BACK KHOA UNIVERSITY - HCMC NATIONAL UNIVERSITIES

FACULTY OF COMPUTER SCIENCE AND ENGINEERING
COURSE: BIG DATA (CO5135)

Final Report

Mass Healthcare analytics with Fast distributed SQL query engine (Trino) and Big data Business Intelligence (Apache Superset)

Instructor:	Thoai Nam	
Group BD01:	Nguyen Binh	- 2270754
	Nguyen Minh Anh	- 2370644
	Tran Nguyen Gia Hung	- 2370182

Ho Chi Minh City, December 26th 2025

Contents

1 Introduction

1.1 Context

In the era of data-driven healthcare, the ability to process and analyze large volumes of medical data efficiently is crucial to healthcare analytics. Modern healthcare systems generate vast amounts of data across networks of hospitals and clinics. Analytic institutes like healthcare consultants, research laboratories, or national departments need to access, analyze, and deliver insights from this vast distributed data. Traditional query methods often struggle to handle the growing scale and complexity of healthcare datasets, they require structured data-warehouse that is created by pre-processing data-lake which can be time-consuming and resource intensive.

To maintain analytic performance and keep up with healthcare industry, there is a growing need to query and analyze distributed data efficiently using fast distributed SQL query engines like Trino. In this case, fast distributed SQL query engines like Trino and big data oriented BI platform like Apeche Superset will help to extract and visualize insights of:

- Patient demographics
- Disease trends
- Treatment cost trends
- Impacts of policies

1.2 Mass Healthcare Analytics problem

Gaining insights into patient demographics, disease trends, treatment cost patterns, and the impacts of policies is one of the crucial aspects in mass healthcare analytics, as these factors collectively shape population-level health outcomes and system performance. Patient demographic analysis enables the identification of vulnerable groups and health disparities, while disease trend analysis supports early detection, prevention strategies, and resource planning. Understanding treatment cost trends is essential for evaluating economic sustainability, optimizing resource allocation, and improving cost-effectiveness of care. Additionally, analyzing the effects of policies allows stakeholders to assess policy effectiveness and guide evidence-based decision-making. Together, these insights enable mass healthcare analytics to transform large-scale health data into actionable intelligence that improves quality of care, operational efficiency, and public health planning.

1.3 Objectives

This mini-project explores the use of Trino, a high-performance distributed SQL query engine, to perform efficient and scalable direct query on distributed healthcare data, and Apache Superset, a big data oriented business intelligence (BI) platform, to visualize, analyze and explore massive queried data.

Trino enables fast querying across diverse data sources without the need for complex data movement or transformation, and Apache Superset eases BI operations through seamless query integration and support with Trino. Using Trino and Apache Superset, this mini-project aims to demonstrate how modern distributed query engines and BI platform can improve healthcare analytics by providing faster insights, supporting interoperability, and enabling more flexible data exploration.

1.4 Team assignments

Here is our team's internal assignment and projected contribution of each member described in Table 1 below:

Names	**Contribution**	**Work Scope**
Nguyen Binh	33.3%	Project & Document management. BI implementer
Nguyen Minh Anh	33.3%	Datasources & dataset development
Tran Nguyen Gia Hung	33.3%	Query engine development

Table 1: Member & Task assignments

2 Technologies

2.1 Fast distributed SQL query engine - Trino

2.1.1 Overview

Trino is architected to support fast, scalable analytics across diverse data sources, including relational databases, distributed file systems, and object storage platforms. It exposes a standard ANSI SQL interface, allowing analytical workloads to be expressed declaratively while abstracting away the complexity of distributed execution.

Figure 1: Apache Trino Logo

Key characteristics of Trino include:

- **Distributed execution**: Queries are executed in parallel across multiple worker nodes, enabling efficient processing of large datasets.
- **Schema-on-read**: Data does not need to conform to a single predefined schema, allowing Trino to query heterogeneous datasets with varying structures.
- **Connector-based architecture**: Trino integrates with different data sources through connectors, enabling federated queries across multiple systems.
- **Columnar data optimization**: Native support for columnar formats such as Parquet improves I/O efficiency and query performance.

These features make Trino particularly suitable for healthcare analytics scenarios, where data is distributed across independent hospital systems and stored in different formats with inconsistent schemas.

2.1.2 Mechanism

Trino adopts a master–worker architecture composed of a single coordinator and multiple worker nodes. The coordinator handles query parsing, planning, and scheduling, while worker nodes execute tasks in parallel and stream intermediate results for aggregation.

When a query is submitted, Trino performs the following steps:

1. **Query parsing and planning**: The coordinator parses the SQL statement and generates a distributed execution plan.
2. **Query decomposition**: The plan is divided into multiple stages and tasks that can be executed concurrently.
3. **Parallel execution**: Worker nodes retrieve data directly from the underlying data sources via connectors and execute assigned tasks.
4. **Result aggregation**: Intermediate results are exchanged between workers and aggregated by the coordinator before returning the final output.

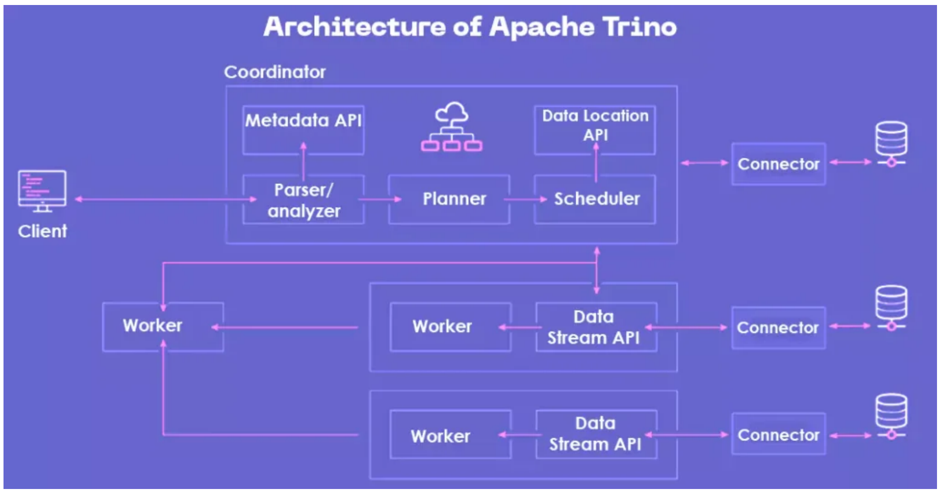


Figure 2: High-level architecture of Apache Trino

In the proposed system, Trino accesses healthcare datasets stored as Parquet files in object storage through a dedicated connector. To handle schema inconsistencies and privacy-induced data gaps across hospital nodes, Trino relies on flexible SQL constructs such as `COALESCE` and `TRY_CAST` to normalize data at query time without enforcing strict schema standardization during ingestion.

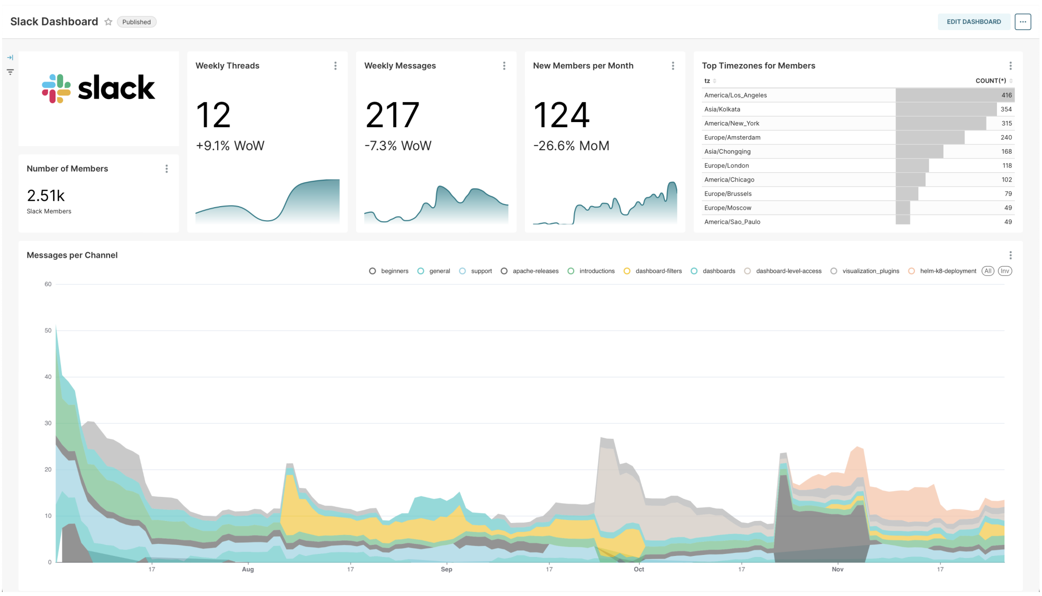
2.2 Big data oriented BI platform - Apache Superset

2.2.1 Overview

Apache Superset is an open-source, enterprise-grade data exploration and visualization platform designed to support large-scale analytics on modern data infrastructures. Developed under the Apache Software Foundation, Superset provides a web-based interface for querying, visualizing, and dashboarding data directly from a wide range of SQL-based data sources. It is built with scalability and extensibility in mind, making it suitable for analytical workloads in data-intensive domains such as healthcare, finance, and e-commerce.

Figure 3: Apache Superset Logo (source: [1])

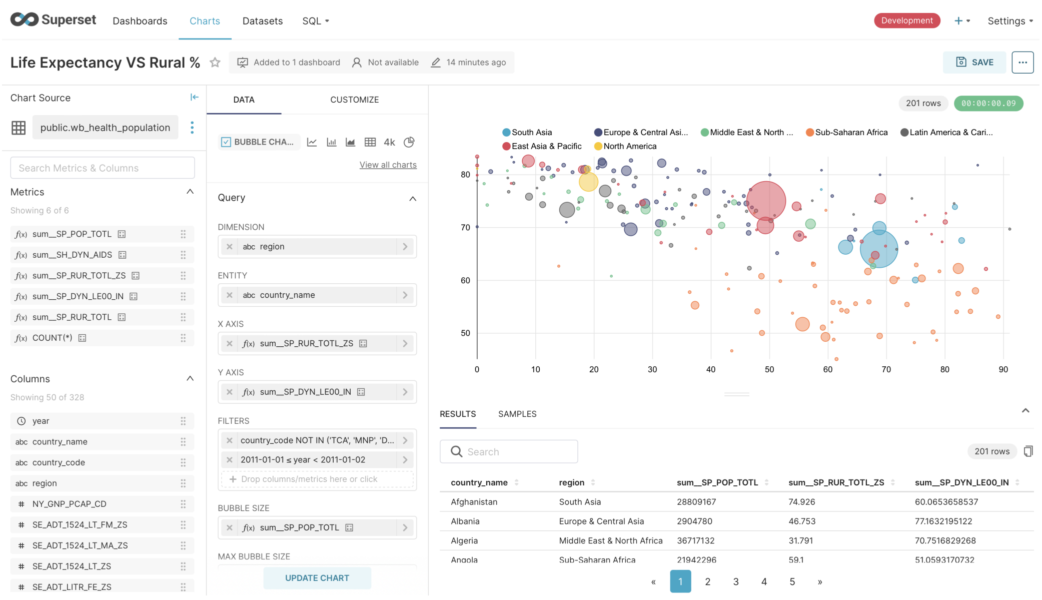
Apache Superset includes native connectivity to diverse data sources, an interactive SQL Lab for ad-hoc querying and exploratory analysis, and flexible visualization and dashboarding capabilities for presenting insights. Superset also provides role-based access control and supports scalable deployments, making it appropriate for real-world analytics scenarios involving large datasets and multiple stakeholders.



Top Timezones for Members

tz	COUNT(*)
America/Los_Angeles	416
Asia/Kolkata	354
America/New_York	315
Europe/Amsterdam	240
Asia/Chongqing	168
Europe/London	118
America/Chicago	102
Europe/Brussels	79
Europe/Moscow	49
America/Sao_Paulo	49

Figure 4: Apache Superset Dashboard Example (source: [1])



DIMENSION: region
ENTITY: country_name
X AXIS: sum__SP_RUR_TOTL_ZS
Y AXIS: sum__SP_DYN_LE00_IN
FILTERS: country_code NOT IN ('TCA', 'MNP', 'D...
2011-01-01 ≤ year < 2011-01-02
BUBBLE SIZE: sum__SP_POP_TOTL

RESULTS SAMPLES

Search 201 rows

country_name	region	sum__SP_POP_TOTL	sum__SP_RUR_TOTL_ZS	sum__SP_DYN_LE00_IN
Afghanistan	South Asia	28809167	74.926	60.0653658537
Albania	Europe & Central Asia	2904780	46.753	77.1632195122
Algeria	Middle East & North Africa	36717132	31.791	70.7516829268
Angola	Sub-Saharan Africa	21942296	59.1	51.0593170732

« 1 2 3 4 5 »

Figure 5: Apache Superset Chart builder Feature (source: [1])

2.2.2 Mechanism

At its core, Superset follows a SQL-centric architecture, enabling direct interaction with databases without requiring data duplication or proprietary storage layers. The platform is implemented using a Python-based backend (Flask, SQLAlchemy) and a modern JavaScript frontend (React), allowing seamless integration with existing data pipelines and cloud-native deployments. Its modular design supports role-based access control, secure multi-tenant usage, and horizontal scalability, which are essential for production-level analytics systems.

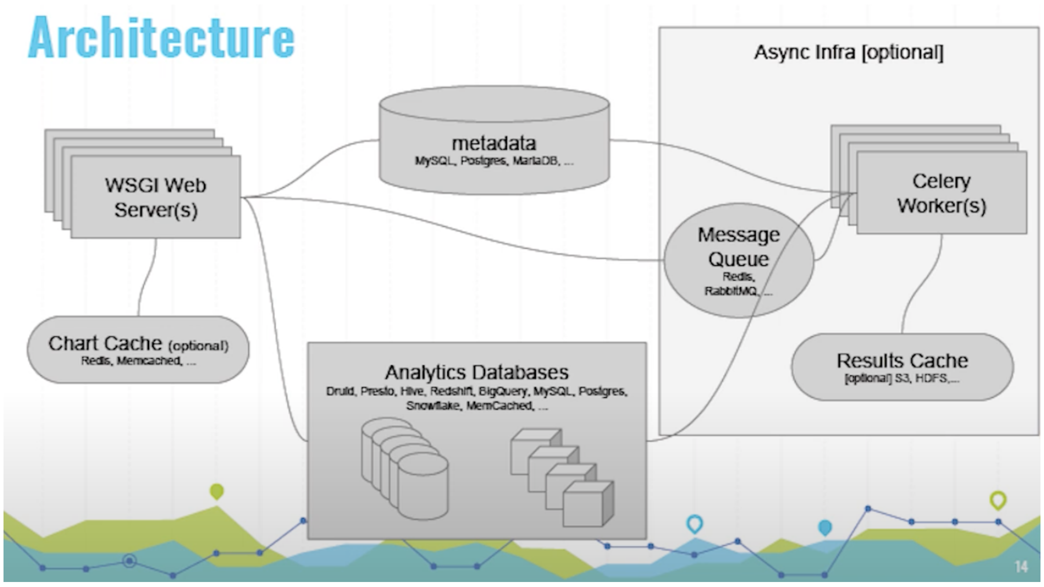


Figure 6: Apache Superset Flow Diagram (source: [1])

Apache Superset connects to external data sources using SQLAlchemy-based database connectors, allowing direct interaction with heterogeneous SQL engines. Once connected, datasets are defined as logical abstractions over database tables or views, enabling consistent reuse across queries and visualizations. Users interact with the data through SQL Lab, where ad-hoc SQL queries can be executed, validated, and optimized, supporting exploratory and iterative analysis.

Query execution is handled asynchronously, with Superset delegating computation to the underlying database engine to leverage its native performance and scalability. Query results are cached when appropriate to reduce latency and improve dashboard responsiveness. The processed results are then passed to Superset's visualization layer, where configurable chart types transform raw query outputs into graphical representations. These visual components can be composed into interactive dashboards with filtering and cross-interaction capabilities.

Security and governance are enforced through fine-grained role-based access control, which regulates access to data sources, datasets, charts, and dashboards. Superset supports scalable deployment models, including containerized and distributed environments, enabling it to handle large datasets and concurrent users efficiently. This mechanism allows Superset to function as an end-to-end analytics layer, bridging raw data storage and actionable insight delivery.

3 Methodology

3.1 System design

We propose a system architecture (illustrated by Figure 7) that is designed to support scalable healthcare analytics by integrating heterogeneous data sources, efficient storage, distributed querying, and visualization. The workflow consists of four main stages: data ingestion, data storage and transformation, query processing, and visualization. It is illustrated in the flow diagram below:

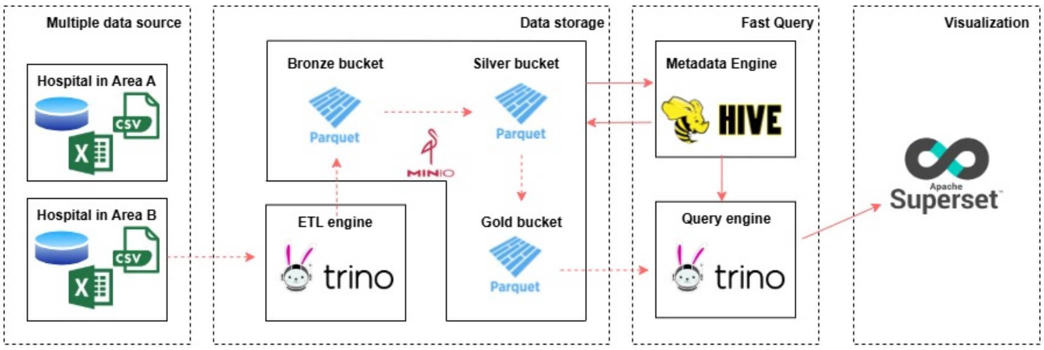


Figure 7: System flow diagram

The data ingestion layer handles multiple data sources originating from hospitals in different regions through TCP/IP over the Internet, and each data source will be interfaced from each hospital's self-hosted server (fig. 8). Due to the widespread adoption of Windows-based systems in healthcare environments, dataset will primarily exist in CSV files, Microsoft Excel files, and tables from Microsoft SQL Server database.

An ETL engine based on Trino is employed to extract data from these sources and standardize it into a unified analytical format. During this process, the raw input data is converted into Apache Parquet, a columnar storage format optimized for analytical workloads. This conversion significantly improves query performance and reduces storage overhead compared to row-based formats such as CSV or Excel.

The transformed data is then stored in MinIO, an object storage system that provides an S3-compatible interface. Data within MinIO is logically organized into Bronze, Silver, and Gold buckets, representing different stages

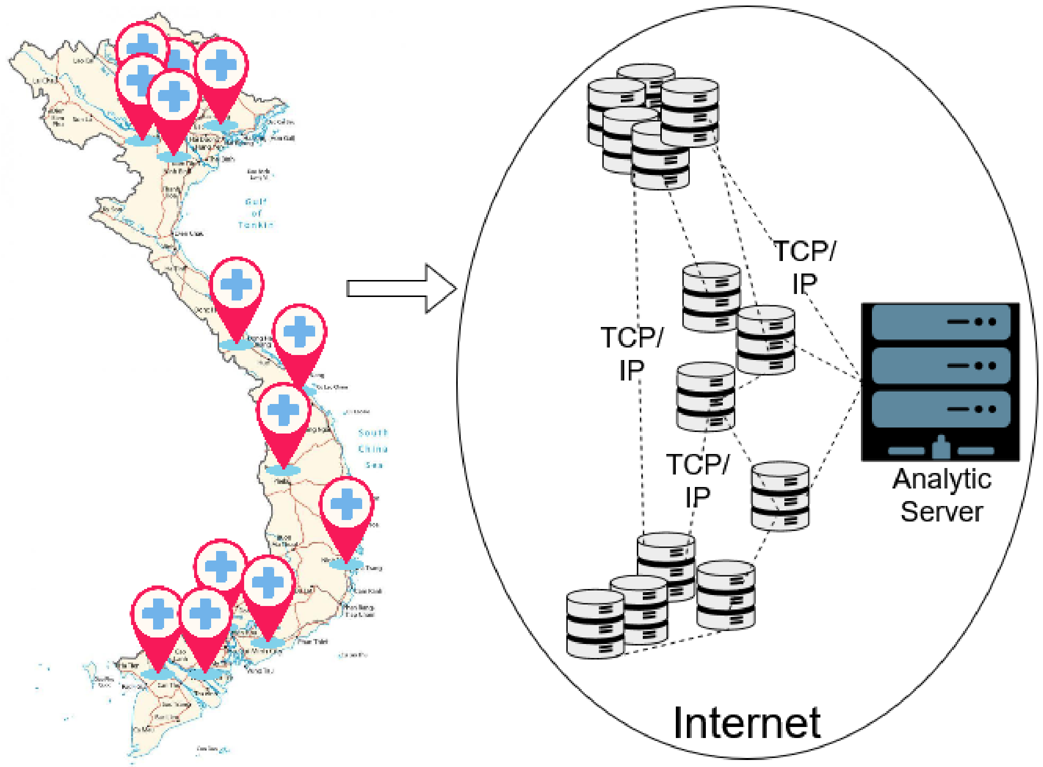


Figure 8: Hospital Network in Internet

of data refinement. The Bronze layer stores raw ingested data, the Silver layer contains cleaned and standardized datasets, and the Gold layer holds curated, analytics-ready data optimized for reporting and decision support.

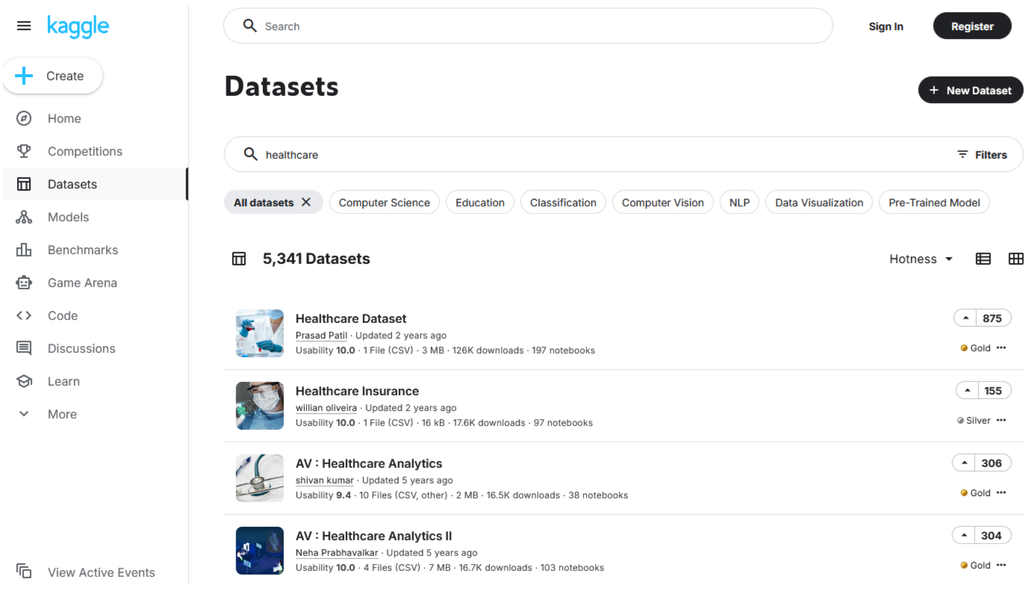
For query processing, Trino is used as the distributed SQL query engine, enabling fast, scalable queries directly over Parquet files stored in MinIO. A Hive Metastore is employed as the metadata engine to manage table schemas, partitions, and dataset locations, allowing Trino to interpret the stored data as structured tables without requiring data movement.

Finally, Apache Superset serves as the visualization layer. It connects to Trino to execute analytical queries and present results through interactive dashboards and charts. The communication can be local-only or through TCP/IP if there is a need for separation of query engine and BI platform.

3.2 Experimental Data

In real world, the amount of health care available for public uses is limited, most of them are protected by data privacy policy which is enforced rigorously by almost all hospitals. The experimental data used for this project will be synthesized using references from https://www.kaggle.com/datasets and Synthea, an open-source data synthesizer by MITRE (USA) to produces realistic, representative healthcare data based on national health statistics and clinical guidelines.

The data will span across multiple files and each file will contain a combination of patient data (admission, treatment process, demographic information, etc.), administrative data (operation, vacancy, review, admission rate, etc.), and disease information in tabular form with various schema. Our data will have size of tens of thousands records.



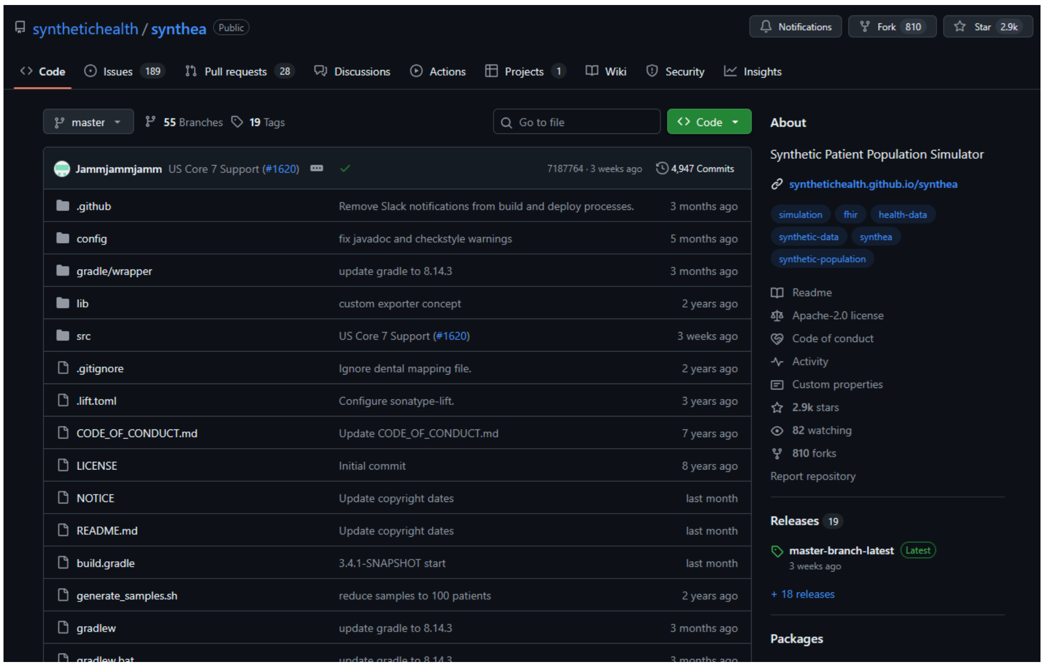
Healthcare Dataset
Prasad Patil · Updated 2 years ago
Usability **10.0** · 1 File (CSV) · 3 MB · 126K downloads · 197 notebooks
875 Gold

Healthcare Insurance
willian oliveira · Updated 2 years ago
Usability **10.0** · 1 File (CSV) · 16 kB · 17.6K downloads · 97 notebooks
155 Silver

AV : Healthcare Analytics
shivan kumar · Updated 5 years ago
Usability **9.4** · 10 Files (CSV, other) · 2 MB · 16.5K downloads · 38 notebooks
306 Gold

AV : Healthcare Analytics II
Neha Prabhavalkar · Updated 5 years ago
Usability **10.0** · 4 Files (CSV) · 7 MB · 16.7K downloads · 103 notebooks
304 Gold

Figure 9: Kaggle Healthcare dataset search



Jammjammjamm US Core 7 Support (#1620) 7187764 · 3 weeks ago 4,947 Commits

.github	Remove Slack notifications from build and deploy processes.	3 months ago
config	fix javadoc and checkstyle warnings	5 months ago
gradle/wrapper	update gradle to 8.14.3	3 months ago
lib	custom exporter concept	2 years ago
src	US Core 7 Support (#1620)	3 weeks ago
.gitignore	Ignore dental mapping file.	2 years ago
.lift.toml	Configure sonatype-lift.	3 years ago
CODE_OF_CONDUCT.md	Update CODE_OF_CONDUCT.md	7 years ago
LICENSE	Initial commit	8 years ago
NOTICE	Update copyright dates	last month
README.md	Update copyright dates	last month
build.gradle	3.4.1-SNAPSHOT start	last month
generate_samples.sh	reduce samples to 100 patients	2 years ago
gradlew	update gradle to 8.14.3	3 months ago

About

Synthetic Patient Population Simulator

synthetichealth.github.io/synthea

simulation fhir health-data synthetic-data synthea synthetic-population

Readme
Apache-2.0 license
Code of conduct
Activity
Custom properties
2.9k stars
82 watching
810 forks
Report repository

Releases 19

master-branch-latest Latest
3 weeks ago

+ 18 releases

Packages

Figure 10: Synthea Github page

4 Implementation

4.1 Dataset & Datasource

4.1.1 Dataset Selection & Overview

Due to strict patient privacy constraints, real-world patient-level datasets are rarely published for public research and system benchmarking. Therefore, this project adopts a synthetic healthcare dataset to enable demographic and clinical analytics without exposing sensitive personal information.

We use **Synthea Synthetic Patient Records** (MITRE, USA) [2], an open-source generator that produces EHR-like records following realistic population-based distributions. The dataset is distributed in CSV format and contains multiple tables covering demographics, encounters, diagnoses, procedures, medications, and healthcare organizations.

The official download source: `https://synthea.mitre.org/downloads`.

In this project, we use the **1K Sample Synthetic Patient Records (CSV)** package. The selected core schema and analytical tables are summarized in Section 4.1.2.

Package	**Format**	**Size**	**Description**
1K Sample Synthetic Patient Records	CSV	~80 MB	Approximately 1,000 synthetic patients and 18 tables (e.g., patients, encounters, conditions, procedures, medications, organizations, etc.).

Table 2: Selected Synthea dataset package used in this project.

4.1.2 Core Data Schema

From the Synthea package, we select six core tables that are sufficient to represent an end-to-end patient journey (demographics → visits → diagnoses/procedures/medications) together with the provider context. This schema remains compact enough for federated, distributed querying.

The selected core tables include:

- `patients`: contains demographic and basic information of synthetic patients, such as gender, birthdate, race, and geographic attributes.
- `encounters`: represents patient visits and interactions with healthcare providers, including timestamps, payer information, and encounter-related costs.
- `conditions`: records diagnosed medical conditions associated with patient encounters, identified by standardized medical codes.
- `procedures`: describes clinical procedures performed during encounters, together with procedure codes, dates, and associated costs.
- `medications`: captures prescribed or administered medications, including drug codes, duration, and cost-related attributes.
- `organizations`: stores information about healthcare providers and hospital entities responsible for patient encounters.

These tables are logically connected through primary and foreign key relationships. Each patient may have multiple encounters, and each encounter may be associated with multiple conditions, procedures, and medications. Healthcare organizations are linked to encounters, providing the institutional context for clinical activities. Figure 11 illustrates the logical relationships among the selected core tables.

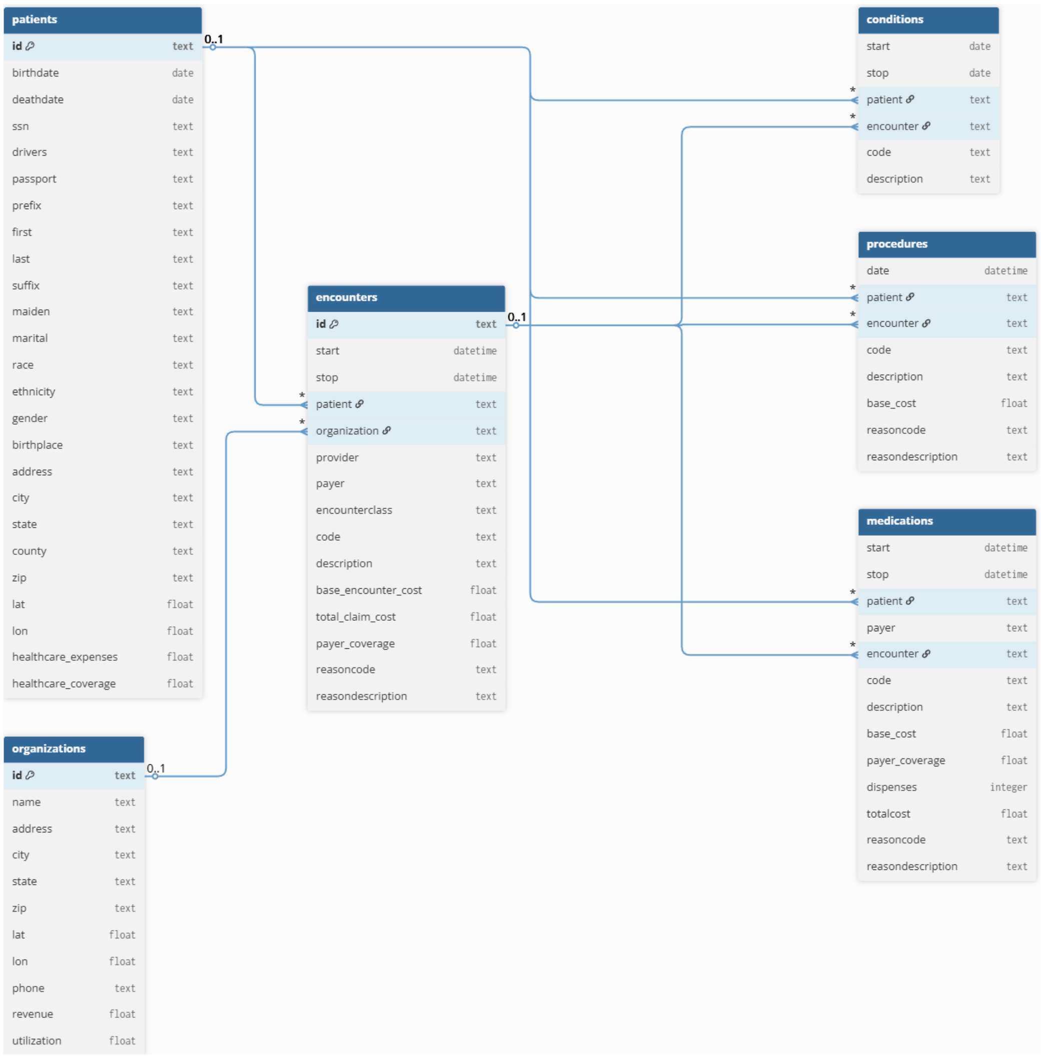


Figure 11: Logical schema of the 6 Synthea core tables and their relationships.

To support analytical workloads, the selected schema provides key attributes required for demographic analysis, disease trend analysis, treatment cost evaluation, and policy impact assessment. Table 3 summarizes the core tables and their primary analytical roles.

Problem	Files Used	Application / Key Attributes
1. Patient Demographic	patients.csv	Demographic attributes: gender, age, race, ethnicity, income, insurance, and location.
2. Disease Trend	conditions.csv, patients.csv	Disease occurrence and temporal trends by gender and geographic region.
3. Treatment Cost Trend	encounters.csv, procedures.csv, medications.csv	Healthcare cost analysis by disease type, time period, and treatment category.
4. Policy Impact	encounters.csv, patients.csv	Cost comparison across insurance types and income levels.

Table 3: Mapping between project problems and the selected Synthea tables.

Overall, this core schema is the basis for the subsequent 8-node partitioning and heterogeneity simulation, and it is designed to work well with Trino's schema-on-read processing.

4.1.3 Multi-node Data Architecture

To model a realistic distributed healthcare environment, the prepared dataset is partitioned into **eight independent data nodes**, each representing a separate hospital group or healthcare provider cluster.

Partitioning strategy

The dataset is divided using an *organization-centric* strategy:

- Each healthcare organization is assigned to exactly one node.
- All encounters, conditions, procedures, and medications related to that organization are placed in the same node.
- Patient records may appear in multiple nodes, reflecting patients receiving care at different hospitals.

Node structure

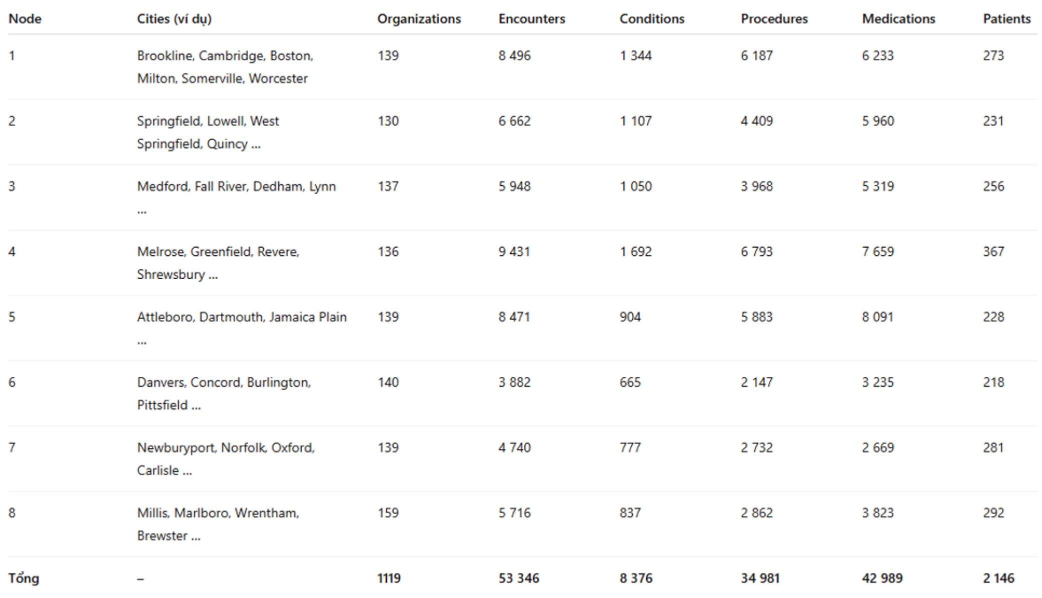
Each node contains the same set of six core tables:

- `patients`
- `encounters`
- `conditions`
- `procedures`
- `medications`
- `organizations`

While the table structure is consistent across nodes, the underlying data distributions differ, resulting in heterogeneous node characteristics.

Node statistics.

Figure 12 illustrates the data distribution across the eight nodes after organization-centric partitioning. The variation in the number of organizations, encounters, and clinical records reflects realistic differences in hospital sizes and operational workloads.



Node	Cities (ví dụ)	Organizations	Encounters	Conditions	Procedures	Medications	Patients
1	Brookline, Cambridge, Boston, Milton, Somerville, Worcester	139	8 496	1 344	6 187	6 233	273
2	Springfield, Lowell, West Springfield, Quincy ...	130	6 662	1 107	4 409	5 960	231
3	Medford, Fall River, Dedham, Lynn ...	137	5 948	1 050	3 968	5 319	256
4	Melrose, Greenfield, Revere, Shrewsbury ...	136	9 431	1 692	6 793	7 659	367
5	Attleboro, Dartmouth, Jamaica Plain ...	139	8 471	904	5 883	8 091	228
6	Danvers, Concord, Burlington, Pittsfield ...	140	3 882	665	2 147	3 235	218
7	Newburyport, Norfolk, Oxford, Carlisle ...	139	4 740	777	2 732	2 669	281
8	Millis, Marlboro, Wrentham, Brewster ...	159	5 716	837	2 862	3 823	292
Tổng	–	**1119**	**53 346**	**8 376**	**34 981**	**42 989**	**2 146**

Figure 12: Data distribution across the eight nodes after organization-centric partitioning.

Data layout

After partitioning, the dataset is organized into a hierarchical directory structure, where each node stores its own set of six tables. This layout facilitates independent access and federated querying by the Trino engine.

```
nodes_csv_all/
  node_1/
    patients.csv
    encounters.csv
    conditions.csv
    procedures.csv
    medications.csv
    organizations.csv
  node_2/
    patients.csv
    encounters.csv
    conditions.csv
    procedures.csv
    medications.csv
    organizations.csv
  ...
  node_8/
    patients.csv
    encounters.csv
    conditions.csv
    procedures.csv
    medications.csv
    organizations.csv
```

Design rationale

This multi-node architecture establishes a clean distributed baseline before introducing datasource heterogeneity and privacy constraints. It closely mirrors real-world healthcare systems, where data is decentralized across multiple institutions but must still support unified analytical queries.

4.1.4 Datasource Challenges Simulation

In real-world healthcare systems, data from different hospitals is heterogeneous and subject to privacy restrictions. Therefore, the eight data nodes are intentionally modified to simulate two major challenges: **datasource incompatibility** and **data privacy constraints**.

Data source incompatibility

- Schema differences across nodes (added, removed, or renamed columns).
- Data type and format variations (numeric stored as text, different date-time formats).
- Inconsistent medical coding standards (ICD-9, ICD-10, CPT, RxNorm).
- Additional internal metadata or missing data due to different export versions.

Data privacy constraints

- Masking or removal of personally identifiable information (PII).
- Redaction or suppression of sensitive clinical data.
- Partial data sharing with only a subset of records or attributes.
- Reduced location precision through geographic anonymization.

Design purpose

Despite differences in schema, format, and data completeness, the distributed nodes can still be queried in a unified manner using Trino's schema-on-read and federated query capabilities.

4.2 Fast distributed SQL query engine - Trino

Trino is in progress of local deployment. However, with synthesized dataset mentioned in the back-end progression section, we have defined a template query (likely subjected to change) for Healthcare Policy impact problem (basing on organizations.csv and providers.csv):

4.2.1 System architecture and data flow

Figure 5 presents the overall system architecture and data flow adopted in this project. Healthcare data is collected from multiple hospitals located in different geographical areas (for example multiple area) and provided in structured formats such as CSV and Excel files.

The system follows a layered data lake architecture consisting of three storage layers: Bronze, Silver, and Gold, all deployed on object storage. Raw data from multiple sources is first ingested into the Bronze layer in Parquet format with minimal transformation. This ensures that original data is preserved for traceability and potential reprocessing.

Using Trino as an ETL engine, data from the Bronze layer is processed and transformed into the Silver layer. At this stage, common data preparation tasks such as schema alignment, data type normalization, and duplicate removal are applied to standardize heterogeneous datasets from different hospitals.

Finally, the Gold layer contains analytics-ready datasets derived from the Silver layer. This layer stores aggregated and policy-oriented data that directly supports analytical queries and visualization. Trino provides a unified SQL interface across all data layers, enabling both transformation and querying without moving data outside the object storage.

- **Multiple data sources**: Raw healthcare data is collected from multiple source files representing different aspects of healthcare operations. The datasets are provided in structured CSV format, which was selected due to its practicality and effectiveness for data processing, and include the following key files from 8 nodes represent 8 hospital areas. In each node includes collection of tables:
  - `patients.csv`: contains demographic and identification information of patients.
  - `encounters.csv`: records patient encounters with healthcare facilities, including visit dates and encounter types.
  - `conditions.csv`: stores diagnosed medical conditions associated with patient encounters.

- `procedures.csv`: captures medical procedures performed during healthcare encounters.
- `medications.csv`: represents prescribed or administered medications.
- `organizations.csv`: contains information about healthcare organizations such as hospitals and clinics.

Each dataset originates from different operational subsystems and represents a specific domain of healthcare data. Although the files are structurally organized, they are logically connected through shared identifiers such as patient IDs, encounter IDs, and organization IDs. As a result, meaningful analysis requires integrating multiple datasets rather than relying on a single data source.

In addition, differences in record volume, update frequency, and data completeness exist across these datasets. For example, encounter and medication records are significantly larger and more frequently updated than organizational reference data. These characteristics reflect realistic healthcare data complexity and motivate the use of a distributed data processing and querying approach.

4.2.2 Bronze layer (raw data)

The Bronze layer represents the initial ingestion stage of the data pipeline, where raw healthcare datasets are collected and prepared for further processing. This stage focuses on format conversion and schema registration while preserving the original data semantics.

CSV to Parquet conversion Raw healthcare datasets are originally provided in CSV format, which is suitable for data exchange but inefficient for large-scale analytical querying. To improve storage efficiency and query performance, all CSV files are converted into Parquet format during the ingestion process.

Trino is used as the query engine to directly read CSV files from the source directories and write the data into Parquet files stored in the Bronze bucket. This conversion is implemented using SQL-based ingestion queries, enabling consistent handling of multiple datasets without introducing additional ETL tools. Only minimal transformations, such as basic data type casting and column normalization, are applied to ensure that the data remains as close as possible to its original form.

The use of Parquet provides columnar storage, compression, and improved I/O efficiency, which are essential for scalable data processing in subsequent layers.

Schema definition and metadata registration After converting raw data into Parquet format, logical table schemas are defined to map the stored datasets into Trino. These schemas specify column names, data types, and storage locations, allowing Trino to interpret Parquet files as relational tables.

The schema definitions are registered in the Hive Metastore, which acts as a centralized metadata repository for the data lake. By storing schema information separately from the data itself, the system follows a schema-on-read approach, enabling flexible schema evolution without rewriting existing data.

Registering Bronze-layer tables in Hive Metastore allows Trino to query raw datasets consistently across different data sources and provides a foundation for downstream transformations into the Silver and Gold layers. This design ensures data discoverability, governance, and reproducibility throughout the data processing pipeline.

Schema creation and table registration in Hive Metastore After converting raw CSV files into Parquet format and storing them in the Bronze bucket, logical table schemas are created to enable Trino to query the data in a structured and consistent manner. In this project, a dedicated schema (`hospital_id`) is defined to represent data originating from a specific hospital area source.

Within this schema, multiple tables are created to map different healthcare data domains, including patients, conditions, procedures, encounters, medications, and organizations. Each table definition specifies column names and data types that correspond to the structure of the underlying Parquet files. This explicit schema definition ensures that Trino can correctly interpret the stored data and perform type-safe analytical queries.

The tables are created as external tables using the `external_location` property, which points to the corresponding Parquet directories in object storage. By using external tables, the data remains physically stored in the Bronze layer while metadata information is registered in the Hive Metastore. This separation between data storage and metadata management enables a schema-on-read approach and avoids unnecessary data duplication.

For example, the `patients` table captures demographic and geographic information of patients, while the `encounters` table records detailed information about healthcare visits, costs, and coverage. Similarly, domain-specific tables such as `conditions`, `procedures`, and `medications` provide clinical and treatment-related data that can be joined through shared identifiers such as patient IDs and encounter IDs. The `organizations` table stores reference information about healthcare facilities, supporting organizational-level and regional analysis.

Registering these tables in the Hive Metastore allows Trino to treat Parquet files in the Bronze layer as relational tables that can be queried using standard SQL. This design enables efficient joins and aggregations across multiple datasets and establishes a consistent foundation for subsequent data cleaning and transformation processes in the Silver and Gold layers.

4.2.3 Silver layer (standardized and anonymized data)

The Silver layer is designed to store standardized, analytics-ready datasets derived from the Bronze layer. Unlike the Bronze layer, which preserves raw data, the Silver layer performs essential transformations for downstream analysis, including schema normalization, anonymization, and dimensional modeling. In this project, the Silver layer is implemented under the schema `minio.silver`, with its storage location configured on object storage:

- **Schema location**: `s3a://final-project-bigdata/silver/`
- **Storage format**: Parquet (columnar format for efficient analytical queries)

Dimensional model design (Dim / Fact tables) To support demographic analysis and policy-oriented analytics, the Silver layer adopts a star-schema style structure. This includes a set of *dimension tables* for descriptive attributes and *fact tables* for event-based measures.

- **Dimensions**:

  * `dim_patient`: demographic attributes and location fields, prepared for analysis with anonymization.
  * `dim_organization`: organization reference data (facility location, revenue, utilization).
  * `dim_date`: time dimension to support time-series analysis and calendar-based aggregation.
  * `dim_condition_code`, `dim_procedure_code`, `dim_medication_code`: reference dictionaries for code-to-description mapping.
- **Facts**:
  * `fact_encounter`: encounter-level events and cost-related measures.
  * `fact_condition`: conditions associated with encounters, including severity and effective dates.
  * `fact_procedure`: procedures performed, including base cost and reason fields.
  * `fact_medication`: medication events, including cost, coverage, and dispenses.

Anonymization strategy for patient data To protect sensitive patient identifiers while still enabling analytical joins, patient IDs from the Bronze layer are anonymized using a deterministic hashing approach. Specifically, the original patient identifier is transformed into `patient_hash` using `to_hex(md5(to_utf8(id)))`. This ensures:

- Patient identity is masked (no direct exposure of raw IDs in Silver tables)
- The same patient can still be joined consistently across tables (patients, encounters, conditions, procedures, medications)

In addition, `dim_patient` includes derived demographic features such as `age` and `age_group` to support cohort-based analysis (e.g., 0–17, 18–35, 36–60, 60+).

Time standardization with `dim_date` A dedicated `dim_date` table is generated to provide consistent date keys (`YYYYMMDD`) and calendar attributes (day, month, year, quarter, weekday, weekend flag). This enables efficient joins and aggregation in the fact tables. Two approaches are supported:

- Generating a fixed date range (e.g., multiple decades)
- Generating dynamic bounds based on min/max encounter dates

Population of Silver tables from Bronze (hospital_8 as initial source) For the prototype stage, Silver tables are populated using data from `minio.hospital_8.*` as the initial source system. Each record is tagged with `source_system = 'hospital_8'` to support future multi-hospital integration.

Dimension tables are loaded first to establish consistent keys and reference mappings:

- `dim_patient`: generated `patient_key` and `patient_hash`, plus standardized demographic features.
- `dim_organization`: generated `org_key` and mapped organization attributes.
- `dim_condition_code`, `dim_procedure_code`, `dim_medication_code`: populated using `SELECT DISTINCT` on code and description fields.

Fact tables are then populated using Bronze event datasets:

- `fact_encounter`: generated `encounter_key`, computed date keys from timestamps, and joined with `dim_organization` to resolve `org_key`.
- `fact_condition`, `fact_procedure`, `fact_medication`: derived from the corresponding Bronze tables and standardized to use hashed patient identifiers and date keys.

Overall, the Silver layer provides a consistent, anonymized, and analysis-ready foundation for downstream aggregation in the Gold layer and for healthcare policy impact queries.

4.2.4 Gold layer (analytical data marts)

The Gold layer represents the final stage of the data processing pipeline and is specifically designed to support analytical reporting, visualization, and healthcare policy analysis. At this stage, data is no longer stored at an event or transactional level but is aggregated into subject-oriented data marts that directly answer predefined analytical questions.

A dedicated schema, `minio.gold`, is created to logically separate analytical outputs from the standardized datasets in the Silver layer. All Gold-layer tables are stored in Parquet format on object storage, ensuring efficient access for BI tools and analytical queries executed through Trino.

Design principles of the Gold layer The design of the Gold layer follows three main principles:

- **Aggregation**: reduce data volume by summarizing large fact tables into concise analytical metrics.
- **Business orientation**: structure tables around analytical use cases rather than operational entities.
- **Reusability**: create stable data marts that can be reused across dashboards and reports without additional transformations.

Data Mart 01: Patient Demographic Analysis The data mart mart_patient_demographic is designed to analyze patient population distribution across demographic and geographical dimensions. It aggregates encounter data on a yearly basis and groups results by gender, age group, state, and city.

To ensure privacy preservation, patient counts are calculated using distinct anonymized patient identifiers derived in the Silver layer. This approach allows population-level analysis while preventing direct exposure of sensitive personal information.

The resulting data mart supports analytical questions such as:

- How is the patient population distributed across age groups and gender?
- Are there regional differences in healthcare utilization?
- How does patient distribution vary across different source systems (hospitals)?

Data Mart 02: Disease Trend Analysis (Monthly) The data mart mart_disease_trend_monthly focuses on analyzing disease occurrence trends over time. It aggregates condition-level events at a monthly granularity, grouped by disease code, location, and source system.

By joining standardized fact and dimension tables from the Silver layer, this data mart enables time-series analysis of disease patterns and supports the identification of seasonal effects or regional variations in disease prevalence.

Typical analytical questions supported by this data mart include:

- Which diseases show increasing or decreasing trends over time?
- Are there seasonal patterns in disease occurrence?
- How do disease trends differ across regions or hospitals?

Role of Trino in Gold layer generation Trino is used to generate Gold-layer data marts through SQL-based aggregation queries executed directly on the Silver layer. By leveraging Trino's distributed execution capabilities, large fact tables can be efficiently aggregated without requiring additional processing frameworks.

This approach simplifies the overall system architecture by using a single query engine across Bronze, Silver, and Gold layers, while still supporting scalable analytical workloads.

Usage and downstream integration The Gold layer serves as the primary data source for visualization and exploratory analysis. Due to its aggregated and stable structure, Gold-layer data marts can be directly consumed by BI tools and dashboards without further transformation. This separation of concerns allows analytical users to focus on interpretation and decision-making rather than data preparation.

Overall, the Gold layer transforms standardized healthcare data into actionable analytical insights, forming the foundation for evidence-based healthcare policy evaluation.

All data layers are stored on **MinIO**, which provides S3-compatible object storage and enables scalable, decoupled compute and storage.

4.2.5 Metadata and query execution

To enable efficient querying across the data lake, Hive Metastore is used as a metadata engine. Hive Metastore maintains table schemas, partitions, and storage locations for Parquet files across the Bronze, Silver, and Gold layers.

Trino connects to Hive Metastore to retrieve metadata and directly queries Parquet files stored in MinIO without requiring data movement. This architecture enables:

- Low-latency, distributed SQL queries
- Schema-on-read flexibility
- Cross-layer querying (e.g., validating Silver data against Bronze sources)

For analytical workloads and dashboarding, Trino serves as the primary query engine and is integrated with Apche Superset. This allows policymakers and analysts to perform interactive exploration and visualization of healthcare indicators derived from the Gold layer.

4.2.6 Application to healthcare policy impact analysis

Healthcare policy evaluation typically requires integrating multiple perspectives of the healthcare system rather than relying on a single dataset. In this project, analytical queries are executed over six interrelated datasets representing different aspects of healthcare operations, including patients, encounters, conditions, procedures, medications, and organizations.

Each dataset captures a distinct dimension of the healthcare system. Patient and organization tables provide demographic and geographical context, while encounter, condition, procedure, and medication tables represent healthcare utilization and clinical activities. Individually, these datasets offer limited insight; however, when combined, they enable a comprehensive view of healthcare service delivery and resource allocation.

To support policy-oriented analysis, the system integrates these datasets to derive analytical indicators that would not be obtainable from individual tables in isolation:

- **Demographic perspective**: patient attributes such as age group, gender, and location provide population-level context.
- **Utilization perspective**: encounter, procedure, and medication records reflect healthcare service usage and treatment intensity.
- **Clinical perspective**: condition-level data captures disease occurrence and severity patterns.

Trino is used to join and aggregate these six datasets through shared identifiers such as patient hashes, encounter identifiers, organization keys, and date keys. By consolidating information across demographic, clinical, and organizational dimensions, the system enables policy-oriented analyses that relate population characteristics to healthcare utilization patterns, disease occurrence, and treatment activities across different regions and institutions.

From a data volume perspective, event-based tables such as encounters, conditions, procedures, and medications contain significantly more records than reference tables such as patients and organizations. As a result, analytical queries frequently involve aggregating large fact tables and joining them with smaller dimension tables. Trino's distributed query execution model allows these operations to be performed efficiently, even as the number of records across datasets increases.

To further reduce analytical complexity and improve reusability, the results of these integrations are materialized in the Gold layer as analytical data marts:

- Complex joins and aggregations across multiple Silver-layer tables are executed once during data mart creation.
- The resulting Gold-layer tables provide stable, reusable views for dashboards and exploratory analysis.
- Analytical consistency is maintained across reports, as all downstream queries reference the same pre-aggregated structures.

By consolidating integrated data and exposing it through reusable analytical views, the system supports healthcare policy analysis use cases such as assessing regional healthcare demand, identifying disparities in service utilization, and evaluating the impact of healthcare policies on different population groups. Overall, Trino serves as the analytical backbone that unifies heterogeneous healthcare data into coherent, scalable, and policy-relevant insights.

4.3 BI Platform - Apache Superset

4.3.1 Setup

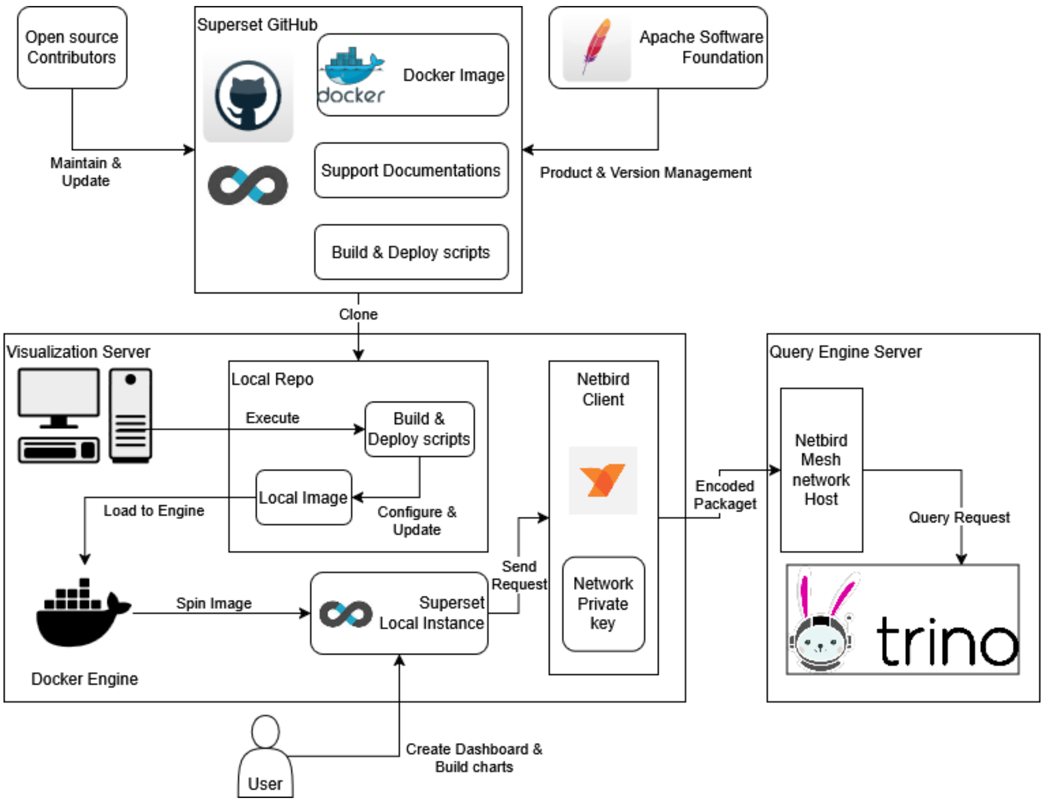
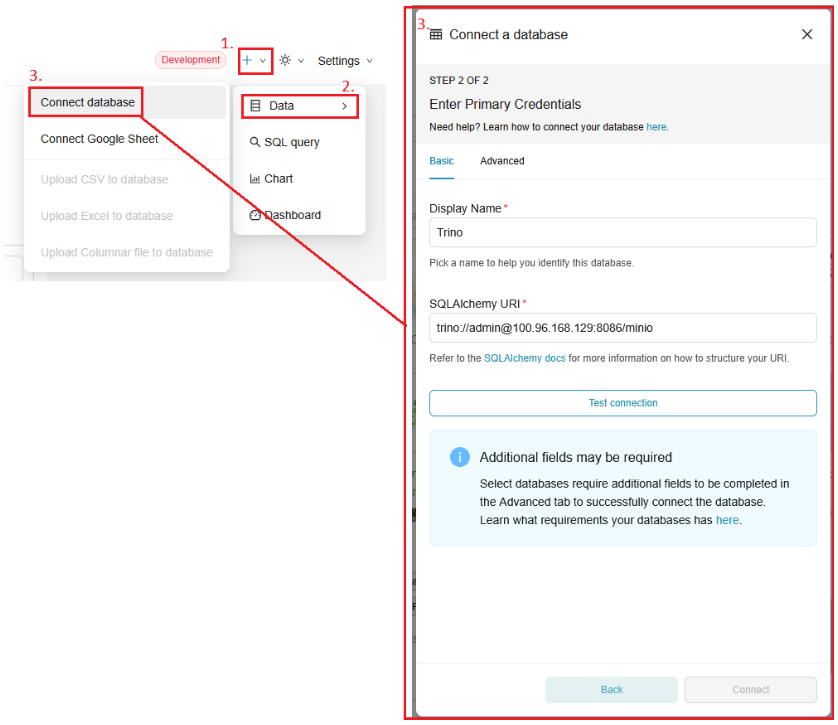


Figure 13: Superset Implementation in local machine

The whole setup flow diagram is described by Figure 13 above. From the flow diagram, Apache Superset is distributed by Apache Software Foundation through public open-source GitHub repository (https://github.com/apache/superset). The repository includes a docker image which contains the binary of Superset and relevant scripts to configure and deploy Superset. We will clone the repository into our local machine (visualization server) and use provided scripts to run our local Superset host instance on Docker Engine (using docker-compose command with "docker-compose-image-tag.yml" script). The instance will follow default configuration set by Apache. In this case, our instance will host at http://localhost:8088/ and can be logged in using "admin"/"admin" credential.

To let Superset communicate with Query Engine (Trino) server, we will install a Netbird client, an open source platform for Zero Trust Network Access over TCP/IP protocol, on the same machine Superset installed on and supply it with our private network key to access the private network hosted on the Trino server. Before sending query request to Trino, Superset must know the address of our Trino instance. To solve this, we supply a connection string to Superset using ”trino://username:password@hostname:port/catalog” format. The connection string can be input through UI as follows:



Need help? Learn how to connect your database here.

Display Name *
Trino
Pick a name to help you identify this database.

SQLAlchemy URI *
trino://admin@100.96.168.129:8086/minio
Refer to the SQLAlchemy docs for more information on how to structure your URI.

Additional fields may be required
Select databases require additional fields to be completed in the Advanced tab to successfully connect the database. Learn what requirements your databases has here.

Figure 14: Superset Database connection UI with Trino connection string

4.3.2 Dashboard creation

Superset supports Dashboard display and the dashboard will be the layout containing multiple curated data charts that visualize the insight received from query to our Trino engine. The charts are our visualization building block and will follow implementation strategies below for each of the four analytic problems:

- Patient Demographics:
  - Cluster Bar Charts: Age cohorts are mapped to the ordinal x-axis with gender categories differentiated by color hue within each cluster. This approach permits precise comparative analysis of

population structure, enabling the identification of age-sex specific dominance and demographic shifts.

  - Choropleth Maps (Spatial Density Analysis): Geographic distribution is rendered through choropleth mapping, where administrative regions (e.g., zip codes) are shaded according to patient density metrics. This spatial analysis elucidates regional disparities in healthcare access, highlighting clusters of high service demand versus underserved rural localities to inform resource allocation strategies.

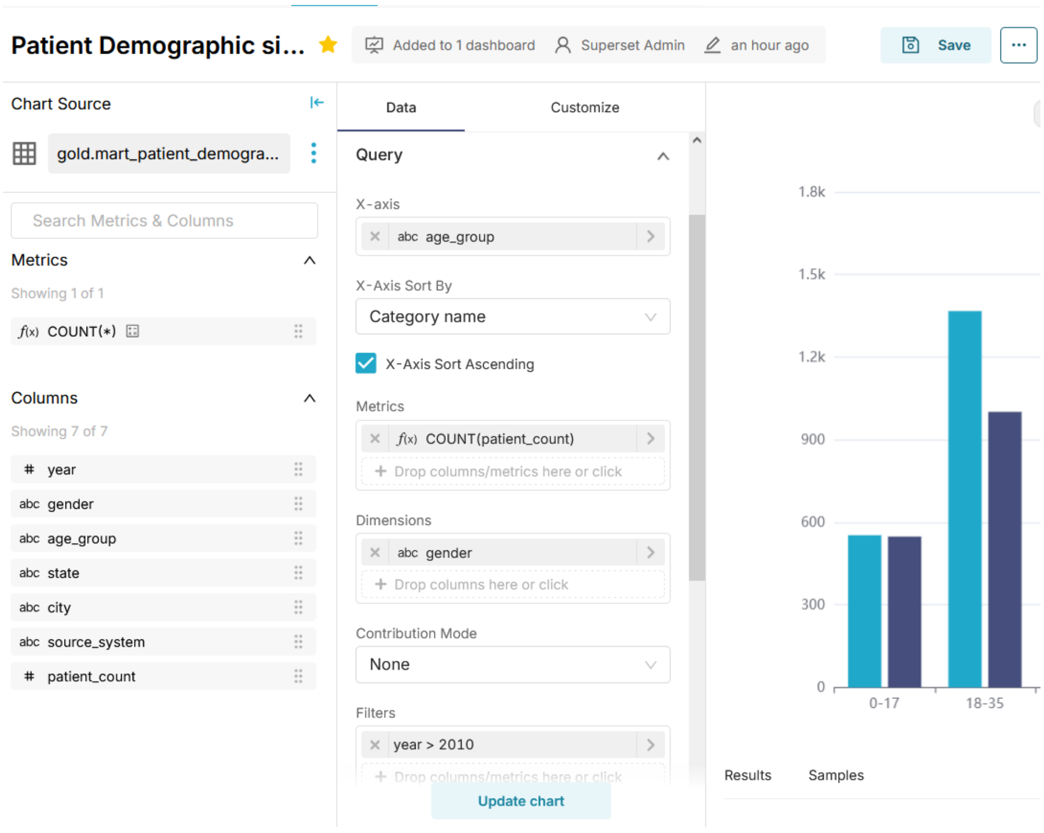
- Disease trend:
  - Multi-Line Time Series Charts (Longitudinal Incidence): Disease prevalence is monitored via longitudinal line charts, plotting incidence rates on a continuous temporal scale. Distinct pathologies are separated by color channels, facilitating the detection of divergent trends, inflection points in infection rates, and comparative analysis of multiple morbidity vectors simultaneously.
  - Calendar Heat Maps (Periodicity Detection): To diagnose temporal clustering and seasonality, incident volumes are projected onto a calendar matrix (Day x Month). Color saturation intensity encodes case frequency, thereby exposing cyclical epidemiological patterns—such as seasonal influenza spikes or weekday admission variability—that remain obscured in aggregate linear trends.

- Treatment cost trend:
  - Box-and-Whisker Plots (Statistical Dispersion): Cost data is analyzed using box plots to visualize statistical distribution rather than central tendency alone. By delineating the interquartile range (IQR) and median, this visualization isolates systemic cost variability from outlier anomalies, providing a robust method for auditing billing irregularities and procedure-specific cost volatility.
  - Bivariate Scatter Plots (Correlational Analysis): The relationship between economic and clinical variables is examined by plotting total episode cost against independent variables such as Length of Stay (LOS) or patient age. This bivariate approach facilitates the identification of linear or non-linear correlations, determining the extent to which specific clinical factors function as primary cost drivers.

- Impact of healthcare policy:

- Interrupted Time Series with Annotations (Causal Inference): Policy efficacy is evaluated using time-series charts augmented with vertical annotation layers marking intervention dates. This visualization approximates an Interrupted Time Series (ITS) design, allowing for the visual assessment of level changes or slope deviations in Key Performance Indicators (KPIs) post-implementation, thus supporting statistical validity in impact reporting.
- Paired Interval (Dumbbell) Charts (Magnitude of Change): To quantify the absolute delta between pre- and post-policy states, a paired interval chart is utilized. Categories are plotted with distinct markers for baseline and post-intervention values connected by a directional segment; the length of this segment visually encodes the magnitude of the effect, isolating the intervention's impact on specific hospital units or patient cohorts.

To build a chart, we supply the data from our analytical data marts (gold layer, in Section 4.2.4). Currently, we have two data marts compiled: Patient Demographic Analysis and Disease Trend Analysis. Each data mart will correspond to a chart; in this case, Patient Demographic Analysis data mart corresponds to a Cluster Bar chart (containing information about patient count of two genders across multiple age groups within a time period, illustrated in chart builder UI in Figure 15) and Disease Trend Analysis data mart corresponds to a seasonal Heatmap chart (containing information about frequency of Common cold case across seasons within a time period, illustrated in chart builder UI in Figure 16).



Patient Demographic si...

Added to 1 dashboard · Superset Admin · an hour ago

Chart Source: gold.mart_patient_demogra...

Metrics (Showing 1 of 1): COUNT(*)

Columns (Showing 7 of 7): year, gender, age_group, state, city, source_system, patient_count

Query

X-axis: age_group

X-Axis Sort By: Category name

X-Axis Sort Ascending: ✓

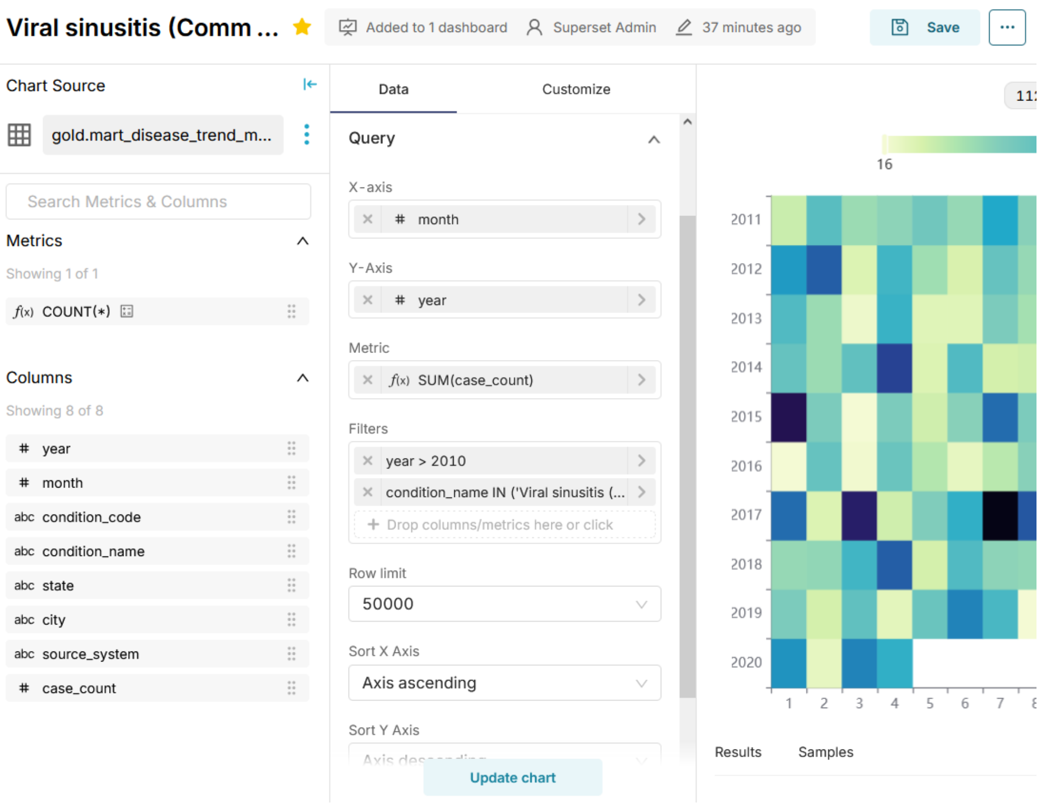
Metrics: COUNT(patient_count)

Dimensions: gender

Contribution Mode: None

Filters: year > 2010

Figure 15: Cluster Bar chart builder using dataset from Patient Demographic Analysis data mart


Viral sinusitis (Comm ...

Added to 1 dashboard · Superset Admin · 37 minutes ago

Chart Source: gold.mart_disease_trend_m...

Metrics: Showing 1 of 1
- COUNT(*)

Columns: Showing 8 of 8
- # year
- # month
- abc condition_code
- abc condition_name
- abc state
- abc city
- abc source_system
- # case_count

Query
- X-axis: # month
- Y-Axis: # year
- Metric: SUM(case_count)
- Filters: year > 2010; condition_name IN ('Viral sinusitis (...
- Row limit: 50000
- Sort X Axis: Axis ascending
- Sort Y Axis: Axis descending

Figure 16: Heatmap chart builder using dataset from Disease Trend Analysis data mart

5 Result & Evaluation

5.1 Result

5.1.1 Dataset and Data source

The data preparation workflow in this project is summarized as follows:

`Raw Synthea CSV (6 tables)` → `Partition into 8 Nodes` → `Schema & Privacy Simulation` → `Final Parquet Output`

The final Parquet datasets represent the completed data preparation stage and are handed over to the Trino query engine. These datasets support schema-on-read processing, unified views, and distributed SQL queries across heterogeneous healthcare data sources.

5.1.2 Fast distributed query engine and Data marts

The completed implementation demonstrates several significant outcomes at both the technical and analytical levels. Although real patient data could not be used due to healthcare privacy regulations, the use of synthesized datasets enabled a full end-to-end validation of the proposed data lake and analytics pipeline, from ingestion and standardization to policy-oriented analytical outputs.

At the system level, Trino was successfully deployed as a fast distributed SQL query engine and integrated as the central analytical layer across the Bronze, Silver, and Gold data tiers. Unlike traditional healthcare data processing approaches that rely on centralized relational databases or monolithic data warehouses, the proposed architecture leverages decoupled object storage and distributed query execution. This design allows compute and storage resources to scale independently and supports efficient processing of high-volume, event-based healthcare data without rigid infrastructure constraints.

From a data processing perspective, the layered Bronze–Silver–Gold architecture provides clear separation of concerns. Raw data is preserved in the Bronze layer, while anonymization, schema standardization, and dimensional modeling are performed in the Silver layer. Compared to traditional schema-first approaches that require extensive upfront modeling, this method offers greater flexibility through a progressive refinement process, enabling new analytical requirements to be accommodated with minimal re-engineering. The Gold layer further materializes complex analytical logic into reusable data marts, significantly reducing query complexity and ensuring analytical consistency across dashboards and reports.

At the analytical level, the resulting Gold-layer data marts successfully transform low-level operational healthcare records into concise, policy-relevant indicators. Patient demographic distributions and disease trend metrics can be queried directly without repeatedly executing complex joins across multiple datasets. This contrasts with conventional methods where analytical logic is often embedded in ad-hoc queries, leading to performance bottlenecks and inconsistent results. By encapsulating analytical logic within the data marts, the system improves performance, reusability, and comparability of analytical outcomes.

In addition, the simulated distributed data lake environment and secured communication setup demonstrate the robustness of the proposed approach under realistic deployment constraints. By limiting direct access to sensitive data and exposing only aggregated, privacy-preserving metrics in the Gold layer, the architecture aligns with privacy-by-design principles and healthcare data protection regulations more naturally than traditional analysis workflows.

Overall, the integrated use of a distributed SQL query engine and analytical data marts establishes a scalable, flexible, and privacy-aware foundation for healthcare policy impact analysis. The results confirm that the proposed approach not only addresses technical scalability challenges but also provides clear advantages over traditional healthcare data processing methods in terms of analytical efficiency, governance, and long-term extensibility.

5.1.3 Visualization

The Dashboard is successfully created and it contains two designated charts for visualizing Patient demographics and Disease trend (common cold) problems: The Cluster Bar chart of Patient demographic and the Seasonal Heatmap chart of Disease trend (common cold)

- **Cluster Bar chart**: has age groups as x-axis and patient count as y-axis with gender as dimension. All queried patient counts are aggravated to SUM basing on their age group and gender (illustrated by the first chart from Figure 17).

- **Seasonal Heatmap chart**: has months as x-axis and year as y-axis. Both axes is sorted as ascending. All queried patient counts of common cold case are aggravated to SUM and percentage within the total case counts of their specific month-year (illustrated by the second chart from Figure 17).

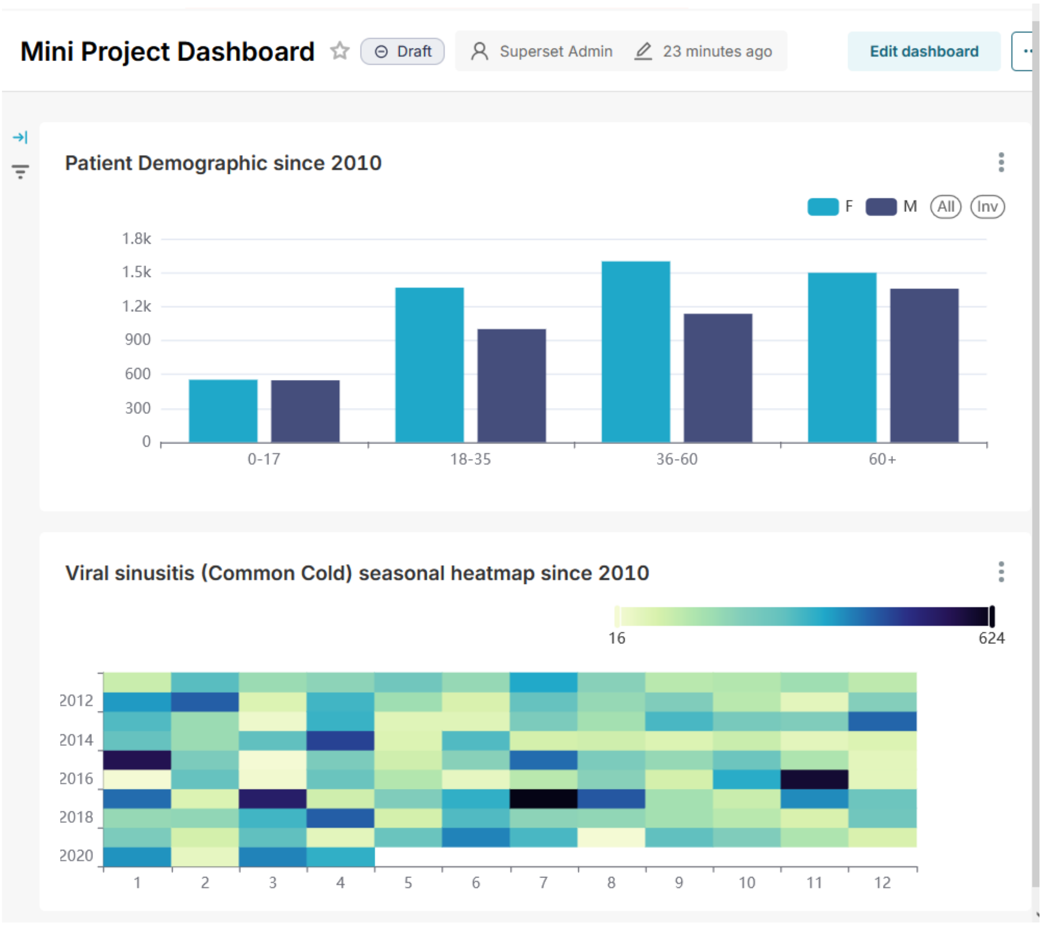


Figure 17: Dashboard result

Considering the size of the queried dataset (50 000 rows per query) used to create our charts, the Superset chart builder is able to refresh and draw our charts quickly: all of the chart is updated in parallel and delays average around 4 seconds across three refresh attempts (Superset UI framework draw time plus time to fetch result from Trino engine). The visualization performance is summarized with Table 4 below.

Chart	Dataset	Query Time
1. Cluster Bar Chart	gold.mart_patient_demographics (query 50K rows)	Fetch 1: 3.687s; Fetch 2: 4.322s; Fetch 3: 4.01s
2. Seasonal Heatmap	gold.mart_disease_trend_monthly (query 50K rows)	Fetch 1: 4.112s; Fetch 2: 3.928s; Fetch 3: 4.153s

Table 4: Visualization query performance table.

5.2 Evaluation

5.2.1 Challenges when implementing Dataset and Datasource

From the dataset and datasource perspective, several practical challenges were encountered during data preparation.

The first challenge was partitioning the original dataset into multiple independent hospital nodes while preserving meaningful data relationships. A naive split could easily break links between patients, encounters, and organizations. This issue was addressed using an organization-centric partitioning strategy, ensuring that each node represents a logically consistent hospital system.

The second challenge was simulating datasource incompatibility across nodes without destroying the underlying meaning of the data. Real healthcare systems often store the same information in different schemas, formats, and coding standards. To reflect this, a baseline schema was defined and controlled variations were applied to other nodes, such as column renaming, type changes, format differences, and medical code variations, while keeping the core semantics intact.

The third challenge involved data privacy constraints. Sensitive personal and clinical information could not be fully shared across all nodes. This was handled by masking, anonymizing, or partially suppressing sensitive attributes, while preserving essential analytical features such as age groups, aggregated locations, and clinical categories.

Overall, these solutions resulted in a realistic distributed healthcare dataset that balances data integrity, heterogeneity, and privacy, and is suitable for evaluating federated analytics using Trino.

5.2.2 Challenges when implementing Trino and Datamarts

During the implementation of Trino and analytical data marts, several challenges were encountered. These challenges mainly come from the distributed architecture, the structure of healthcare data, and data privacy requirements.

The first challenge is related to schema design and data modeling. Healthcare data is collected from different sources and often follows different schemas. Creating consistent schemas for the Silver layer required careful mapping of patients, encounters, and organizations. Incorrect mappings could cause data duplication or inaccurate analytical results, especially when joining multiple tables.

Another challenge is query performance in a distributed environment. Although Trino supports distributed query execution, queries that join large fact tables can still introduce network and coordination overhead. This issue was observed when running complex analytical queries directly on Silver-layer tables. To address this, Gold-layer data marts were created to store pre-aggregated results. However, selecting the right level of aggregation for these data marts required several iterations.

Data privacy is also a major challenge. Due to healthcare regulations, real patient data could not be used during development. As a result, synthesized datasets were used to validate the data pipeline. In addition, sensitive identifiers had to be anonymized in the Silver layer using hashing techniques while still allowing analytical joins across tables.

From an infrastructure perspective, deploying Trino in a multi-node environment added operational complexity. Proper configuration of Trino workers, object storage access, and metadata services was required. Simulating secure communication between nodes further increased setup complexity but was necessary to reflect real-world healthcare deployment conditions.

Overall, these challenges show that while Trino and data marts provide strong analytical capabilities, careful design and configuration are required to achieve stable performance, correct results, and compliance with healthcare data regulations.

6 Conclusion

Our group has successfully set up the proposed system that utilizes both fast distributed query engine - Trino and robust business intelligence platform - Apache Superset and demonstrated ability to handle varied dataset from multiple data sources in Windows ecosystem. The data was able to be queried quickly and transformed into data marts which contain meaningful insight for mass Healthcare analytic through Trino. Then, these data marts were quickly visualized with Superset in real-time and able to aid decision making for both hospitals and healthcare policy makers.

However, biggest challenges remained were the inconsistency of data schema between hospitals, the data source incompatibility from aged systems, and the inherently sensitive personal health information in patient records that demands utmost data privacy compliance from developers. These challenges represent an unavoidable information gap in gathered data and force us to devise a common standard format to store, extract and transform that is also susceptible to change in each data gathering cycle.

In the future, we aim to research and develop an automatic machine learning system that can categorize and compress schema data column of initial dataset into a same semantic bucket. Therefore, any new data schema arise to visibility will be quickly identified and correctly mapped into our common standard schema/format without manual update from developers. For data source incompatibility, we hope that contribution from the current open source community into both Trino and Superset will add greater support capability for numerous data sources, solving the problem of dealing with legacy systems.

References

[1] Apache, “GitHub - apache/superset: Apache Superset is a Data Visualization and Data Exploration Platform,” GitHub. https://github.com/apache/superset

[2] The MITRE Corporation, “Synthea Synthetic Patient Records,” Available: https://synthea.mitre.org/downloads, accessed Dec. 2025.

A Trino Table Definitions for Bronze Layer

This appendix presents the SQL definitions used to create external tables in Trino for the Bronze layer. The tables map Parquet files stored in object storage to relational schemas registered in the Hive Metastore. All definitions are included for implementation completeness and reproducibility.

A.1 Patient Data Table

```sql
CREATE TABLE IF NOT EXISTS minio.hospital_1.patients (
    id                      VARCHAR,
    birthdate               DATE,
    deathdate               DATE,
    ssn                     VARCHAR,
    drivers                 VARCHAR,
    passport                VARCHAR,
    prefix                  VARCHAR,
    first_name              VARCHAR,
    last_name               VARCHAR,
    suffix                  VARCHAR,
    maiden                  VARCHAR,
    marital                 VARCHAR,
    race                    VARCHAR,
    ethnicity               VARCHAR,
    gender                  VARCHAR,
    birthplace              VARCHAR,
    address                 VARCHAR,
    city                    VARCHAR,
    state                   VARCHAR,
    county                  VARCHAR,
    zip                     VARCHAR,
    lat                     DOUBLE,
    lon                     DOUBLE,
    healthcare_expenses     DOUBLE,
    healthcare_coverage     DOUBLE
)
WITH (
    external_location = 's3a://final-project-bigdata/bronze/
   hospital_1/patients/',
    format = 'PARQUET'
);
```

A.2 Condition Data Table

```sql
CREATE TABLE IF NOT EXISTS minio.hospital_1.conditions (
    "start"       DATE,
    "stop"        DATE,
    patient       VARCHAR,
    encounter     VARCHAR,
    code          VARCHAR,
    description   VARCHAR,
    severity      VARCHAR
)
WITH (
    external_location = 's3a://final-project-bigdata/bronze/
   hospital_1/conditions/',
    format = 'PARQUET'
);
```

A.3 Procedure Data Table

```sql
CREATE TABLE IF NOT EXISTS minio.hospital_1.procedures (
    "date"              TIMESTAMP,
    patient             VARCHAR,
    encounter           VARCHAR,
    code                VARCHAR,
    description         VARCHAR,
    base_cost           DOUBLE,
    reasoncode          VARCHAR,
    reasondescription   VARCHAR
)
WITH (
    external_location = 's3a://final-project-bigdata/bronze/
   hospital_1/procedures/',
    format = 'PARQUET'
);
```

A.4 Encounter Data Table

```sql
CREATE TABLE IF NOT EXISTS minio.hospital_1.encounters (
    id                    VARCHAR,
    "start"               TIMESTAMP,
    "stop"                TIMESTAMP,
    patient               VARCHAR,
    organization          VARCHAR,
    provider              VARCHAR,
    payer                 VARCHAR,
    encounterclass        VARCHAR,
    code                  VARCHAR,
    description             VARCHAR,
    base_encounter_cost     DOUBLE,
    total_claim_cost        DOUBLE,
    payer_coverage          DOUBLE,
    reasoncode              VARCHAR,
    reasondescription       VARCHAR
)
WITH (
    external_location = 's3a://final-project-bigdata/bronze/
   hospital_1/encounters/',
    format = 'PARQUET'
);
```

A.5 Medication Data Table

```
CREATE TABLE IF NOT EXISTS minio.hospital_1.medications (
    "start"               TIMESTAMP,
    "stop"                TIMESTAMP,
    patient               VARCHAR,
    encounter             VARCHAR,
    payer                 VARCHAR,
    code                  VARCHAR,
    description           VARCHAR,
    base_cost             DOUBLE,
    payer_coverage        DOUBLE,
    dispenses             INTEGER,
    totalcost             DOUBLE,
    reasoncode            VARCHAR,
    reasondescription     VARCHAR
)
WITH (
    external_location = 's3a://final-project-bigdata/bronze/
   hospital_1/medications/',
    format = 'PARQUET'
);
```

A.6 Organization Data Table

```
CREATE TABLE IF NOT EXISTS minio.hospital_1.organizations (
    id            VARCHAR,
    name          VARCHAR,
    address       VARCHAR,
    city          VARCHAR,
    state         VARCHAR,
    zip           VARCHAR,
    lat             DOUBLE,
    lon             DOUBLE,
    phone           VARCHAR,
    revenue         DOUBLE,
    utilization     DOUBLE
)
WITH (
    external_location = 's3a://final-project-bigdata/bronze/
   hospital_1/organizations/',
    format = 'PARQUET'
);
```

A Silver Layer SQL (Schema, Dimensional Tables, and Load Queries)

This appendix contains the SQL statements used to create the Silver schema, define dimension/fact tables, and populate them from the Bronze layer (initially using `hospital_8` as the source system). All tables are stored in Parquet format on object storage and registered for querying via Trino.

A.1 Create Silver schema

```sql
CREATE SCHEMA IF NOT EXISTS minio.silver
WITH (
    location = 's3a://final-project-bigdata/silver/'
);
```

A.2 Create dimension tables

A.2.1 dim_patient

```sql
CREATE TABLE IF NOT EXISTS minio.silver.dim_patient (
    patient_key       BIGINT,
    patient_hash      VARCHAR,
    gender            VARCHAR,
    birthdate         DATE,
    age               INTEGER,
    age_group         VARCHAR,
    race              VARCHAR,
    ethnicity         VARCHAR,
    city              VARCHAR,
    state             VARCHAR,
    county            VARCHAR,
    lat               DOUBLE,
    lon               DOUBLE,
    source_system     VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
   dim_patient/'
);
```

A.2.2 dim_organization

```
CREATE TABLE IF NOT EXISTS minio.silver.dim_organization (
    org_key          BIGINT,
    org_id_src       VARCHAR, s
    org_name         VARCHAR,
    city             VARCHAR,
    state            VARCHAR,
    zip              VARCHAR,
    lat              DOUBLE,
    lon              DOUBLE,
    revenue          DOUBLE,
    utilization      DOUBLE,
    source_system    VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/dim_organization/'
);
```

A.2.3 dim_date

```
CREATE TABLE IF NOT EXISTS minio.silver.dim_date (
    date_key      INTEGER,
    "date"        DATE,
    day           INTEGER,
    month         INTEGER,
    year          INTEGER,
    quarter       INTEGER,
    day_of_week   INTEGER,
    is_weekend    BOOLEAN
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/dim_date/'
);
```

A.2.4 dim_condition_code

```
CREATE TABLE IF NOT EXISTS minio.silver.dim_condition_code (
    condition_code    VARCHAR,
    condition_name    VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
   dim_condition_code/'
);
```

A.2.5 dim_procedure_code

```
CREATE TABLE IF NOT EXISTS minio.silver.dim_procedure_code(
    procedure_code    VARCHAR,
    procedure_name    VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
   dim_procedure_code/'
);
```

A.2.6 dim_medication_code

```
CREATE TABLE IF NOT EXISTS minio.silver.dim_medication_code (
    medication_code    VARCHAR,
    medication_name    VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
   dim_medication_code/'
);
```

A.3 Create fact tables

A.3.1 fact_encounter

```
CREATE TABLE IF NOT EXISTS minio.silver.fact_encounter (
    encounter_key          BIGINT,
    encounter_id_src       VARCHAR,
    patient_hash           VARCHAR,
    org_key                BIGINT,
    start_date_key         INTEGER,
    stop_date_key          INTEGER,
    encounterclass         VARCHAR,
    base_encounter_cost    DOUBLE,
    total_claim_cost       DOUBLE,
    payer_coverage         DOUBLE,
    payer                  VARCHAR,
    reason_code             VARCHAR,
    reason_desc             VARCHAR,
    source_system           VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
    fact_encounter/'
);
```

A.3.2 fact_condition

```
CREATE TABLE IF NOT EXISTS minio.silver.fact_condition (
    encounter_id_src     VARCHAR,
    patient_hash         VARCHAR,
    condition_code       VARCHAR,
    start_date_key       INTEGER,
    stop_date_key        INTEGER,
    severity             VARCHAR,
    source_system        VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
    fact_condition/'
);
```

A.3.3 fact_procedure

```
CREATE TABLE IF NOT EXISTS minio.silver.fact_procedure (
    encounter_id_src     VARCHAR,
    patient_hash         VARCHAR,
    procedure_code       VARCHAR,
    date_key             INTEGER,
    base_cost            DOUBLE,
    reason_code          VARCHAR,
    reason_desc          VARCHAR,
    source_system        VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
    fact_procedure/'
);
```

A.3.4 fact_medication

```
CREATE TABLE IF NOT EXISTS minio.silver.fact_medication (
    encounter_id_src      VARCHAR,
    patient_hash          VARCHAR,
    medication_code       VARCHAR,     -- join dim_medication_code
    start_date_key        INTEGER,
    stop_date_key         INTEGER,
    base_cost             DOUBLE,
    payer_coverage        DOUBLE,
    total_cost            DOUBLE,
    dispenses             INTEGER,
    payer                 VARCHAR,
    reason_code           VARCHAR,
    reason_desc           VARCHAR,
    source_system         VARCHAR
)
WITH (
    format = 'PARQUET',
    external_location = 's3a://final-project-bigdata/silver/
   fact_medication/'
);
```

A.4 Load data into dimensions

A.4.1 Insert into dim_patient (source: hospital_8)

```
INSERT INTO minio.silver.dim_patient
SELECT
    row_number() OVER (ORDER BY id)               AS patient_key,
    to_hex(md5(to_utf8(id)))                      AS patient_hash,
    gender,
    birthdate,
    year(current_date) - year(birthdate)          AS age,
    CASE
        WHEN year(current_date) - year(birthdate) < 18 THEN
   '0-17'
        WHEN year(current_date) - year(birthdate) BETWEEN 18 AND
    35 THEN '18-35'
        WHEN year(current_date) - year(birthdate) BETWEEN 36 AND
    60 THEN '36-60'
        ELSE '60+'
    END                                           AS age_group,
    race,
    ethnicity,
    city,
    state,
    county,
    lat,
    lon,
    'hospital_8'                                   AS source_system
FROM minio.hospital_8.patients;
```

A.4.2 Insert into dim_date (fixed range example)

```
INSERT INTO minio.silver.dim_date
WITH dates AS (
    SELECT x AS dt
    FROM UNNEST(
        sequence(date '1960-01-01', date '1980-12-31', interval
    '1' day)
    ) AS t(x)
)
SELECT
    CAST(format_datetime(dt, 'yyyyMMdd') AS INTEGER) AS date_key
    ,
    dt                                                AS "date",
    day_of_month(dt)                                 AS day,
    month(dt)                                        AS month,
    year(dt)                                         AS year,
    quarter(dt)                                      AS quarter,
    day_of_week(dt)                                  AS
    day_of_week,
    day_of_week(dt) IN (6, 7)                        AS
    is_weekend;
```

A.4.3 Insert into dim_date (dynamic bounds from encounters)

```
INSERT INTO minio.silver.dim_date
WITH bounds AS (
    SELECT
        date(min("start")) AS min_date,
        date(max("stop"))  AS max_date
    FROM minio.hospital_8.encounters
),
dates AS (
    SELECT sequence(min_date, max_date, interval '1' day) AS d
    FROM bounds
)
SELECT
    CAST(format_datetime(x, 'yyyyMMdd') AS INTEGER) AS date_key,
    x                                               AS "date",
    day_of_month(x)                                 AS day,
    month(x)                                        AS month,
    year(x)                                         AS year,
    quarter(x)                                      AS quarter,
    day_of_week(x)                                  AS
   day_of_week,
    day_of_week(x) IN (6, 7)                        AS is_weekend
FROM dates
CROSS JOIN UNNEST(d) AS t(x);
```

A.4.4 Insert into dim_organization (source: hospital_8)

```
INSERT INTO minio.silver.dim_organization
SELECT
    row_number() OVER (ORDER BY id)   AS org_key,
    id                                AS org_id_src,
    name                              AS org_name,
    city,
    state,
    zip,
    lat,
    lon,
    revenue,
    utilization,
    'hospital_8'                      AS source_system
FROM minio.hospital_8.organizations;
```

A.4.5 Insert into dim_condition_code (source: hospital_8)

```
INSERT INTO minio.silver.dim_condition_code
SELECT DISTINCT
    code           AS condition_code,
    description    AS condition_name
FROM minio.hospital_8.conditions;
```

A.4.6 Insert into dim_procedure_code (source: hospital_8)

```
INSERT INTO minio.silver.dim_procedure_code
SELECT DISTINCT
    code           AS procedure_code,
    description    AS procedure_name
FROM minio.hospital_8.procedures;
```

A.4.7 Insert into dim_medication_code (source: hospital_8)

```
INSERT INTO minio.silver.dim_medication_code
SELECT DISTINCT
    code          AS medication_code,
    description   AS medication_name
FROM minio.hospital_8.medications;
```

A.5 Load data into facts

A.5.1 Insert into fact_encounter (source: hospital_8)

```
INSERT INTO minio.silver.fact_encounter
SELECT
    row_number() OVER (ORDER BY e.id)
      AS encounter_key,
    e.id
      AS encounter_id_src,
    to_hex(md5(to_utf8(e.patient)))
     AS patient_hash,
    o.org_key,
    cast(format_datetime(e.start, 'yyyyMMdd') AS INTEGER)
      AS start_date_key,
    cast(format_datetime(e.stop,  'yyyyMMdd') AS INTEGER)
      AS stop_date_key,
    e.encounterclass,
    e.base_encounter_cost,
    e.total_claim_cost,
    e.payer_coverage,
    e.payer,
    e.reasoncode,
    e.reasondescription,
    'hospital_8'
      AS source_system
FROM minio.hospital_8.encounters e
LEFT JOIN minio.silver.dim_organization o
  ON o.org_id_src = e.organization;
```

A.5.2 Insert into fact_condition (source: hospital_8)

```
INSERT INTO minio.silver.fact_condition
SELECT
    c.encounter              AS encounter_id_src,
    to_hex(md5(to_utf8(c.patient)))             AS patient_hash
   ,
    c.code                  AS condition_code,
    CAST(format_datetime(c.start, 'yyyyMMdd') AS INTEGER)  AS
   start_date_key,
    CAST(format_datetime(c.stop,  'yyyyMMdd') AS INTEGER)  AS
   stop_date_key,
    c.severity,
    'hospital_8'            AS source_system
FROM minio.hospital_8.conditions c;
```

A.5.3 Insert into fact_procedure (source: hospital_8)

```
INSERT INTO minio.silver.fact_procedure
SELECT
    p.encounter                          AS encounter_id_src,
    to_hex(md5(to_utf8(p.patient)))      AS patient_hash,
    p.code                               AS procedure_code,
    CAST(format_datetime(p.date, 'yyyyMMdd') AS INTEGER)  AS
   date_key,
    p.base_cost,
    p.reasoncode,
    p.reasondescription,
    'hospital_8'                         AS source_system
FROM minio.hospital_8.procedures p;
```

A.5.4 Insert into fact_medication (source: hospital_8)

```
INSERT INTO minio.silver.fact_medication
SELECT
    m.encounter                          AS encounter_id_src,
    to_hex(md5(to_utf8(m.patient)))      AS patient_hash,
    m.code                               AS medication_code,
    CAST(format_datetime(m.start, 'yyyyMMdd') AS INTEGER) AS
   start_date_key,
    CAST(format_datetime(m.stop,  'yyyyMMdd') AS INTEGER) AS
   stop_date_key,
    m.base_cost,
    m.payer_coverage,
    m.totalcost,
    m.dispenses,
    m.payer,
    m.reasoncode,
    m.reasondescription,
    'hospital_8'                         AS source_system
FROM minio.hospital_8.medications m;
```