples of this and related figures are seen in subsections 6.1.1 and 6.1.4, where the figures have been placed to be better contextualized by analysis and comparison.

Finally, we denormalized certain pieces of data to other tables. Denormalization refers to the redundant storage of existing information (or information that is readily calculable through existing data such as aggregation processes) in another table [2]. While this increases storage requirements, this allows us to bypass needing to perform many time-consuming joins or even aggregations. Examples of this and related figures are seen in subsections 6.1.2 and 6.1.3 for similar reasons as above. Furthermore, SQL code snippets used for creation of denormalized columns and population of those columns are visible in Appendix B and C, while the final denormalized database schema is visible in Appendix A, where it has been placed for brevity.

6. Results and Analysis

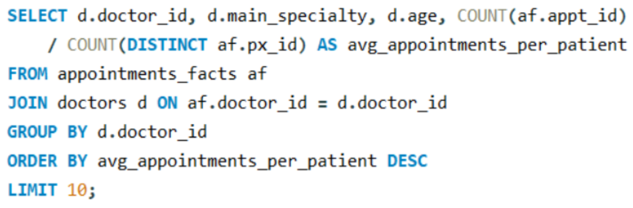
6.1 Performance Testing and Analysis

We performed a series of 18 timed executions on each of the 4 OLAP queries – 6 at 3 different levels of optimization per query utilizing MySQL Workbench's built-in execution timer. These tests were performed utilizing the full dataset within the warehouse. This would allow us to see the effect each optimization had on each query.

Initially, only the primary indices existed in the table, created automatically by the database upon creation of each table's primary key. All 4 queries would be timed with no secondary indices, before secondary indices (and foreign key restraints) would be created. Each query would be retimed under this process before either undergoing query restructuring (Rollup and Dice) or table denormalization (Drilldown and Slice). Finally, each report would be retimed with both indexing and their specific optimization. This is the final version utilized in the OLAP application.

6.1.1 Rollup Optimization

In order to optimize the query "Top 10 Doctors based on Average Number of Appointments per Patient", we utilized query restructuring/optimization through removal of the subquery. Rather than first querying for the number of appointments a patient has had with a specific doctor, the revised query simply calculates the average by dividing the total number of appointments a doctor has with the number of unique patients he's had. This arrives at the same result but far faster.



```sql
SELECT d.doctor_id, d.main_specialty, d.age, COUNT(af.appt_id)
    / COUNT(DISTINCT af.px_id) AS avg_appointments_per_patient
FROM appointments_facts af
JOIN doctors d ON af.doctor_id = d.doctor_id
GROUP BY d.doctor_id
ORDER BY avg_appointments_per_patient DESC
LIMIT 10;
```

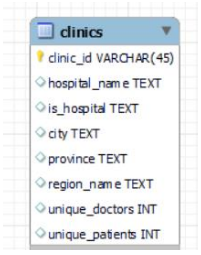
Figure 15. Restructured Rollup SQL Query

Table 1. Rollup SQL Query Performance Comparison

Trial #	1	2	3	4	5	6	AVE	Compared to Unoptimized	Compared to Indexed
UNOPTIMIZED	1.813s	1.562s	1.609s	1.422s	1.422s	1.513s	1.557s	--	--
INDEXED	1.296s	1.281s	1.297s	1.281s	1.328s	1.329s	1.302s	16.378%	--
QUERY RESTRUCTURING	0.984s	0.953s	0.937s	0.953s	0.953s	0.938s	0.953s	38.793%	26.805%

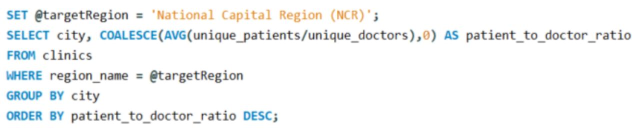
6.1.2 Drilldown Optimization

To calculate the patient to doctor ratio per city for a given region in a more efficient manner, we chose to denormalize and precalculate the number of distinct doctors working at and patients checking in at a given clinic. This is also plausible from a real-world perspective, where a given clinic would naturally have both a staff list and patient database. The count of distinct patients and doctors was then used in the same formula as previously, resulting in exponentially faster results. All trial were conducted on the NCR region, as it contained the largest number of clinics (36,216).



```
clinics
clinic_id VARCHAR(45)
hospital_name TEXT
is_hospital TEXT
city TEXT
province TEXT
region_name TEXT
unique_doctors INT
unique_patients INT
```

Figure 16. Denormalized Clinics Table



```sql
SET @targetRegion = 'National Capital Region (NCR)';
SELECT city, COALESCE(AVG(unique_patients/unique_doctors),0) AS patient_to_doctor_ratio
FROM clinics
WHERE region_name = @targetRegion
GROUP BY city
ORDER BY patient_to_doctor_ratio DESC;
```

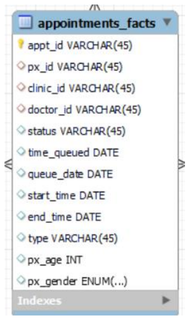
Figure 17. Drilldown SQL Query using Denormalized Clinics

Table 2. Drilldown SQL Query Performance Comparison

Trial #	1	2	3	4	5	6	AVE	Compared to Unoptimized	Compared to Indexed
UNOPTIMIZED	0.938s	0.922s	0.906s	0.922s	0.937s	0.937s	0.927s	--	--
INDEXED	0.781s	0.813s	0.75s	0.75s	0.65s	0.812s	0.759s	2.452%	--
DENORMALIZED	0.078s	0.078s	0.062s	0.062s	0.062s	0.062s	0.067s	99.717%	99.710%

6.1.3 Slice Optimization

Similarly to the optimization used in the patient to doctor ratio per city, we chose to utilize denormalization to optimize the calculation of age distributions of patients of a given gender. We choose to redundantly store the patient age and gender within the appointments table. In this manner, no joins are needed to calculate this report. As such, we observe similarly exponential gains in execution time. Trials for this query were divided into “MALE” and “FEMALE” queries to account for both possibilities.



appointments_facts
- appt_id VARCHAR(45)
- px_id VARCHAR(45)
- clinic_id VARCHAR(45)
- doctor_id VARCHAR(45)
- status VARCHAR(45)
- time_queued DATE
- queue_date DATE
- start_time DATE
- end_time DATE
- type VARCHAR(45)
- px_age INT
- px_gender ENUM(...)

Indexes

Figure 18. Denormalized Appointments Table

```sql
SET @targetGender = 'FEMALE';
SELECT CASE
WHEN px_age < 10 THEN '0+'
WHEN px_age < 20 THEN '10+'
WHEN px_age < 30 THEN '20+'
WHEN px_age < 40 THEN '30+'
WHEN px_age < 50 THEN '40+'
WHEN px_age < 60 THEN '50+'
WHEN px_age < 70 THEN '60+'
WHEN px_age < 80 THEN '70+'
WHEN px_age < 90 THEN '80+'
ELSE '90+'
END AS age_group, COUNT(*) AS number_of_patients
FROM appointments_facts
WHERE px_gender = @targetGender
GROUP BY age_group
ORDER BY age_group;
```

Figure 19. Slice SQL Query using Denormalized Appointments

Table 3. Slice SQL Query Performance Comparison - Male

Trial #	1	2	3	AVE	Compared to Unoptimized	Compared to Indexed
UNOPTIMIZED	35.988s	34.762s	34.485s	35.078s	--	--
INDEXED	4.422s	4.437s	4.468	4.442s	87.337%	--
DENORMALIZED	0.25s	0.25s	0.25	0.25s	99.287%	94.372%

Table 4. Slice SQL Query Performance Comparison - Female

Trial #	1	2	3	AVE	Compared to Unoptimized	Compared to Indexed
UNOPTIMIZED	34.484s	34.156s	34.392s	34.344s	--	--
INDEXED	4.468s	4.437s	4.453s	4.453s	87.034%	--
DENORMALIZED	0.265s	0.265s	0.265s	0.265s	99.228%	94.049%

6.1.4 Dice Optimization

Our dice operation was to calculate and compare the number of minor appointments produced in a yearly basis over a span of X years between 2 target cities. We chose to optimize this similarly to our Rollup operation, via query restructuring. Originally, this query utilized a subquery to compile all appointments involving minors (as well as filtering out entries outside the time range), before then joining onto clinics to filter out entries not within the target cities. This was restructured to not use a subquery by starting at the appointments table and joining onto both patients and clinics. Filtering occurred within the main query instead of the subquery. Trials were all held between “Manila” and “Quezon City”, the 2 cities with the largest number of clinics, with 23,566 and 4,603 each respectively. Trials were split between 2021-2024 cases and 2017-2024 cases to observe the effect of a larger sample size on the performance increases.

```sql
SET @q_city0 = 'Manila';
SET @q_year0 = 2020;
SET @q_city1 = 'Quezon City';
SET @q_year1 = 2024;

SELECT YEAR(af.time_queued) AS yearQueued, c.city, COUNT(*) AS minor_appointment_count
FROM clinics c
JOIN appointments_facts af ON c.clinic_id = af.clinic_id
JOIN px p ON af.px_id = p.px_id
WHERE (c.city = @q_city0 OR c.city = @q_city1) AND
    p.age < 18 AND
    YEAR(af.time_queued) BETWEEN @q_year0 AND @q_year1
GROUP BY c.city, YEAR(af.time_queued)
ORDER BY yearQueued, c.city;
```

Figure 20. Restructured Dice SQL Query

Table 5. Dice SQL Query Performance Comparison (2021-2024)

Trial #	1	2	3	AVE	Compared to Unoptimized	Compared to Indexed
UNOPTIMIZED	2.488s	2.485s	2.347s	2.44s	--	--
INDEXED	2.406s	2.172s	2.122s	2.233s	8.484%	--
QUERY RESTRUCTURING	1.578s	1.578s	1.61s	1.589s	34.877%	28.840%

Table 6. Dice SQL Query Performance Comparison (2017-2024)

Trial #	1	2	3	AVE	Compared to Unoptimized	Compared to Indexed
UNOPTIMIZED	17.812s	15.234s	14.578s	15.875s	--	--
INDEXED	12.484s	11.948s	12.016s	12.149s	23.471%	--
QUERY RESTRUCTURING	10.203s	10.25s	10.219s	10.224s	35.597%	15.845%

6.2 Function Testing

Function testing was performed early on in the development process on a small subset of the provided data. Utilizing Jupyter Notebook to split the initial CSV file into far smaller samples, the members of the group systematically attempted to create an SQL statement that fulfilled the task independently of one another. Due to the small size of the data, it was simple to manually calculate and verify our answers. Furthermore, as solutions were developed independently, we were able to cross-check one another to catch minor mistakes. Future optimized versions of the queries were then checked against the already tested unoptimized queries to ensure consistent results.

A similar process was used in the ETL process, involving testing the Apache Nifi pipeline on an extremely small subset of the data before manually verifying it. Pandas was used to ensure that no foreign key violations would appear by pruning all entries with invalid keys, as mentioned previously in Section 3, ETL Script.

The function testing process allowed us to be confident in the validity of both the data we were processing as well as the information output by the OLAP application. By performing this task early on in the process, we minimized the amount of wasted work due to errors later on in development and ensured no errors were present in the final product.

6.3 Analysis of Results

The results indicate that all 3 optimization techniques utilized (indexing, query restructuring/optimization, and table denormalization) produced significant and meaningful improvements to the speed of the queries, although to different magnitudes.

Firstly, indexing consistently provided a large optimization to any query that utilized the generated indices, ranging from 27% to 87% (The exception being the slice query, only showing a marginal 2% increase). These results highlight the strength of indexing, and why it is commonly recommended as the first optimization to be made. Although the results shown are not as strong as denormalization, they still provided major improvements, considering how simple it is to set up within a database, being automatically created upon foreign key assignment.

Table denormalization provided an exceedingly strong improvement, cutting down from both optimized and indexed queries by orders of magnitude (roughly 99.7% faster in both cases). Reports once at risk of not generating within the time limit were completed in less than a second. This result corroborates our theoretical understanding of what causes SQL queries to slow down, as it minimizes the number of operations and overall disk access cost [8] it takes for the database to perform the query by culling the need to perform joins at all.

This performance increase does not come without a cost, however. Denormalizing data requires increased storage space and is much harder to maintain in an environment with continuous updates. There is the possibility of desynchronization of data due to failure to update one or more columns, causing data and reports to become inaccurate [2]. As such, caution must be taken with this method.

Query restructuring proved to be the most time consuming of all three methods, requiring a rigorous understanding of the inner workings of SQL constructs and using logic/relational algebra concepts to minimize the work being done. It provided a 15% - 28% increase in comparison to the indexed but unoptimized SQL queries.

While it may seem that query restructuring is not worth the effort it takes to perform, it should be noted that it is the only method utilized in our data warehouse that has strictly no side-effects whatsoever. Indices affect write time, denormalized data must be carefully maintained or may become desynchronized or outdated, both require additional space, but any time improvements made via query restructuring are strict improvements over previous attempts. As such, any effort and learned skill that goes towards providing more efficient SQL queries will result in inarguably better report generation in every possible scenario.

7. Conclusion

This project required the use of all previous database management knowledge, as well as the learning of new data wrangling tools and optimization techniques. Most glaringly, this project highlighted the need for familiarity with the ETL process tools such as NiFi, which was necessary to clean and load the large quantities of data being handled.

This project also highlighted the difficulties that come with real world data. Previous projects utilized sanitize sample data to learn with. However, the dataset provided to us here was messy, large, and extremely difficult to manage. Previous tools utilized such as Excel became useless, as the data was too large to even open without the application crashing. We were forced to confront a situation wherein the data we were using was both too dirty and unpredictable to be simply inserted into Apache Nifi, but too large to examine and troubleshoot by hand. We were forced to adapt and learn new ways to analyze, split, transform, and utilize this data in a way we had never done before.

The process of working with this dataset also showed the importance of problem-solving and utilizing logic and design to overcome issues. Initially, we were completely unable to load or view the appointment data whatsoever due to its size. However, we realized that the data was bloated, as it contained over 9 million lines of invalid data. We realized we could prune this data to not only clean our data warehouse, but also load it on our struggling machines, solving 2 issues at once. This shows the importance of critical thinking outside of tasks like query formation and optimization.

One of our largest realizations came not from the process of query optimizations, but from the ETL process in the first place. Initially, we viewed Apache Nifi as an ETL tool able to easily handle any and all challenges. While it is true it is exceedingly capable and efficient, we found that under real-world conditions, such as limited learning time, slower machines, and inconvenient sizes of data, we were forced to adapt to other ETL tools to compensate. Pandas acted as an excellent way to manually clean the provided data and split it when Apache Nifi was simply to slow and cumbersome. This process highlighted the importance of branching out and gaining proficiency in more than one avenue for ETL and the need for flexibility in database management.

Finally, this project showed us that data optimization is a necessity. Initially, we were under the impression that data optimization techniques such as indexing and query restructuring would provide marginal to moderate effects, and certainly would not be needed for non-industrial purposes such as ours. However, over the course of this project, the need for faster queries became more and more obvious and necessary for the creation of our OLAP application. In the end, we became acutely aware of the strong need for database optimization tools like indexing, query restructuring, and data denormalization as tools in database management.

It also highlights the different weaknesses and use cases between the 3 options explored in this paper. It became clear why indexing was ubiquitous (and automatically generated by the database software in some cases), as the performance increase was immediately visible for nearly no effort. It highlighted the benefits of denormalizing data, but also the issues that could arise from outdated/desynchronized data causing incorrect reports. Finally, it also showed the immediate consequences of careless SQL query formulation and the need to minimize the amount of work placed on the server to maximize efficiency.

8. References

[1] Simplilearn (2022, December 27). Star Schema vs Snowflake Schema: Key Differences Between The Two. *Simplilearn.* https://www.simplilearn.com/star-schema-vs-snowflake-schema-article

[2] B., G., & B., A. (2023, September 12). When and how you should denormalize a relational database. *RubyGarage.*

https://rubygarage.org/blog/database-denormalization-with-examples

[3] Microsoft. (2017, July 24). *Pass parameter to SQL queries statement using Power Bi. Microsoft Learn.* https://learn.microsoft.com/en-us/shows/mvp-azure/pass-parameter-to-sql-queries-statement-using-power-bi

[4] Roy, A. (2023, May 23). What is SQL optimization? how to do it?. *LinkedIn.* https://www.linkedin.com/pulse/what-sql-optimization-how-do-amit-roy

[5] MySQL. (n.d.). MySQL :: MySQL 8.0 Reference Manual :: 8.3 Optimization and Indexes. *MySQL :: Developer Zone.* https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html/

[6] Johnson, T. (2014, June 29). Is it always faster to create indexes after loading data?. Database Administrators Stack Exchange. Retrieved February 20, 2024, from https://dba.stackexchange.com/questions/69299/is-it-always-faster-to-create-indexes-after-loading-data

[7] Darling, E. (2022, May 16). How constraints and foreign keys can hurt data loading performance in SQL server. Darling Data. Retrieved February 20, 2024, from https://erikdarling.com/data-loading-and-referential-integrity

[8] Sachdeva, S. (2023, December 26). A detailed guide on SQL query optimization. Analytics Vidhya. Retrieved February 20, 2024, from https://www.analyticsvidhya.com/blog/2021/10/a-detailed-guide-on-sql-query-optimization

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 provide general direction and guidance for the purpose of learning MySQL and to operate Jupyter Notebook. 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

Chua: Generated Slice and Drilldown Query. Accomplished Exercise 4. Generated Optimizations for Drilldown and Slice Queries. Performed research. Aided in writing activity report.

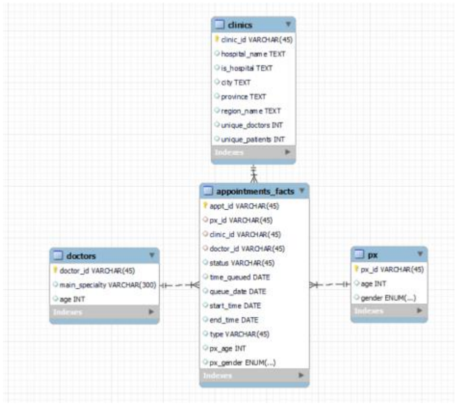
Lim: Generated Drilldown and Rollup Query. Generated Optimizations for All Queries. Performed ETL. Aided in writing activity report. Created OLAP Application.

Ngo: Created database schema. Generated Dice and Slice Query. Generated Optimizations for Dice and Slice Queries. Aided in writing activity report.

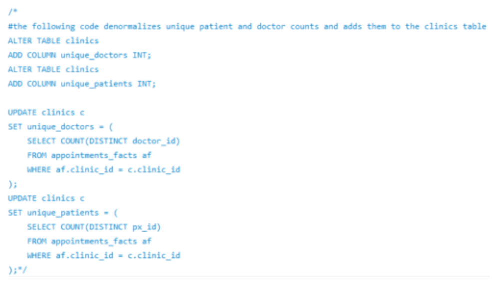
Santiago: Generated Rollup and Dice Query. Generated Optimizations for Dice and Rollup Queries. Accomplished Exercise 4. Aided in writing activity report.

A. Appendix

The following section contains supplementary SQL code snippets and figures not directly shown within the main text of the report to aid viewer comprehension. All sample query results are the same for both pre- and post-optimization, and as such, are only included once.

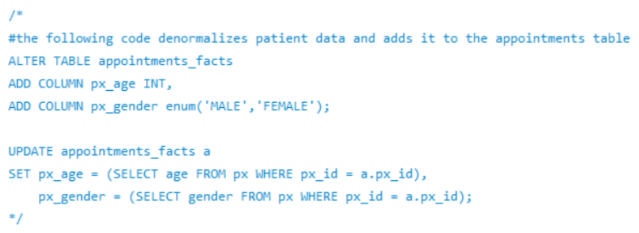


Appendix A. Optimized/Denormalized Dimensional Model



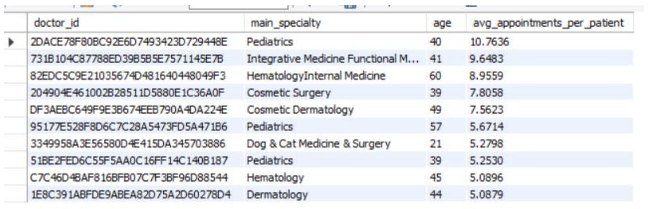
```
/*
#the following code denormalizes unique patient and doctor counts and adds them to the clinics table
ALTER TABLE clinics
ADD COLUMN unique_doctors INT;
ALTER TABLE clinics
ADD COLUMN unique_patients INT;

UPDATE clinics c
SET unique_doctors = (
    SELECT COUNT(DISTINCT doctor_id)
    FROM appointments_facts af
    WHERE af.clinic_id = c.clinic_id
);
UPDATE clinics c
SET unique_patients = (
    SELECT COUNT(DISTINCT px_id)
    FROM appointments_facts af
    WHERE af.clinic_id = c.clinic_id
);*/
```

Appendix B. Clinic Denormalization SQL Code



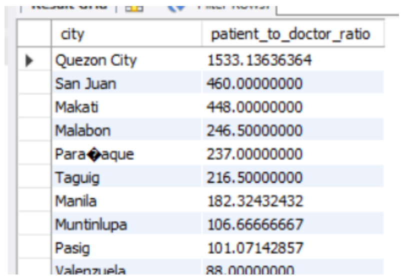
```
/*
#the following code denormalizes patient data and adds it to the appointments table
ALTER TABLE appointments_facts
ADD COLUMN px_age INT,
ADD COLUMN px_gender enum('MALE','FEMALE');

UPDATE appointments_facts a
SET px_age = (SELECT age FROM px WHERE px_id = a.px_id),
    px_gender = (SELECT gender FROM px WHERE px_id = a.px_id);
*/
```

Appendix C. Appointments Denormalization SQL Code



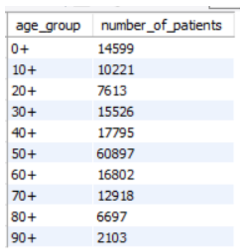
doctor_id	main_specialty	age	avg_appointments_per_patient
2DACE78F80BC92E6D7493423D729448E	Pediatrics	40	10.7636
731B104C87788ED39B5B5E7571145E7B	Integrative Medicine Functional M...	41	9.6483
82EDC5C9E21035674D481640448049F3	HematologyInternal Medicine	60	8.9559
204904E461002B28511D5880E1C36A0F	Cosmetic Surgery	39	7.8058
DF3AEBC649F9E3B674EEB790A4DA224E	Cosmetic Dermatology	49	7.5623
95177E528F8D6C7C28A5473FD5A471B6	Pediatrics	57	5.6714
3349958A3E56580D4E415DA345703886	Dog & Cat Medicine & Surgery	21	5.2798
51BE2FED6C55F5AA0C16FF14C140B187	Pediatrics	39	5.2530
C7C46D4BAF816BFB07C7F3BF96D88544	Hematology	45	5.0896
1E8C391ABFDE9ABEA82D75A2D60278D4	Dermatology	44	5.0879

Appendix D. Rollup Query Results



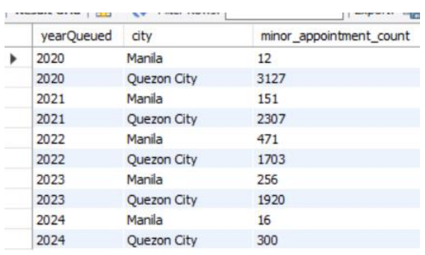
city	patient_to_doctor_ratio
Quezon City	1533.13636364
San Juan	460.00000000
Makati	448.00000000
Malabon	246.50000000
Para�aque	237.00000000
Taguig	216.50000000
Manila	182.32432432
Muntinlupa	106.66666667
Pasig	101.07142857
Valenzuela	88.00000000

Appendix E. Drilldown Query Results



age_group	number_of_patients
0+	14599
10+	10221
20+	7613
30+	15526
40+	17795
50+	60897
60+	16802
70+	12918
80+	6697
90+	2103

Appendix F. Slice Query Results



yearQueued	city	minor_appointment_count
2020	Manila	12
2020	Quezon City	3127
2021	Manila	151
2021	Quezon City	2307
2022	Manila	471
2022	Quezon City	1703
2023	Manila	256
2023	Quezon City	1920
2024	Manila	16
2024	Quezon City	300

Appendix G. Dice Query Results