



MCO1: Query Processing

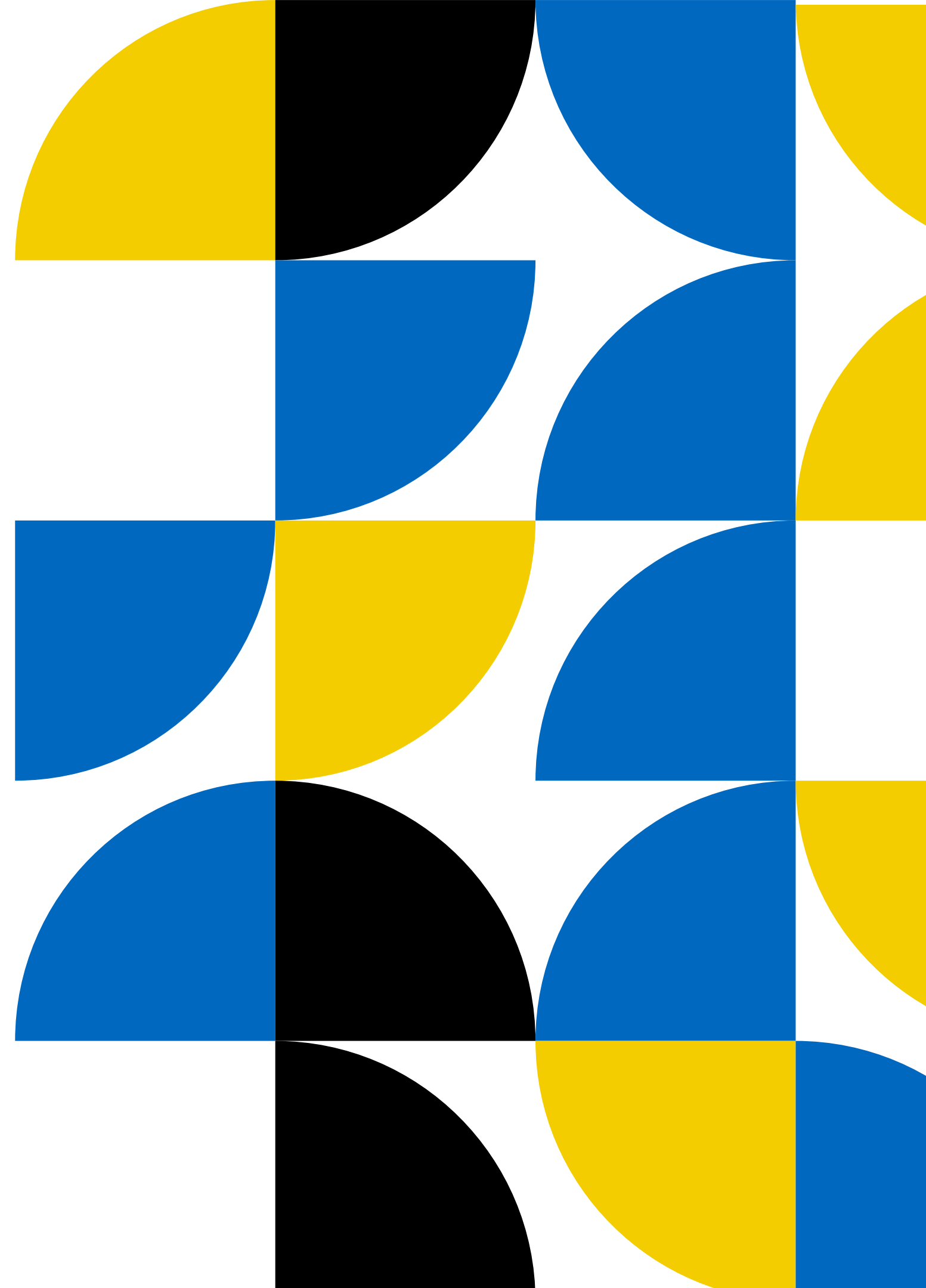
STADVDB - S12 || Group 8

CHUA, EDRIC JARVIS UY

LIM, RYDEL RIDLEY ANG

NGO, RENDELL CHRISTIAN JIM

SANTIAGO, NIKOLAI ANDRE



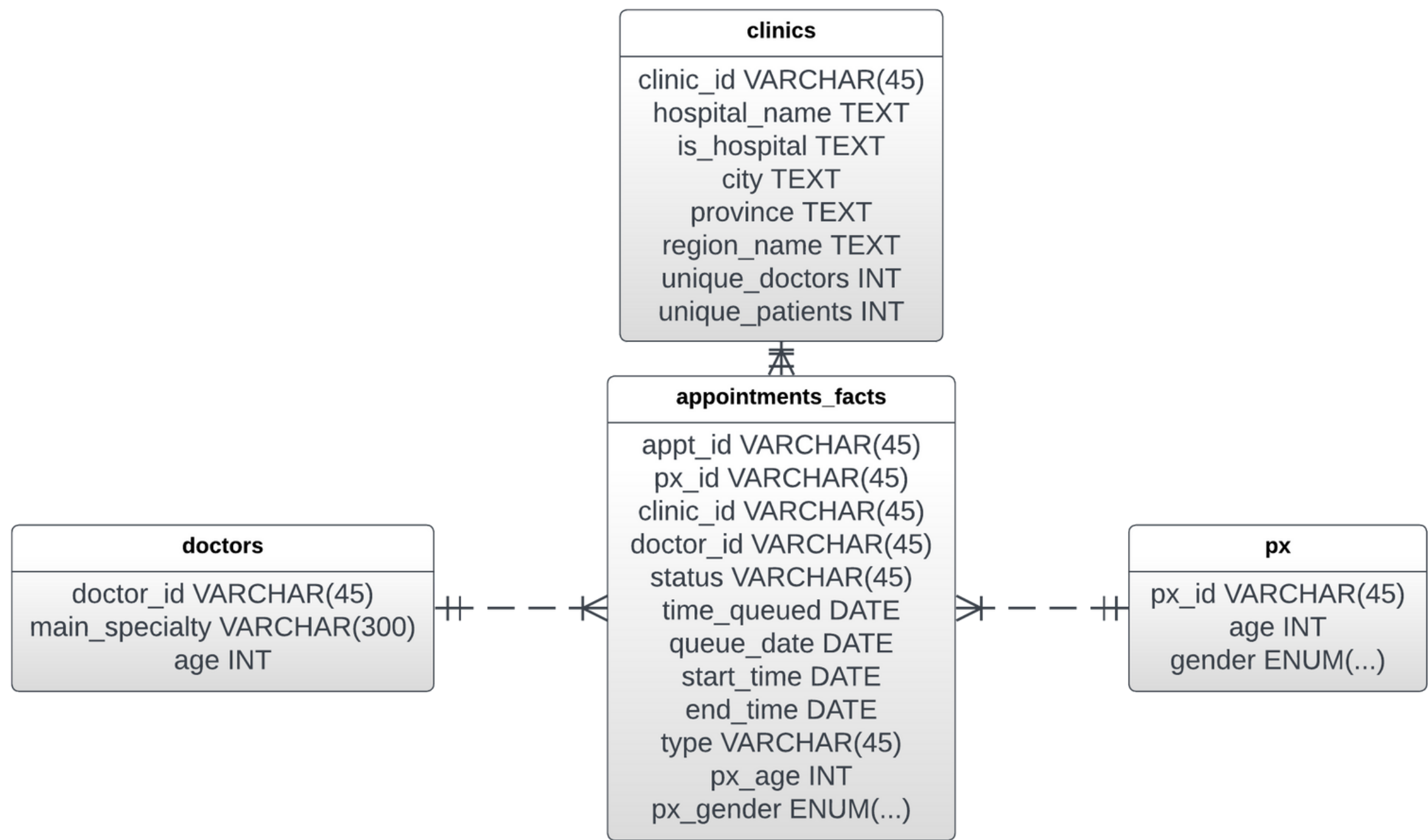
Data Warehouse Design

We used a star schema to represent the different tables. This ensured that relating the data to each other was easy, efficient, and simple.

appointments_facts is our fact table, containing foreign keys pointing to three dimension tables: px, doctors, and clinics.



Data Warehouse Design





ETL Script



Tools Used:

Jupyter Notebook
NIFI Nodes
Pandas

I. Cleaning

After scanning the CSV files containing the records, we found a large number of inconsistent data, invalid and missing rows, and invalid characters.

To solve this, we made use of ReplaceText nodes within the NIFI script which removed all invalid characters (especially within doctor's specialties) and ValidateCSV to ensure correct data types for every column.

Additionally, we used Pandas, a python data analysis library, in conjunction with Jupyter Notebook to more efficiently spot and clean invalid entries.





Tools Used:

Apache Nifi

ETL Script

II. Data Importing

For actual data loading, we chose to use exclusively Nifi.

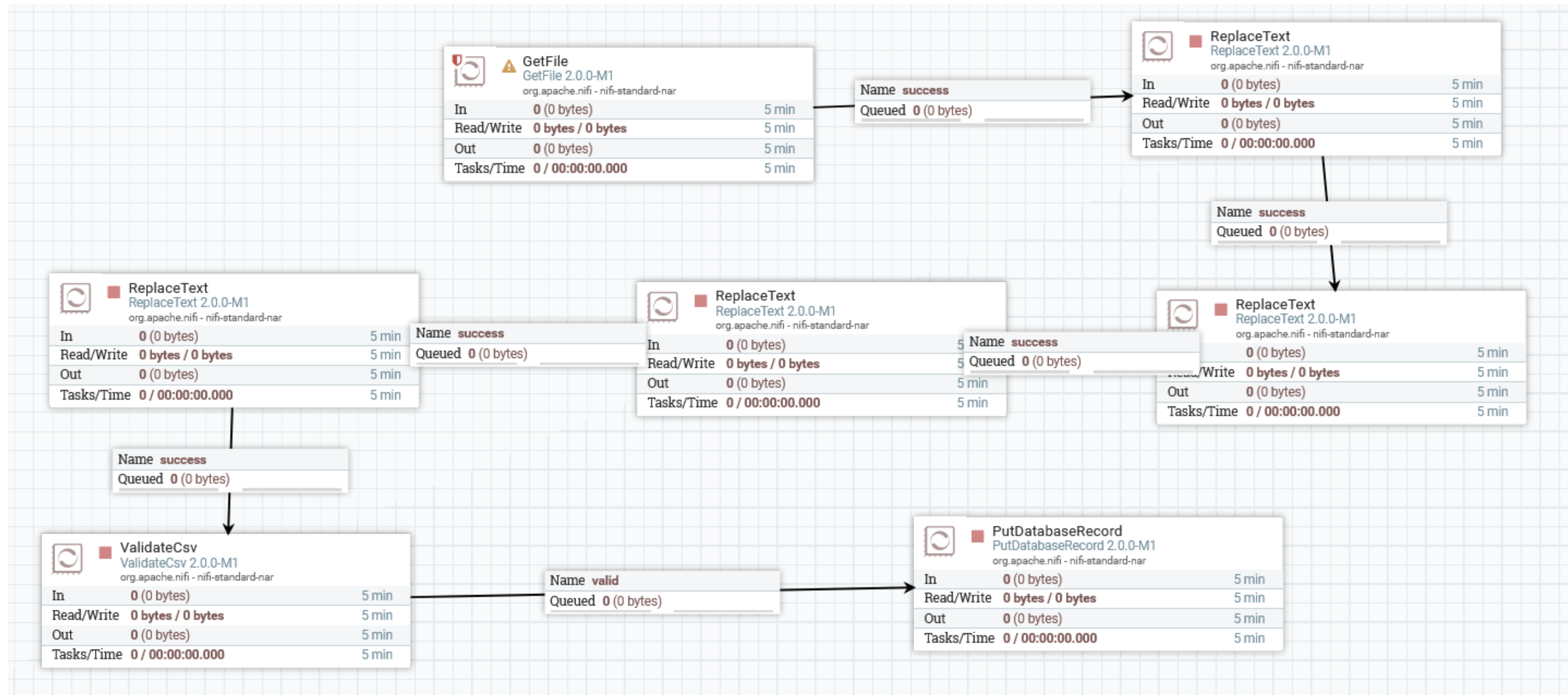
The ETL script we used is extremely simple. It consists mainly of the 'GetFile -> PutDatabaseRecord' nodes, with ValidateCSV and ReplaceText nodes between.

This was done for the four CSV files: px.csv, doctors.csv, clinics.csv and appointments.csv.



ETL Script

The screenshot displays an Apache NiFi workflow on a grid background. The workflow consists of several processing nodes and flow connectors. At the top left is a **GetFile** node (GetFile 2.0.0-M1) with a warning icon. It has a status box showing 'Name success' and 'Queued 0 (0 bytes)'. Its output flows into a **ReplaceText** node (ReplaceText 2.0.0-M1) at the top right. This node also has a status box showing 'Name success' and 'Queued 0 (0 bytes)'. Below the top-left **GetFile** node is another **ReplaceText** node. Below the top-right **ReplaceText** node is a third **ReplaceText** node. In the bottom left is a **ValidateCsv** node (ValidateCsv 2.0.0-M1). In the bottom right is a **PutDatabaseRecord** node (PutDatabaseRecord 2.0.0-M1). Flow connectors link the nodes: from the top-left **GetFile** to the top-right **ReplaceText**; from the top-right **ReplaceText** to the middle-right **ReplaceText**; from the middle-right **ReplaceText** to the bottom-right **PutDatabaseRecord**; from the bottom-left **ValidateCsv** to the bottom-right **PutDatabaseRecord**; and from the middle-left **ReplaceText** to the bottom-left **ValidateCsv**. Each node displays its name, version, and a table of metrics: In, Read/Write, Out, and Tasks/Time, each with a 5-minute interval. The status boxes for the **GetFile** and **ValidateCsv** nodes show 'Name success' and 'Queued 0 (0 bytes)', while the others show 'Name valid' and 'Queued 0 (0 bytes)'. The top-left **GetFile** node has a warning icon, while the others have a success icon.



OLAP Application

- We generated 4 queries utilizing 4 distinct OLAP operations - rollup, drilldown, slice, and dice.
- The OLAP application implements a visual representation for these 4 reports and 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.

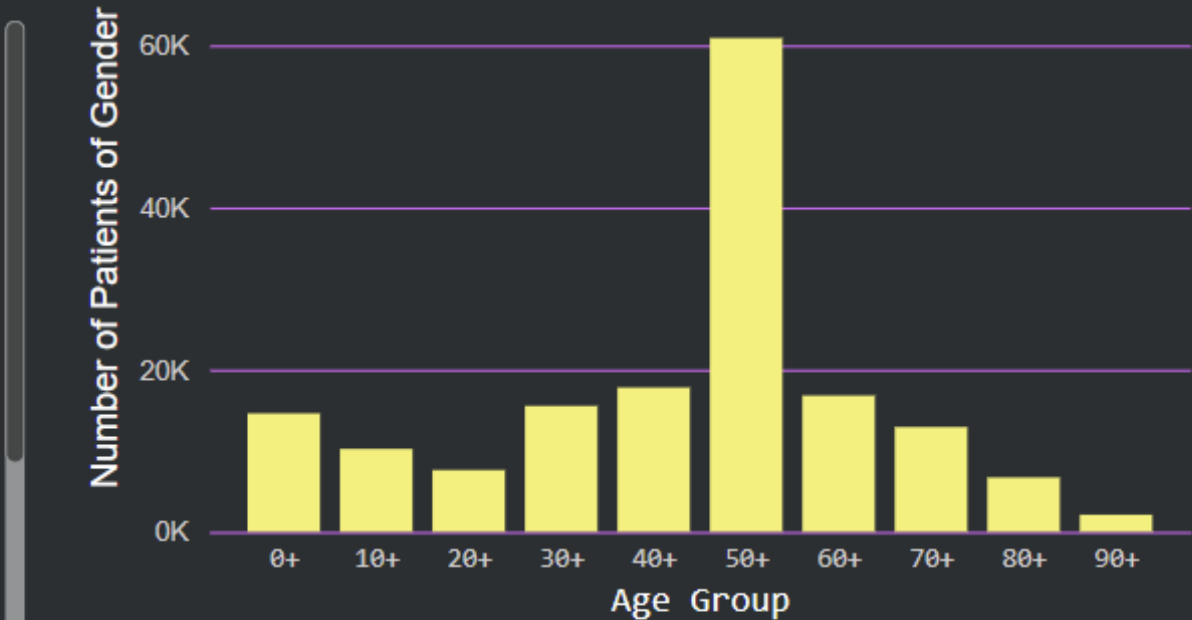


Dashboard

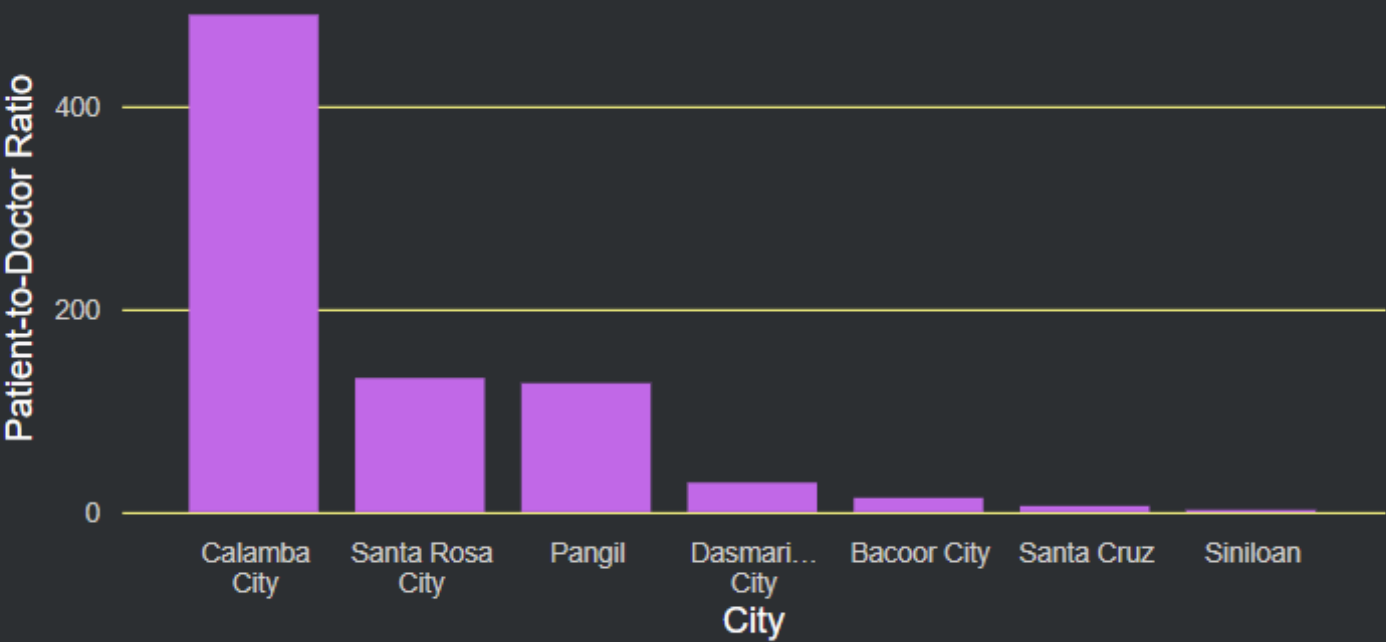
Top 10 Doctors based on Average Number of Appointments per Patient

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
DF3AEBBC649F9E3B674E EB790A4DA224E	Cosmetic Dermatology	49	7.56
95177E528F8D6C7C28A 5473FD5A471B6	Pediatrics	57	5.67
3349958A3E56580D4E4 15DA345703886	Dog & Cat Medicine & Surgery	21	5.28

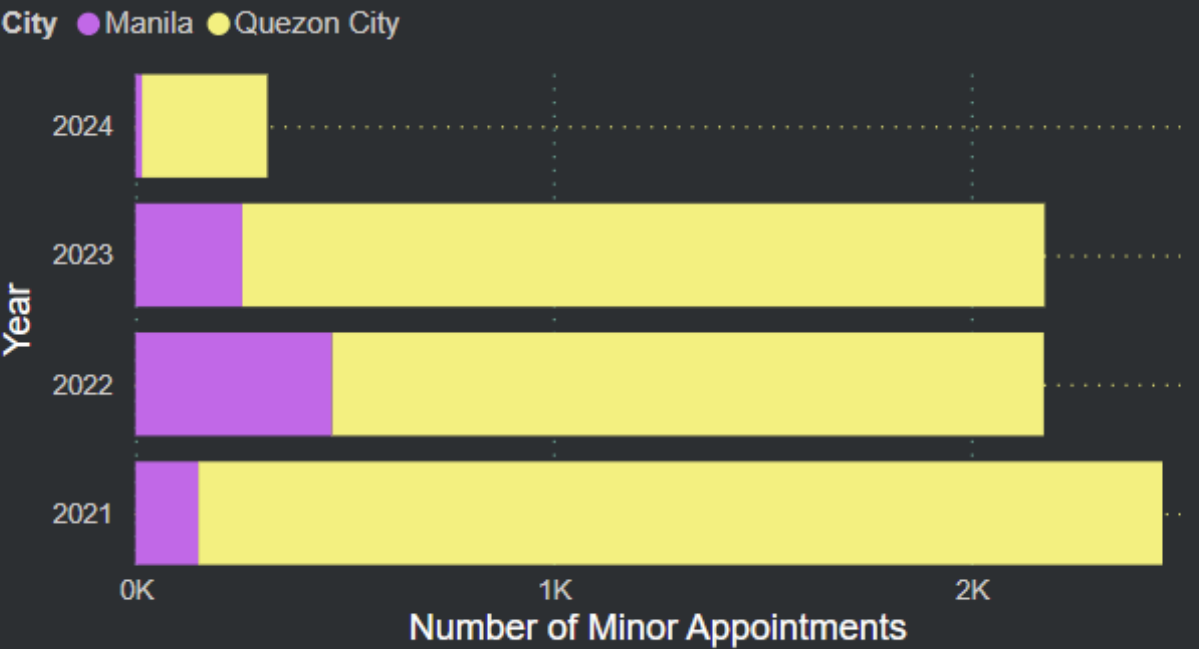
Age Distribution of Patients of Target Gender



Patient-to-Doctor Ratio by City in Target Region



Number of Minor Appointments by Year and City



Optimization Timing Process

- 18 timed executions on each of the 4 OLAP queries, 6 iterations on 3 different levels of optimization
- Firstly timed with no indices and no further optimizations
- Each query is retimed after adding secondary indices.
- Queries then underwent query restructuring (Rollup and Dice) or table denormalization (Drilldown and Slice).
- Retiming is conducted after each restructuring or denormalization step.
- The final phase involved retiming each report with both indexing and specific optimization.



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

This statistic 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.

Rollup

Doctors with Highest Average
Number of Appointments per Patient

```
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;
```

Rollup Optimization

- Utilized query restructuring/optimization by removing the subquery.
- The original approach involved querying the number of appointments per patient with a specific doctor.
- The revised query calculates the average by dividing the total appointments a doctor has by the number of unique patients.

```
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;
```



Query Performance Comparison

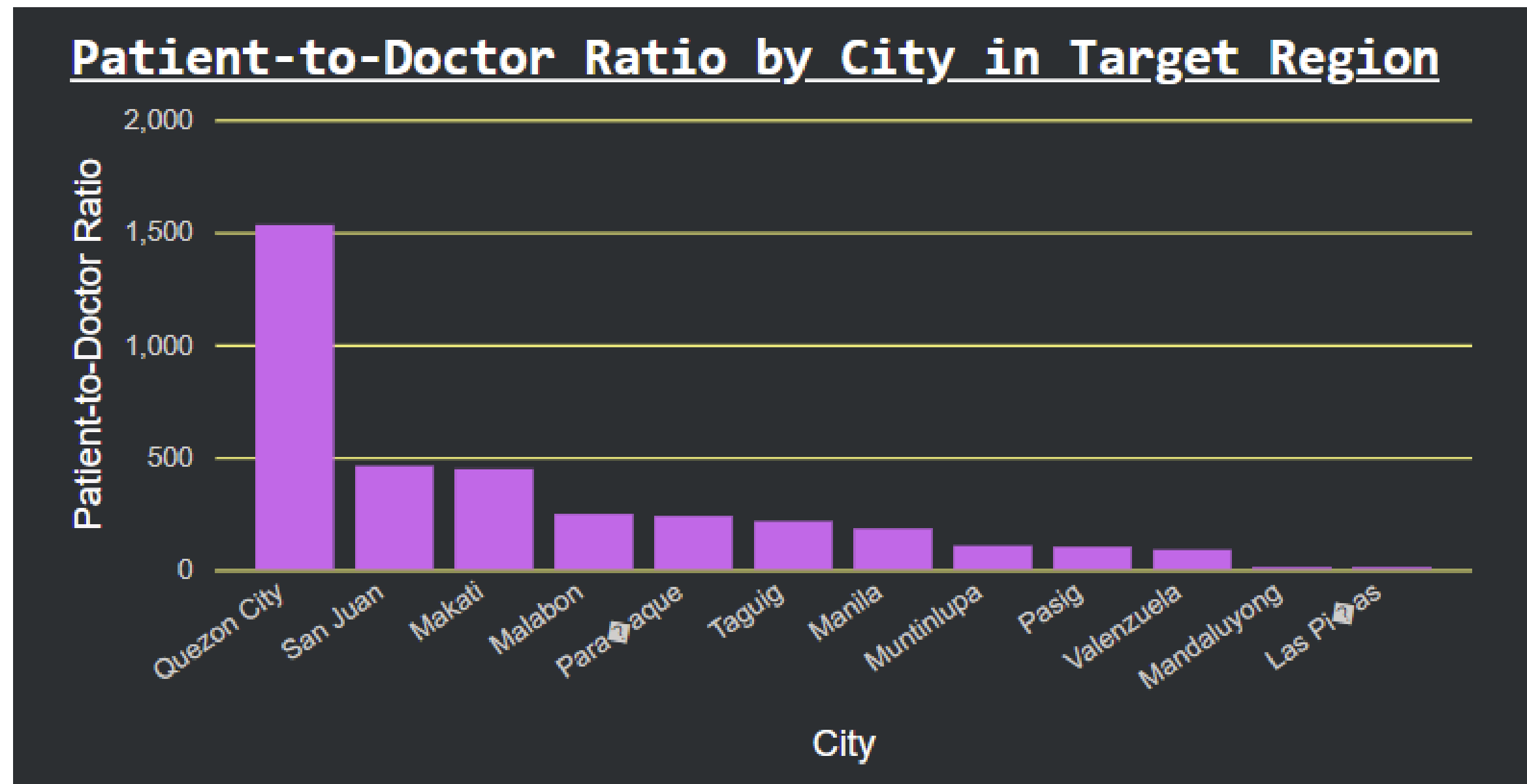
Table 1. Rollup SQL Query Performance Comparison

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%



Drilldown

Patient to Doctor Ratio per City for a Given Region



This statistic is essentially a measure of which areas are undersaturated and oversaturated with medical professionals.

Drilldown

Patient to Doctor Ratio per City for a Given Region

```
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;
```

Drilldown Optimization

- Choose denormalization to precalculate the number of distinct doctors and patients at each clinic.
- Counted distinct patients and doctors within clinics, used in the existing formula.
- Exponentially faster results were obtained.
- NCR region due to its substantial clinic count (36,216).

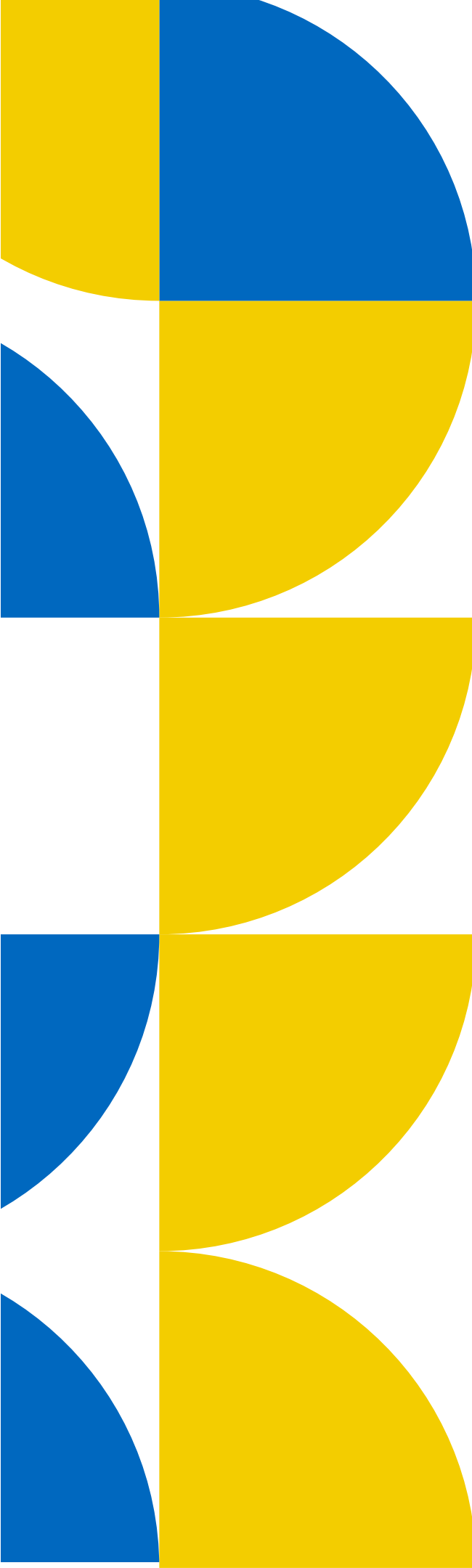
```
SET @targetRegion = 'National Capital Region (NCR)';  
SELECT city, COALESCE(AVG(unique_patients/unique_doctors),0) AS patient_to_doctor_ratio  
FROM clinics  
WHERE region_name = @targetRegion  
GROUP BY city  
ORDER BY patient_to_doctor_ratio DESC;
```



Query Performance Comparison

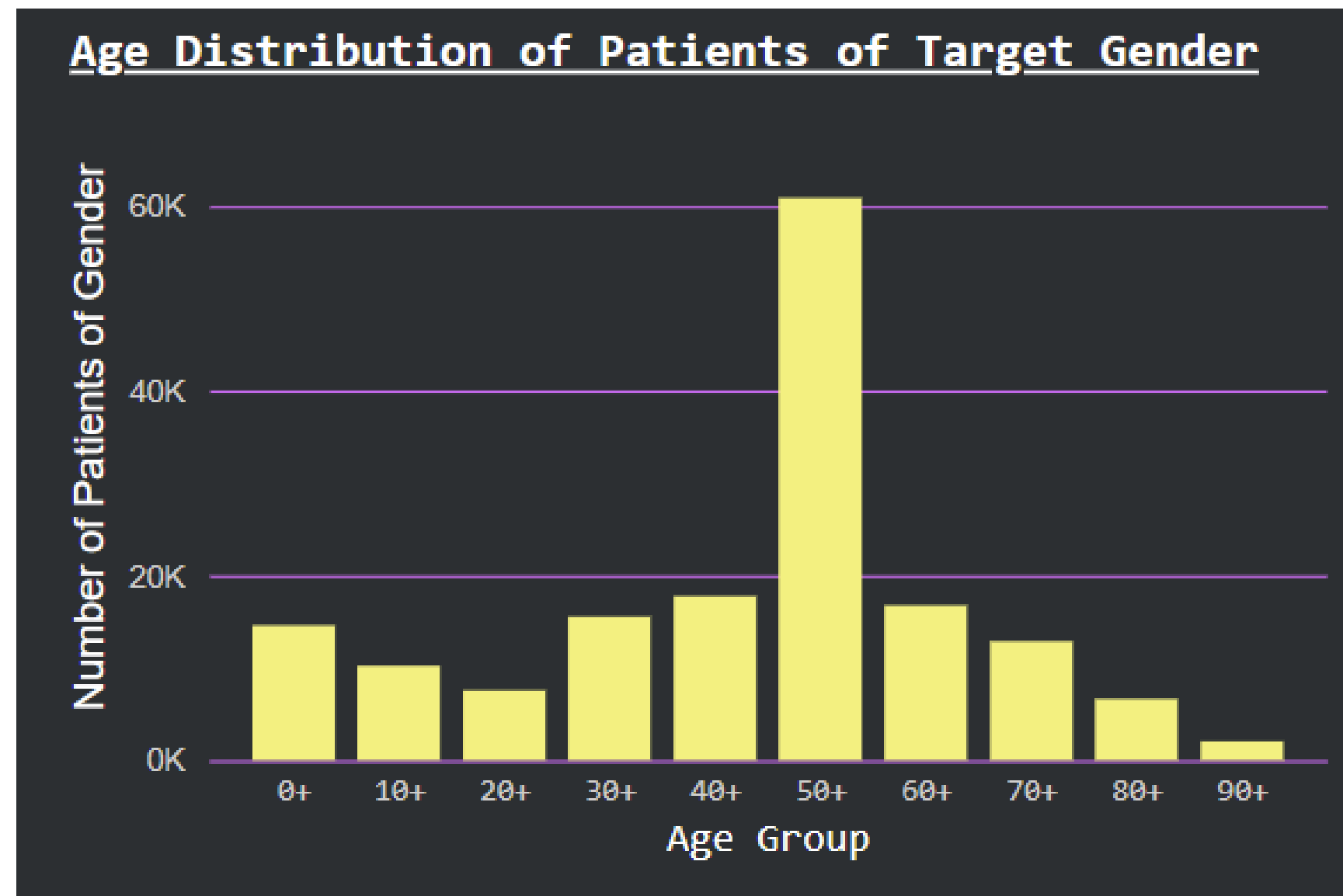
Table 2. Drilldown SQL Query Performance Comparison

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



Slice

Age Distribution of Patients of a Given 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.

Slice

Age Distribution of Patients of a Given Gender

```
SET @targetGender = 'MALE';
```

```
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;
```



Slice Optimization

```
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;
```

- Implemented denormalization to optimize the calculation of age distributions of patients by gender.
 - Redundantly stored patient age and gender within the appointment table Eliminating the need for joins to calculate the report.
 - Exponential gains in execution time were observed
 - Conducted trials for both "MALE" and "FEMALE" queries to account for all possibilities.
- 

Query Performance Comparison

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%

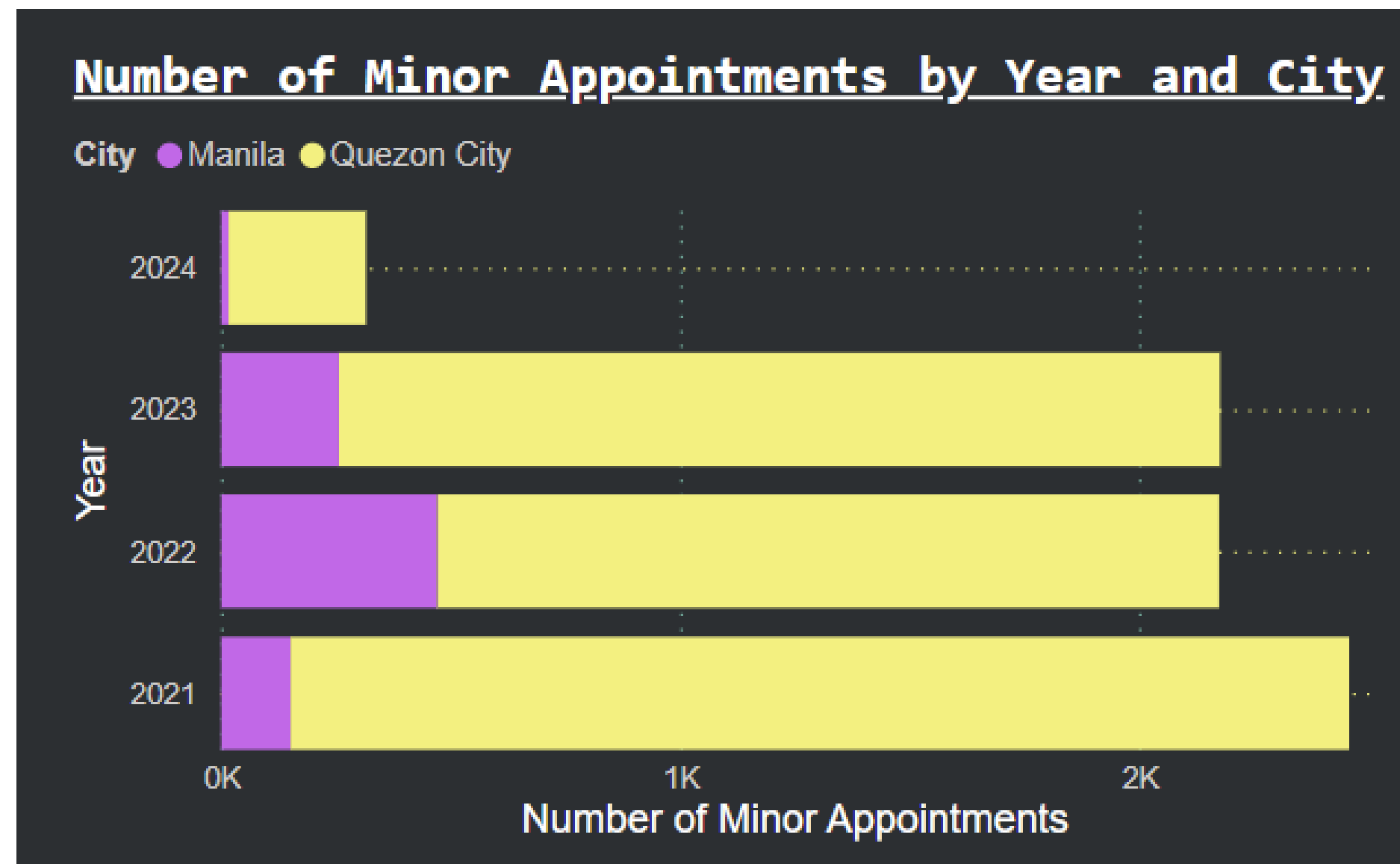
Table 4. Slice SQL Query Performance Comparison - Female

Trial #	1	2	3	AVE	Compare d to Unoptimi zed	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%



Dice

Number of Minor Appts. By City Within Time Span



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.

Dice

Number of Minor Appts. By City Within Time Span

```
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;
```



Dice Optimization

```
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;
```

- Utilized query restructuring, The original query involved a subquery to compile minor appointments and filter by time range before joining clinics for city filtering.
 - The restructured query began at the appointments table, joining patients and clinics directly, with filtering within the main query.
 - Trials were conducted between "Manila" and "Quezon City", the cities with the largest clinic counts (23,566 and 4,603 respectively).
 - Trials divided into two time ranges: 2021-2024 and 2017-2024, to observe performance differences with larger sample sizes.
- 

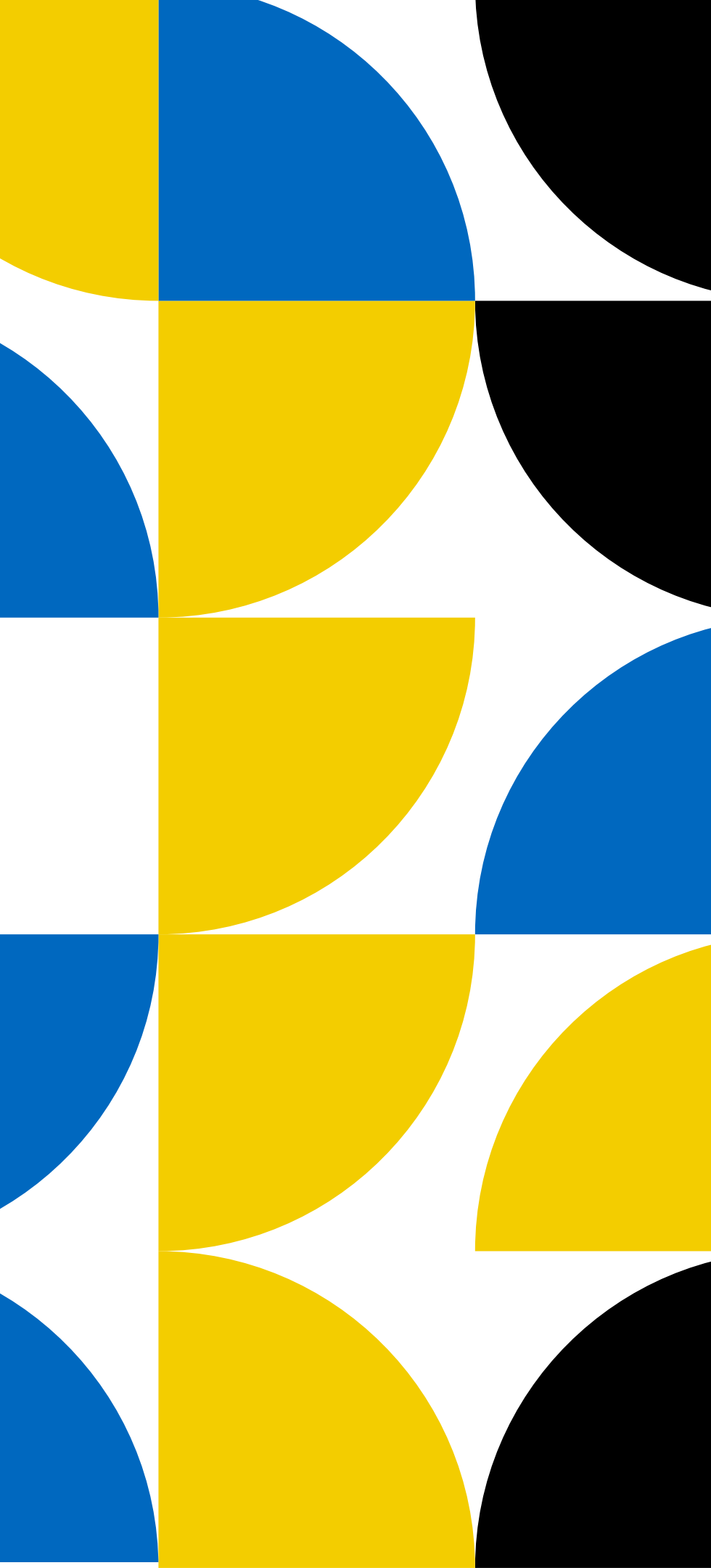
Query Performance Comparison

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%



Function Testing

- Early function testing on small data subset.
- Initial CSV split into smaller samples with Jupyter.
- Group members independently attempted SQL creation.
- Manual verification due to small data size.
- Independent development allowed cross-checking.
- Future optimized queries compared to initial ones.
- Pandas used to prevent foreign key violations.
- Function testing boosted confidence in data and OLAP output.
- Early testing reduced errors for an error-free product.



Analysis of Results

Indexing Optimization:

- Consistently provided significant optimization ranging from 27% to 87%.
- Exceptionally improved query speed except for the slice query, showing a marginal 2% increase.
- Highlighted as a recommended first optimization due to simplicity and major improvements.



Analysis of Results

Table Denormalization:

- Produced exceedingly strong improvement, with queries being orders of magnitude faster (99.7% faster).
- Reports that were previously at risk of not generating within time limits were completed in less than a second.
- Requires increased storage space and is harder to maintain with continuous updates.
- Potential for data desynchronization due to failure to update columns, impacting accuracy [2].



Analysis of Results

Query Restructuring/Optimization:

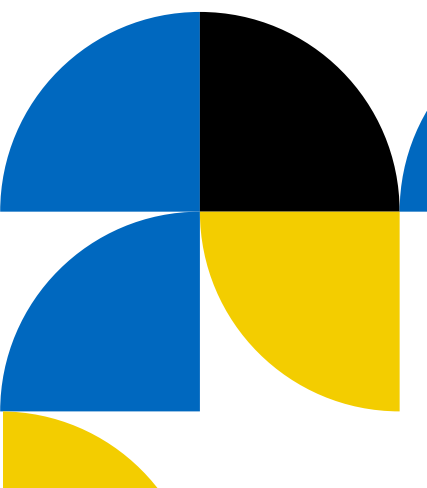
- Most time-consuming method requiring a rigorous understanding of SQL constructs and logic.
- Provided a 15% - 28% increase in comparison to indexed but unoptimized SQL queries.
- Sole method with no side effects, strictly improving query performance.
- Efforts toward efficient SQL queries result in better report generation in all scenarios.





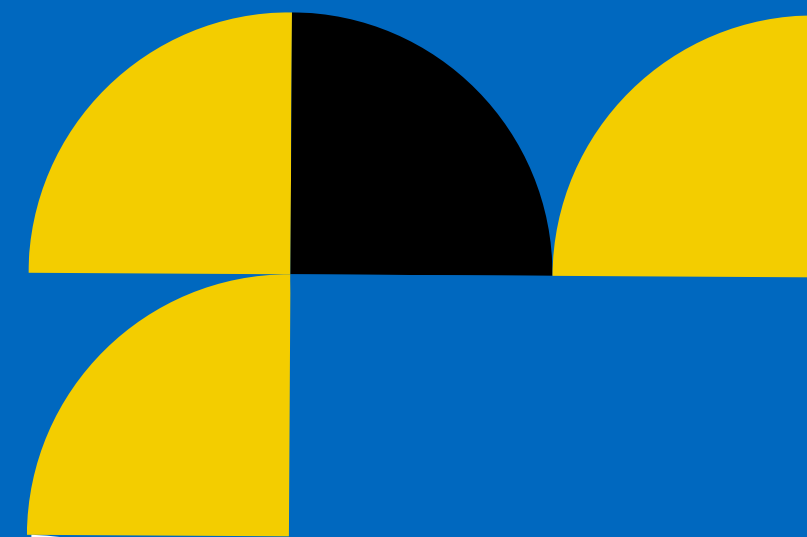
Conclusion

- The Project highlighted the necessity of familiarity with ETL process tools like NiFi for cleaning and loading large quantities of data.
- Real-world data posed challenges, with datasets being messy, large, and difficult to manage.
- Previous tools like Excel became unusable due to dataset size, necessitating adaptation and learning new data analysis methods.
- Realization of the importance of critical thinking beyond query formation and optimization tasks.
- Initial reliance on Apache NiFi for ETL processes was challenged by real-world conditions, leading to adaptation to other tools like Pandas.
- The project underscored the necessity of data optimization techniques such as indexing, query restructuring, and data denormalization for faster queries, even in non-industrial settings.





Thank You





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>
- 