Extracting, Transforming, and Loading SeriousMD with Apache NiFi: OLAP Dashboard Creation in Tableau and Query Optimization

Mari Salvador Lapuz[1], Elaine Riz Martin[2], Gwen Kathleen Roco[3], and Benson Yao[4]
Software Technology Department, College of Computer Studies, De La Salle University, Manila, Philippines
[1]mari_lapuz@dlsu.edu.ph, [2]elaine_martin@dlsu.edu.ph, [3]gwen_roco@dlsu.edu.phl, [4]benson_yao@dlsu.edu.ph

ABSTRACT
The paper presents a comprehensive approach to streamline data processing using the SeriousMD dataset from an online database. Apache NiFi was utilized for Extract, Transform, Load (ETL) operations to organize the data into a data warehouse consisting of a star schema. Furthermore, the group used Tableau as a visual analytics tool to develop an Online Analytical Processing application for creating a dashboard. Through functional and performance testing, we validate the integrity and efficiency of our data processing pipeline and OLAP queries. Results showed that proper formulation of the query greatly reduces the execution time of the query. Moreover, the use of foreign key indexing improves query performance when used along with proper SQL statements.

Keywords
Data Warehouse, ETL, OLAP Application

1. Introduction
During the pandemic, different web-based applications emerge to help maintain proper queueing in healthcare facilities without having to meet physically. One of which is SeriousMD, a platform for keeping and maintaining medical records, online consultations, appointments, and many more in order. It also provides datasets of appointments, patients, doctors, and clinics, supplying a rich repository for healthcare analytics and strategic decision-making.

This paper delves into the process of setting up and organizing the data from SeriousMD, using the Extract, Transform, Load (ETL) method to gather, transform, and load the data into the data warehouse. Additionally, we developed an Online Analytical Processing (OLAP) application using Tableau, a software designed for analyzing data within a data warehouse. Furthermore, this paper investigates various optimization techniques to enhance the speed of data analysis queries.

2. Data Warehouse
A data warehouse is a storage of large amounts of valuable information from different sources [1]. This valuable information is often subject to data analytics, usually to improve the processes and decision-making of businesses or individuals. The upcoming section details the source dataset and the corresponding schema designed for the data warehouse.

2.1 Source Dataset
The data warehouse is composed of data from the `SeriousMD` dataset. The dataset consists of four tables: `appointments`, `px`, `doctors`, and `clinics`. Descriptions of these tables are provided in Table 1.

Table 1. SeriousMD dataset

Table Name	Description
appointments	appointments made in the dataset
px	list of patients in the dataset
doctors	list of doctors in the dataset
clinics	list of clinics in the dataset

2.2 Dimensional Model
In accordance with project specifications, the decision between a star schema and a snowflake schema for the data warehouse was examined. Considering the inherent table structure of the original dataset, a choice was made to implement a star schema, primarily for its superior query efficiency [6]. This decision aligns well with the paper's primary goal of query optimization. Additionally, it was observed that the tables in `SeriousMD` were inherently denormalized, eliminating the need for extensive redesigning. Only minor renaming to some fields was done to improve readability as shown in Table 2.

Table 2. Renamed fields in the SeriousMD dataset

Table Name	Original Field Name	New Field Name
appointments	pxid	px_id
	clinicid	clinic_id
	doctorid	doctor_id
	apptid	appt_id
	TimeQueued	time_queued
	QueueDate	queue_date
	StartTime	start_time
	EndTime	end_time

	Patient type	patient_type
	Virtual	is_virtual
px	pxid	px_id
doctors	doctorid	doctor_id
	mainspecialty	main_speciality
clinics	clinicid	clinic_id
	hospitalname	hospital_name
	IsHospital	is_hospital
	RegionName	region

Figure 1 illustrates the Entity Relationship Diagram of the data warehouse, consisting of `appointments` as the fact table, and `px`, `clinics` and `doctors` as the three dimension tables. There is a hierarchical relationship observed wherein doctors, patients and clinics may have many appointments. In contrast, a single appointment may only have one patient, doctor and clinics.

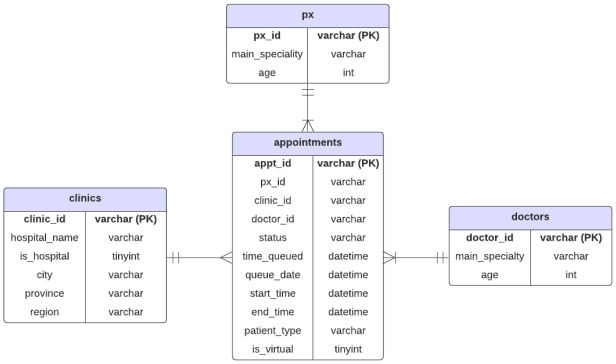


Figure 1. ERD of the SeriousMD data warehouse

The fact table, `appointments`, is identified by the primary key `appt_id`. The dimension tables, `px`, `doctors` and `clinics`, are identified by the primary keys, `px_id`, `doctor_id` and `clinic_id` respectively.

`appointments` was chosen as the fact table as it is the center of the dataset, meaning all dimension tables are directly related to it. The details of each appointment contain details from other tables. Hency, it is the best choice for the fact table role, saving the need to create a completely new table.

3. ETL Script

After establishing the data warehouse schema, data was then populated into it utilizing an ETL script.

ETL is a pipeline that extracts data from one or multiple sources, applies transformations, and then loads the result into a designated location like a data warehouse. [3] Initially, raw data from its source location is duplicated or extracted to an area called the staging location. Afterward, it undergoes various processes such as data cleaning, performing calculations, and others to authenticate the data. Finally, it is loaded from the staging location into a data warehouse. [7]

3.1 ETL Process Overview

Due to the large size of the uncleaned data source, it was initially duplicated to a Jupyter Notebook for the transformation process. The group cleaned the data and reduced the size of the data source accordingly to facilitate easier handling and analysis. Following this, it was loaded into Datagrip, another tool for Database Management Systems (DBMS), and linked to MySQL, which now serves as the referenced data source for the group. This ETL process was undertaken to streamline the ETL pipeline tailored to extract, transform, and load a data source from MySQL to its designated data warehouse using Apache Nifi.

Additionally, it is notable that the group performed the ETL process twice; however, we will only focus further on discussing the second ETL process in sections 3.2 to 3.3.

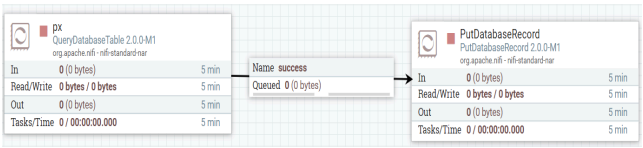


Figure 2. Overview of ETL Pipeline

3.2 Discussion of ETL Pipeline

Figure 2 illustrates the ETL pipeline utilized by the group. First, the QueryDatabaseTable processor extracted data from various tables such as appointments, clinics, doctors, and px from the database source. This processor retrieved data based on specified source database tables. The extracted data was then transformed into an Avro-encoded format and processed by the PutDatabaseRecord processor, which reads the Avro-encoded data before loading it into the target database table.

Before proceeding to fully load the records into the data warehouse, a problem was encountered with the Avro Schema, particularly regarding the naming of header names in the tables. The group followed the naming conventions of the database design created, which differed from the header names of the database source. To address this issue, all header names in both the data warehouse table and the data source table were renamed to comply with Avro's naming scheme restrictions. Following these adjustments, the issue was resolved.

In the context of Figure 2, the data will be stored in the "`dbwarehouse-mco1`" schema, specifically in the `px` table, using INSERT statements. Applying this process to all tables, as outlined in Appendix E, facilitates the loading of records into the data warehouse.

3.3 ETL Automation

The ETL pipeline was automated so it ran every 24 hours. To ensure regular execution, the scheduling strategy of the QueryDatabaseTable and PutDatabaseRecord processors are adjusted to "timer-driven," with a run schedule set to occur every 24 hours or 86,400 seconds.

4. OLAP Application

OLAP is a software that allows the execution of multidimensional analysis on a large number of datasets sourced from a data

warehouse [8]. In analyzing the SeriousMD dataset, the group designed an OLAP application suitable for visualizing the results of the queries better. To be specific, we utilized Tableau as our visual analytics tool [9].

All queries utilized in the OLAP application focus on appointments and patient metrics within a healthcare system. With this, the queries are made for healthcare administrators, clinic managers, medical directors, and healthcare analysts. This can give them insights regarding resource allocation, staffing, scheduling optimizations, and service improvements within the healthcare system.

4.1 Average appointment time in hours of all appointments held in non-hospital clinics per month

Figure 3. Tableau Dashboard Query for the Average Appointment Time in Hours of All Appointments Held in Non-hospital Clinics Per Month

Figure 3 exhibits the average duration of appointments monthly at non-hospital clinics. The query first utilizes the slice operations for the clinics dimension using the criterion where a clinic is a non-hospital clinic. We then group appointments by month and calculate the average duration for each month. Lastly, a rollup operation was implemented to aggregate the overall average appointment duration across all months.

4.2 Overall number of patients monthly

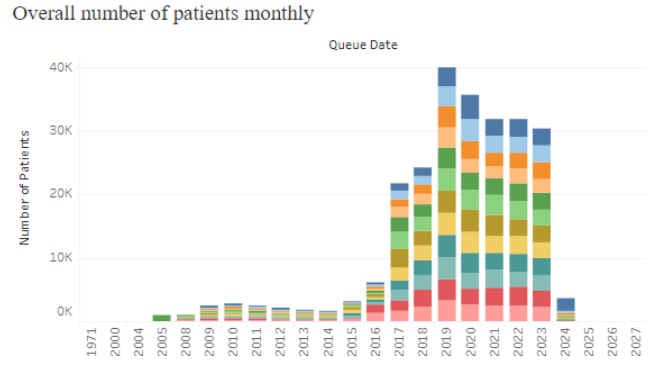


Figure 4. Tableau Dashboard Query for the Overall Number of Patients Monthly

Figure 4 shows the overall number of patients each year. As shown in Figure 4, it can be noted that the patients are distributed on a monthly basis as well.

To attain this, our query first organizes appointments by year and month, then the distinct number of patients seen in each month are counted. This allows us to track the monthly patient flow over time. By utilizing drill-down operations, we transition from aggregating data by years to breaking it down further into months. This detailed view enables us to identify any seasonal trends or fluctuations in patient numbers throughout the year.

4.3 Appointments made in Manila with doctors specialized in Pediatrics on a monthly basis

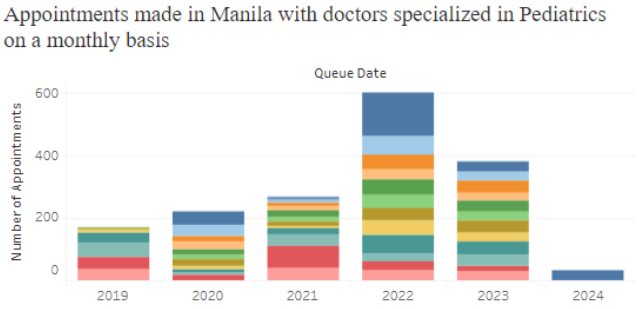


Figure 5. Tableau Dashboard Query for the Average Appointment Time in Hours of All Appointments Held in Non-Hospital Clinics Per Month

This query shows the number of all appointments made in Manila under Pediatrics doctor on a monthly basis. As seen in Figure 5, the generated report shows the total number of appointments per month in every year, which is only under the doctors who specialize in pediatrics.

To achieve this, the application first employed a dice operation to narrow down the data, specifically to appointments in clinics within Manila whose doctors specialized in pediatrics. Appointments, clinics, and doctors' tables were combined, filtering appointments where the province is “Manila” and doctors specializing in “Pediatrics”. Then, the remaining data were grouped by year and month to track the monthly appointment trends.

5. Query Processing and Optimization

With thousands of records in the database, executing queries involving JOINs might lead to sluggish performance. Thus, there is a necessity for query optimization through efficient database-level optimization and proper query formulation.

5.1 Database Level Optimization

A foreign key (FK) is a column or a set of columns utilized to create and enforce a connection between data in two tables, regulating the type of data permitted in the table containing the foreign key. [5] In MYSQL Workbench, to create a foreign key, an index on the referencing columns of the foreign key should be created. This index is often referred to as a "foreign key index.” Creating an index on foreign key columns in the child table

improves query performance by accelerating join operations with the parent table. This indexing practice is particularly beneficial in speeding up data retrieval and minimizing computational costs associated with maintaining relational integrity [4].

To improve the original database design, foreign keys were added to link the fact table with three dimension tables. Figure 6 shows these connections, where `appointments`' `px_id`, `clinic_id`, and `doctor_id` refer to `px`'s `px_id`, `clinics`' `clinic_id`, and `doctors`' `doctor_id`, respectively. Executing this in MySQL Workbench created foreign key indices: `fk_appointments_px_idx`, `fk_appointments_doctors_idx`, and `fk_appointments_clinics_idx`.

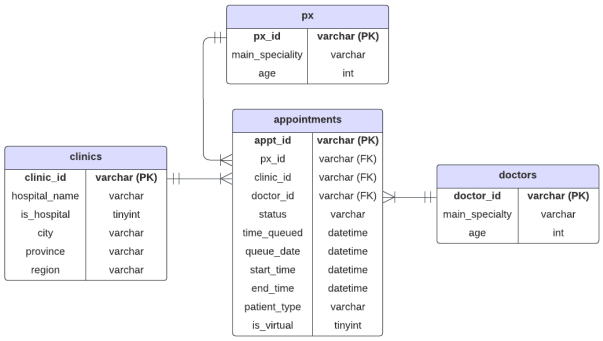


Figure 6. ERD of Serious MD data warehouse with foreign keys and proper indexing

5.2 Query Formulation

The next optimization step is to use efficient SQL querying techniques, refining and organizing queries to make the database work better overall. This includes removing unnecessary JOINs and poorly structured subqueries.

5.2.1 Average appointment time in hours of all appointments held in non-hospital clinics per month

For this query, the unoptimized SQL script uses subquery which significantly decreases the performance of generating the output of the query. Furthermore, there are joins and conditions within the subquery. On the other hand, the optimized version is much faster due to the simple construction of the SQL query.

Listing 1. Unoptimized SQL Script for Query 5.2.1

```
SELECT      MONTH(a.queue_date) AS 'month',
              ROUND(AVG(ABS(TIMESTAMPDIFF(S
              ECOND, a.start_time,
              a.end_time)/3600)), 2) AS
              'appointment_duration'
FROM        appointments a
WHERE       a.start_time IS NOT NULL
AND         a.end_time IS NOT NULL
AND         a.clinic_id IN
        (
          SELECT a2.clinic_id
          FROM   appointments a2, clinics c
          WHERE  c.is_hospital = 'False'
          AND    a2.clinic_id = c.clinic_id

        )
GROUP BY    'appointment_duration',
            MONTH(a.queue_date)
ORDER BY    MONTH(a.queue_date);
```

Listing 2. Optimized SQL Script for Query 5.2.1

```
SELECT      MONTH(a.queue_date) AS 'month',
              ROUND(AVG(ABS(TIMESTAMPDIFF(S
              ECOND, a.start_time,
              a.end_time)/3600)), 2) AS
              'appointment_duration'
FROM        appointments a
JOIN        clinics c  ON
            a.clinic_id = c.clinic_id
WHERE       c.is_hospital = 'False'
AND         a.start_time IS NOT NULL
AND         a.end_time IS NOT NULL
GROUP BY    'appointment_duration',
            MONTH(a.queue_date)
ORDER BY    MONTH(a.queue_date);
```

5.2.2 Overall number of patients monthly

The query drills down from the overall yearly view to a detailed monthly breakdown per year by using the `GROUP BY` clause for both the year and month. In this query formulation, only table `appointments` is needed. However, The unoptimized SQL for this query, as depicted in Listing 3, uses a left join which retrieves more records than what was needed. This contributes to a slower query execution time. Thus, the optimal SQL query, depicted in Listing 4, would be to remove unnecessary `JOIN`s as other tables' records are not needed for the requirement.

Listing 3. Unoptimized SQL Script for Query 5.2.2

```
SELECT      YEAR(queue_date) AS year,
            MONTH(queue_date) AS month,
            COUNT(DISTINCT a.px_id) AS
            num_patients
FROM        appointments a
LEFT JOIN   clinics c ON a.clinic_id =
            c.clinic_id
LEFT JOIN   doctors d ON a.doctor_id =
            d.doctor_id
LEFT JOIN   px p ON a.px_id = p.px_id
GROUP BY    year, month
ORDER BY    year, month;
```

Listing 4. Optimized SQL Script for Query 5.2.2

```
SELECT    YEAR(queue_date) AS year,
          MONTH(queue_date) AS month,
          COUNT(DISTINCT px_id) AS
          num_patients
FROM      appointments
```

```
GROUP BY  year, month
ORDER BY  year, month;
```

5.2.3 Appointments made in Manila with doctors specialized in Pediatrics on a monthly basis

The output requires the use of two different tables which are the `appointments`, `clinics` and `doctors`. The query slices the data by focusing on appointments in Manila ('`province`' dimension) and filters further by selecting appointments with doctors specialized in Pediatrics ('`specialty`' dimension). The aggregation of appointment counts based on the '`year`' and '`month`' dimensions further reflects a multidimensional analysis of the data. The unoptimized SQL script uses the necessary number of tables, but it uses subquery which can greatly increase SQL execution time whereas the optimized SQL script achieved the same output without the need of subquery.

Listing 5. Unoptimized SQL Script for Query 5.2.3

```
SELECT     YEAR(queue_date) AS year,
           MONTH(queue_date) AS month,
           COUNT(appt_id) AS
           num_appointments
FROM       appointments AS a
INNER JOIN clinics AS c ON a.clinic_id =
           c.clinic_id
INNER JOIN doctors AS d ON a.doctor_id =
           d.doctor_id
WHERE      c.province = 'Manila'
AND EXISTS (
        SELECT 1 FROM doctors
        WHERE main_specialty =
              'Pediatrics' AND doctor_id =
              a.doctor_id
            )
GROUP BY   year, month
ORDER BY   year, month;
```

Listing 6. Optimized SQL Script for Query 5.2.3

```
SELECT YEAR(queue_date) AS year,
       MONTH(queue_date) AS month,
       COUNT(appt_id) AS num_appointments
FROM   appointments AS a, clinics AS c,
       doctors AS d
WHERE  a.clinic_id = c.clinic_id
       AND c.province = 'Manila'
       AND a.doctor_id = d.doctor_id
       AND d.main_specialty = 'Pediatrics'
GROUP BY year, month
ORDER BY year, month;
```

6. Results and Analysis

This section comprises the results of functional and performance testing for the ETL Pipelines, OLAP queries as well as the dashboard.

6.1 Functional Testing

This section details the validation process to ensure the data behaves as expected and serves its intended purpose, ensuring its reliability and usability.

6.1.1 ETL Script

To authenticate the validity of data from the ETL pipeline utilized, the data was compared to the source database. The first test compares the rows returned for each table. If the tables returned the same number of rows, this suggests that the loaded data in the data warehouse is complete. The scripts used to see the total resulting number of rows can be found in Listing #. For the `appointments` table in both the data source and data warehouse, both queries returned 320,140 rows. For the `clinics` table, both queries returned 25,232 rows. Next, the `doctors` table returned 10,832 rows for both queries. Lastly, `px` table returned 67,194 rows for both queries.

Listing 7. SQL Script for Returning All Table Rows

```
SELECT * FROM INFORMATION_SCHEMA.TABLE;
```

As we are done testing the similarity for the number of rows returned, the contents for each query statement were exported into separate CSV files. The group utilized an external web application to compare the respective CSV files [2]. The interface of the web application has the feature of filtering rows with different resulting rows for each field in the CSV files. After comparing all the tables from both the data source and the data warehouse, it was found that there are no differences for each file. This continuously showed that the data utilized by the group were correct.

6.1.2 OLAP Queries

For OLAP Queries, functional testing is done by executing SQL queries in MySQL and the OLAP applications. The test cases consist of expected results, representing the SQL Query, and actual results, representing the OLAP application.

Table 3. Test Results for OLAP Query 5.2.1

Test Cases	Expected	Actual	Pass/Fail
Average number of appointment duration in hours for February	0.39	0.39	Pass
Average number of appointment duration in hours for October	1.03	1.03	Pass
Overall appointment duration in hours	0.51	0.51	Pass

Table 4. Test Results for OLAP Query 5.2.2

Test Cases	Expected	Actual	Pass/Fail
Number of Patients for December, in the year 2008	28	28	Pass
Number of Patients for April, in the year 2011	183	183	Pass
Number of Patients for September, in the year 2023	2588	2588	Pass

Table 5. Test Results for OLAP Query 5.2.1

Test Cases	Expected	Actual	Pass/Fail
Number of Appointments made in Manila for October, in the year 2019	45	45	Pass
Number of Appointments made in Manila for September, in the year 2022	38	38	Pass
Number of Appointments made in Manila for April, in the year 2023	23	23	Pass

Tables 3, 4 and 5 illustrate the test results of comparing expected output of using MYSQLl and the actual results from the OLAP application. All three tables show that it passes all test cases. Table 1 has additional tests which make use of the built-in OLAP operation in mysql workbench like the rollup operation which further validates the result of the OLAP application. As for the other two tables, the test cases revolve around the different months of specific years. This allowed us to easily test different scenarios by modifying the conditions through the `WHERE` clause. The test was conducted by everyone in the group to show that there are no inconsistencies in the result of the test cases.

6.2 Performance Testing

After the data accuracy was confirmed through functional testing, the performance of MYSQL OLAP queries were assessed. The queries were executed on MYSQL Workbench on a laptop with the following specifications:

- Processor: Intel Core i7-10750H Processor 2.6 GHz
- RAM: 16GB

To assess query performance, the time taken (in seconds) for each query to produce results was measured and recorded using MySQL Workbench. Each query underwent five trials, and the average duration was calculated. Detailed query durations before averaging are presented in Appendix A. The summarized results in Table 6 indicate query numbers and corresponding average execution times, comparing the unoptimized and optimized SQL scripts on schemas with and without foreign keys.

Table 6. Average Duration of Query Test Results (in seconds)

	Unoptimized SQL Script		Optimized SQL Script	
Query	Schema without Foreign Key	Schema with Foreign Keys	Schema without Foreign Key	Schema with Foreign Keys
1	0.425	2.450	0.259	0.253
2	1.766	1.672	0.457	0.447
3	0.841	0.878	0.519	0.515

In Table 6, each query corresponds to a specific query in Section 5.2. Query 1, 2 and 3, refer to Sections 5.2.1, 5.2.2 and 5.2.3, respectively.

The optimized SQL queries exhibited the shortest average query test durations when applied to the schema with FK. However, with the unoptimized SQL queries, inconsistent results were observed between scenarios with and without FK. The query performance varied, with instances of faster execution with FK and other cases where it was faster without FK. This emphasizes the significance of SQL formulation. Well-structured SQL queries, with effective joins and conditions, along with proper utilization of foreign keys, consistently result in superior performance in the schema with FK, as confirmed by the overall test results. Additionally, it is crucial to ensure that indices representing these foreign keys are in place, as they play a vital role in speeding up data retrieval by optimizing the search process based on the linked keys.

6.3 Dashboard Testing

The application, created using Tableau Public (as detailed in Appendix B), was able to load in less than 5 seconds. To validate the results presented in the Tableau Application, the group created reports using Microsoft Excel. This software is also used as a data visualization tool commonly utilized by students, making it an essential tool for validation. The CSV files of the tables in the database source schema were imported into Microsoft Excel.

Upon generating the reports for the four queries in Microsoft Excel (as outlined in Appendix C), it was found to produce the same results, which the group cross-checked.With this, the outputs from both Microsoft Excel and the Tableau Application proved that the application used was correct..

7. Conclusion

In this project, Apache Nifi played a crucial role in extracting, transforming, and loading the data from the source database to the local data warehouse built with MySQL workbench. The data was analyzed, and OLAP techniques were applied to refine and optimize the queries. Subsequently, Tableau was utilized to create interactive data visualizations based on the different OLAP queries.

Optimizing queries included improving the database at the structural level by implementing foreign key indexing and crafting SQL scripts for efficiency. The performance testing indicated that SQL queries designed with precision, coupled with effective use

of foreign keys, consistently led to enhanced performance within the schema featuring foreign keys.

In conclusion, the project showcased the successful integration of ETL processes, OLAP techniques, and visualization tools to enhance data analysis and decision-making capabilities in the healthcare domain.

8. References

[1] Amazon Web Services, Inc. 2023. Data Warehouse Concepts. Retrieved from https://aws.amazon.com/datawarehouse/

[2] Extends Class. CSV diff - Online compare tool. Retrieved from https://extendsclass.com/csv-diff.html

[3] Data Pipelines & ETL | Apache Flink. Retrieved from https://nightlies.apache.org/flink/flink-docs-master/docs/learn-flink/etl/

[4] How to index foreign key columns in SQL server: 2019. https://www.sqlshack.com/index-foreign-key-columns-sql-server/. Accessed: 2024-02-20.

[5] Primary and foreign key constraints - SQL server: https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver16. Accessed: 2024-02-20.

[6] Star schema vs snowflake schema: 6 key differences: 2023. https://www.thoughtspot.com/data-trends/data-modeling/star-schema-vs-snowflake-schema. Accessed: 2024-02-20.

[7] What is ETL (extract, transform, load)? https://www.ibm.com/topics/etl. Accessed: 2024-02-20.

[8] What is OLAP? https://www.ibm.com/topics/olap. Accessed: 2024-02-20.

[9] What is tableau? https://www.tableau.com/why-tableau/what-is-tableau. Accessed: 2024-02-20.

Appendix

A. Query Test Results per Trial

Table 7. Query Test Results (in seconds) with Unoptimized SQL Script on the Original Schema

Query	T1	T2	T3	T4	T5
1	0.422	0.437	0.422	0.422	0.422
2	1.734	1.750	1.735	1.797	1.813
3	0.859	0.844	0.828	0.844	0.828

Table 8. Query Test Results (in seconds) with Unoptimized SQL Script on Schema with Foreign Keys

Query	T1	T2	T3	T4	T5
1	2.438	2.469	2.469	2.422	2.453
2	1.671	1.688	1.672	1.657	1.671
3	0.813	0.937	0.844	0.875	0.922

Table 9. Query Test Results (in seconds) with Optimized SQL Script on the Original Schema

Query	T1	T2	T3	T4	T5
1	0.265	0.250	0.266	0.250	0.266
2	0.469	0.438	0.469	0.454	0.453
3	0.516	0.516	0.531	0.516	0.516

Table 10. Query Test Results (in seconds) with Optimized SQL Script on Schema with Foreign Keys

Query	T1	T2	T3	T4	T5
1	0.250	0.265	0.250	0.250	0.250
2	0.453	0.453	0.438	0.437	0.453
3	0.515	0.515	0.516	0.516	0.515

B. Tableau Dashboard Link

OLAP Application Dashboard Link

https://public.tableau.com/app/profile/elaine.martin6651/viz/Book1_17084105101160/Dashboard1

C. Dashboard Testing

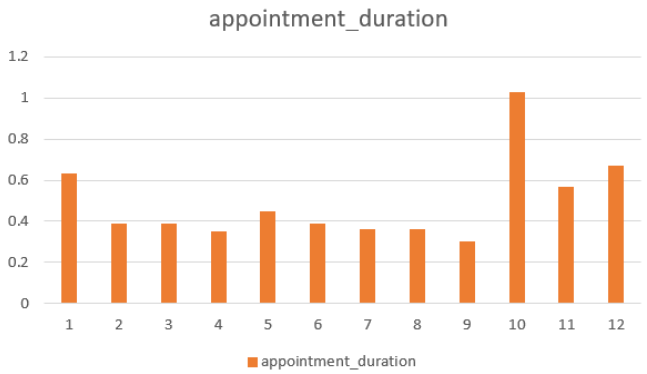


Figure 7. Generated Report for Query 1 from Microsoft Excel

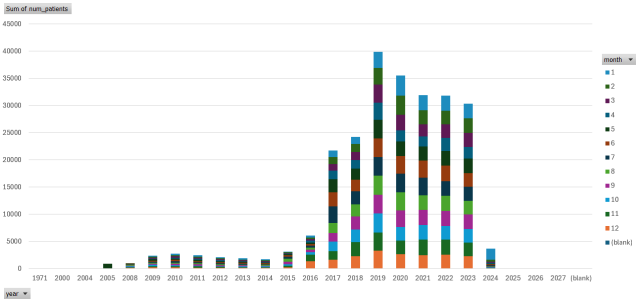


Figure 8. Graph of Query 1 from Tableau Dashboard

Figure 9. Query 2's graph from Tableau Dashboard

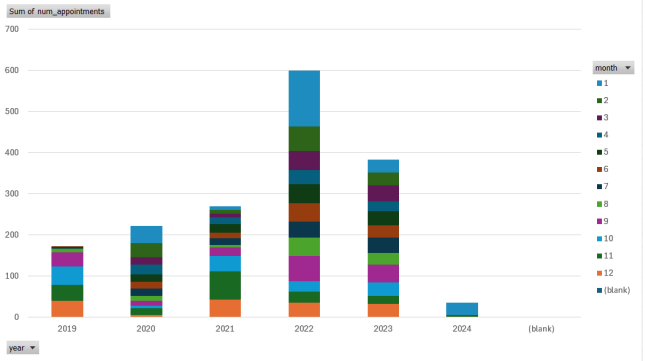


Figure 10. Generated Report for Query 2 from Microsoft Excel

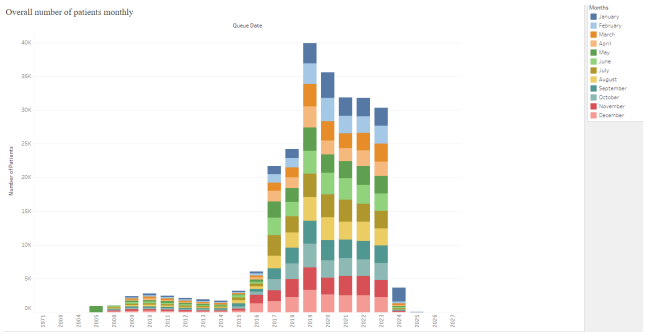


Figure 11. Query 3's graph from Tableau Dashboard

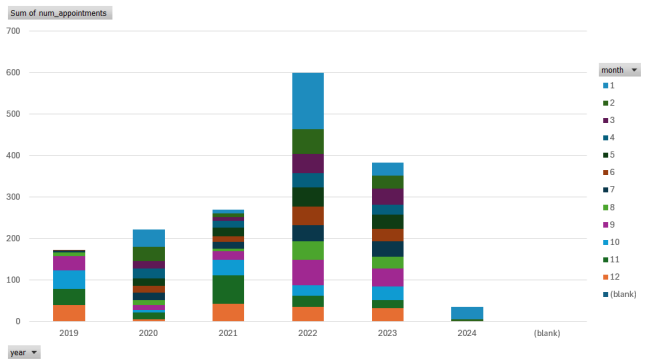
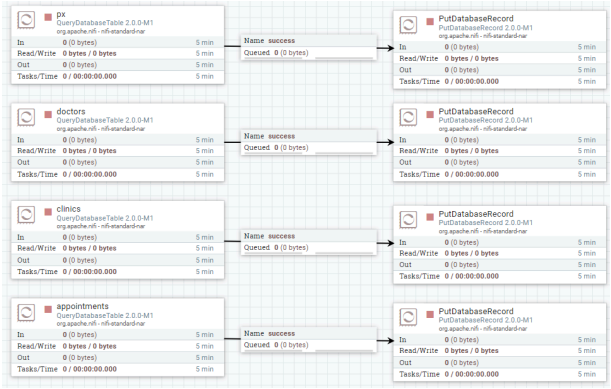


Figure 12. Generated Report of Query 3 from Microsoft Excel

D. Overall ETL Pipeline



E. Resulting Tables of the Optimized Queries

month	appointment_duration
1	0.63
2	0.39
3	0.39
4	0.35
5	0.45
6	0.39
7	0.36
8	0.36
9	0.30
10	1.03
11	0.57
12	0.67

Figure 13. Resulting Table of the Optimized Query 1

year	month	num_patients
1971	12	1
2000	7	1
2004	5	8
2004	6	1
2004	9	1
2004	11	1
2005	5	915
2008	5	28
2008	6	110
2008	7	144
2008	8	143
2008	9	114
2008	10	134
2008	11	151
2008	12	141
2009	1	204
2009	2	157
2009	3	176
2009	4	164
2009	5	204
2009	6	216
2009	7	212
2009	8	202
2009	9	177
2009	10	212
2009	11	251
2009	12	228
2010	1	251
2010	2	235
2010	3	256
2010	4	223
2010	5	211
2010	6	243
2010	7	243
2010	8	225
2010	9	245

Figure 14. Resulting Table of the Optimized Query 2

year	month	num_appointments
2019	6	1
2019	7	5
2019	8	9
2019	9	34
2019	10	45
2019	11	38
2019	12	40
2020	1	42
2020	2	35
2020	3	18
2020	4	23
2020	5	18
2020	6	16
2020	7	19
2020	8	12
2020	9	12
2020	10	6
2020	11	16
2020	12	5
2021	1	9
2021	2	10
2021	3	9
2021	4	15
2021	5	22
2021	6	13
2021	7	16
2021	8	7
2021	9	20
2021	10	37
2021	11	69
2021	12	43
2022	1	136
2022	2	59
2022	3	46
2022	4	34
2022	5	47

Figure 15. Resulting Table of the Optimized Query 3