Healthcare Data Warehouse Project

Group Members

Skander BENKHODJA
Nooraldeen Mustafa Abbas AL SUDANI
Radja BELARBI

Class: Big Data Management
Instructor: Dr. E.Simona ROMBO
Institution: University of Palermo UNIPA
Date: June 20, 2025

Group Member Contributions

Throughout the project, all team members stayed consistently informed about each other's progress and tasks. While the work was complementary and divided, collaboration and communication were maintained to ensure coherence and consistency in all deliverables.

It is worth noting that Radja BELARBI was the only member who successfully managed to install Tableau Desktop on her system. As a result, she took the lead on the visualization process. However, all members participated in defining the visualization requirements and contributed to the creation of the views.

- **Nooraldeen Mustafa Abbas AL SUDANI**: Responsible for database modeling and implementation. Also contributed to the decisional queries engineering, datamarts modeling (attribute tree, fact schema, and star schema).
- **Skander BENKHODJA**: Responsible for dataset cleaning, modeling, and led the ETL process implementation. Also contributed to the decisional queries engineering, data warehouse modeling, and design.
- **Radja BELARBI**: Contributed to all parts of the project.

Table of Contents

Group Member Contributions **1**

I Overview **4**
1.1 Project Details 4

II Data Preprocessing **5**
2.1 Handling Empty Columns 5
2.2 Removing Redundant Columns 5
2.3 Handling Special Characters 5
2.4 Handling NaN Values 6
2.5 Column Count Comparison 6
2.6 Data Type Formatting 7

III Database Implementation **7**
3.1 Database Schema Design 7
3.1.1 Schema Overview 8
3.1.2 Normalization 8
3.2 Entity-Relationship Diagram 8
3.3 Schema Analysis and Key Descriptions 9
3.3.1 Cardinality Matrix and Entity Relationships 9
3.3.2 Identifier Strategy and Referential Integrity 10
3.4 Database Setup and Environment 11
3.4.1 MySQL Server Setup 11
3.4.2 MySQL Workbench Setup 11
3.4.3 Reasons for Choosing MySQL 11
3.4.4 Database Schema Initialization 12
3.4.5 Table Creation 12
3.4.6 Explanation of Keys 14
3.4.7 Bulk Data Loading 15

IV Data Warehouse Design and Implementation **15**
4.1 Decisional Queries 16
4.2 Datamart Attribute Tree, Fact and Star Schema 17
4.2.1 Encounters Datamart 17
4.2.2 Claim Transactions Datamart 19
4.2.3 Medications Datamart 21
4.2.4 Observations Datamart 23
4.2.5 Procedures Datamart 25
4.2.6 Immunizations Datamart 27

V Datamart Implementation and Analytical Interface **29**
5.1 Implementation of SQL Views for Datamarts 29
5.1.1 Encounters Datamart 30
5.1.2 Claim Transactions Datamart 30
5.1.3 Observations Datamart 31
5.1.4 Medications Datamart 31

5.1.5 Procedures Datamart . . . 32
5.1.6 Immunizations Datamart . . . 32
5.2 Analytical Exploration with Tableau Desktop . . . 32
5.2.1 Encounters Datamart Visualizations . . . 33
5.2.2 Query 1: Number of Encounters per Organization per Quarter (2020–2021) . . . 33
5.2.3 Query 2: Most Common Encounter Classes (Females under 40) . . . 34
5.2.4 Query 3: Location of young women seeking medical care during the current year(2021) . . . 35
5.2.5 Claim Transactions Datamart Visualizations . . . 36
5.2.6 Query 1: Total and Average Transaction Amount per Claim Type and Provider by Month (2021) . . . 36
5.2.7 Query 2: 2021 Patient Coverage Analysis by Age Segmentr . . . 38
5.2.8 Observations Datamart Visualizations . . . 39
5.2.9 Query 1: Count of Adult Patients with Elevated Body Weight in 2021, Grouped by Gender . . . 39
5.2.10 Query 2: Average Glucose Distribution over different patients in 2021r . . . 42
5.2.11 Medications Datamart Visualizations . . . 42
5.2.12 Query 1: Average Base Cost of Medications Prescribed to Patients Over 65 in 2021, Grouped by Payer Name . . . 43
5.2.13 Query 2: Top Prescribed Medications: Average Dispenses by Gender and Ethnicity (2021) . . . 43
5.2.14 Procedures Datamart Visualizations . . . 44
5.2.15 Query: Number of Procedures Performed in 2021 on Female Patients Under 30, Grouped by Procedure Description . . . 44
5.2.16 Immunizations Datamart Visualizations . . . 45
5.2.17 Query: Percentage of Patients Aged 65+ Receiving an Influenza Immunization in Winter (2011-2021), and Average Cost . . . 45

VI Discussion and Conclusion **46**
6.1 Challenges and Limitations . . . 46
6.2 Conclusion . . . 47

I Overview

This project presents the end-to-end development of a data warehouse for healthcare analytics starting from implementing the database using the Synthea synthetic patient dataset. It includes data modeling, database implementation, OLAP queries, and interactive dashboards.

1.1 Project Details

- **Tools:** MySQL, Python, Tableau Desktop
- **Domain:** Healthcare / Electronic Health Records (EHR)
- **Goal:** Enable decision-making insights for hospital administration and patient management.
- **Dataset:** Synthea Synthetic Patient Records
- **Source:** `https://synthea.mitre.org/downloads`
- **Format:** Multiple CSV files (relational)
- **Volume:** 1,684,399 records across 18 tables
- **Main Entities:** Patients, Encounters, Claim_transactions, Conditions, Observations, Medications, Providers
- **Use Case:** Simulates realistic healthcare records for building and testing clinical data warehouses and analytics systems.

Table Name	Description	Rows Count
Allergies	Patient allergy records	794
Careplans	Assigned treatment plans	3931
Claims	Insurance claim submissions	117,889
Claim_transactions	Payments and claim processing details	711,238
Conditions	Diagnosed health conditions	38,094
Devices	Medical devices used or assigned	89
Encounters	Patient-provider interactions	61,459
Imaging_studies	Imaging results like X-rays and MRIs	920
Immunizations	Vaccinations administered	17,009
Medications	Medications given or prescribed	56,430
Observations	Clinical observations and vitals	531,144
Organizations	Healthcare organizations and facilities	1,127
Patients	Basic patient demographic information	1,163
Payer_transitions	Insurance provider changes	53,101
Payers	Health insurance providers	10
Procedures	Performed medical procedures	83,823
Providers	Healthcare professionals and staff	5,056
Supplies	Medical items and consumables used	1,573

Table 1: Database Primary tables and size

II Data Preprocessing

Before loading the data into the database, the raw CSV files were cleaned to remove any inconsistencies, handle missing data, and ensure the integrity of the dataset. This step was necessary to make sure that the data would conform to the expected schema and perform optimally in the database.

2.1 Handling Empty Columns

As part of the data cleaning process, columns with no data (i.e., columns entirely composed of missing values) were identified and removed. These columns do not contribute any information to the analysis and retaining them would only increase memory usage and noise in the dataset. Removing them ensures a cleaner and more efficient data structure. Here is the Python code used for this step.

```
import pandas as pd

df = pd.read_csv('.../Path/to/dataset_file.csv')
empty_columns = df.columns[df.isnull().all()].tolist()
print("Columns␣that␣are␣completely␣empty:", empty_columns)
df_cleaned = df.dropna(axis=1, how='all')
df_cleaned.to_csv('.../Path/to/dataset_file.csv', index=False)
print(f"Shape␣of␣cleaned␣data:␣{df_cleaned.shape}")
```

2.2 Removing Redundant Columns

To improve dataset efficiency and reduce noise, columns that carried the same value across all rows were identified and removed. These redundant columns do not contribute to data variability or decision-making and only add unnecessary storage overhead.

```
df = pd.read_csv('.../Path/to/dataset_file.csv', dtype=str)
redundant_cols = [col for col in df.columns if df[col].nunique() == 1]
df.drop(columns=redundant_cols, inplace=True)
print("Removed␣columns:", redundant_cols)
```

2.3 Handling Special Characters

Special characters were standardized during preprocessing to ensure consistency and compatibility. Symbols like 'µ' were replaced with their textual equivalents, such as 'u', to prevent encoding issues, ease data loading during database implementation, and support seamless data processing and analysis.

```python
import pandas as pd

# Sample function to normalize text (just as an example)
def normalize_text(value):
    if isinstance(value, str):
        return value.lower().strip()
    return value
# Load your dataset
df = pd.read_csv('.../Path/to/dataset_file.csv', encoding='ISO-8859-1', dtype=str)

# Apply the normalization function to each column using map (on Series)
for column in df.columns:
    df[column] = df[column].map(normalize_text)
# Continue with cleaning process...
# Mapping of confusing or special characters to their replacements
special_char_map = {
    'µ': 'u', '£': 'GBP', '¤': 'o','ƒ': 'f',
    'ı': 'i', 'ﬁ': 'fi','ﬂ': 'fl',
    '¨': '',     # remove diaeresis
    '·': '.',    # middle dot to period
    '¬': '-',    # logical not to dash
    '“': '"', '”': '"', '': "'", '': "'", '': '-',    # en dash
    '—': '-',    # em dash
    '…': '...',  # ellipsis
    'ø': 'o','Ø': 'O','æ': 'ae',  'Æ': 'AE','œ': 'oe','Œ': 'OE','ç': 'c', 'Ç': 'C', 'ñ': 'n','Ñ': 'N'
}
# Function to normalize string values
def normalize_text(text):
    if isinstance(text, str):
        for char, replacement in special_char_map.items():
            text = text.replace(char, replacement)
    return text
# Apply normalization to entire DataFrame
df_cleaned = df.map(normalize_text)
# Save the cleaned CSV
df_cleaned.to_csv(".../Path/to/dataset_file.csv", index=False)
```

Figure 1: Python Code for Cleaning Special Characters and Formatting

2.4 Handling NaN Values

After loading the data into the database, many columns with numeric values were defined as VARCHAR due to missing values represented as empty strings. MySQL does not accept numeric columns with empty string values, so these columns were converted to VARCHAR. However, since these VARCHAR columns did not contribute to any analytical or decisional queries, this choice had no impact on the overall analysis. Declaring them as VARCHAR allowed us to maintain data integrity and avoid errors during processing, ensuring smooth handling of missing values while preserving the essential structure of the dataset.

2.5 Column Count Comparison

Table Name	Before	After
Allergies	14	13
Claims	30	25
Claim_transactions	29	22
Organizations	11	9

Patients	25	24
Providers	12	11

Table 2: tables size comparison after removing redundancy

2.6 Data Type Formatting

To ensure consistency and enable accurate querying and analysis, the dataset underwent data type formatting involving DATE, DATETIME, INT, and DOUBLE types. For example, the encounters table includes columns that store both date and time information, which were formatted to the DATETIME type (YYYY-MM-DD HH:MM:SS). Other tables like patients contain birth dates and other calendar-based fields that were cast to the DATE type (YYYY-MM-DD). Additionally, several numeric columns—such as codes—were initially stored in scientific notation (e.g., 1.0e+10) and were formatted into standard INT or DOUBLE values as appropriate. This step was essential for ensuring compatibility with MySQL and analytical tools like Tableau, avoiding type mismatches and enabling correct filtering, aggregation, and visualization.

```
import numpy as np

df = pd.read_csv(".../Path/to/dataset_file.csv")
datetime_cols = ['START', 'STOP']
for col in datetime_cols:
    df[col] = pd.to_datetime(df[col], format='%Y-%m-%d␣%H:%M:%S', errors
       ↪ ='coerce')

date_cols = ['BIRTHDATE', 'DEATHDATE']
for col in date_cols:
    df[col] = pd.to_datetime(df[col], format='%Y-%m-%d', errors='coerce'
       ↪ ).dt.date

sci_cols = ['CODE', 'PROCEDURECODE']
for col in sci_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')
    if df[col].dropna().apply(float.is_integer).all():
        df[col] = df[col].astype('Int64')

df.to_csv(".../Path/to/dataset_file.csv")
```

III Database Implementation

3.1 Database Schema Design

The database schema for this project is designed to effectively handle data for an EHR system. It consists of several core entities, including **Patients**, **Encounters**, **Claim_transactions**, and **procedures**, which are interconnected to represent the relationships within the system.

3.1.1 Schema Overview

The main tables in the schema are:

- **Patients**: Stores personal information about patients, including demographic details, contact information, and unique patient identifiers.
- **Encounters**: Contains records of patient visits or interactions with healthcare providers, including visit dates, diagnoses, and treatment details.
- **Claim_Transactions**: Records financial transactions related to claims, including claim IDs, amounts billed, and claim status.
- **Procedures**: Captures information about medical procedures performed during encounters, linking procedures to specific visits and patients.
- **Medications**: Stores information about prescribed medications for patients, including dosage, prescribing physician, and treatment periods.
- **Payers**: Contains details about patient insurance plans and coverage, including payer/provider information, `payer_coverage` and covered encounters number.

These tables are related to each other via foreign keys, ensuring data integrity and consistency across the database. The full structure and relationships can be seen in the Entity-Relationship Diagram (ERD) below.

3.1.2 Normalization

Normalization was performed to ensure data integrity and to reduce redundancy. The schema was normalized up to the Third Normal Form (3NF), which is explained below:

- **1NF**: All tables contain atomic values, meaning that each column holds a single value and no multi-valued attributes are present.
- **2NF**: Each table is free from partial dependencies, where all non-key attributes are fully functionally dependent on the primary key.
- **3NF**: Transitive dependencies were removed, meaning that non-key attributes depend only on the primary key, and not on other non-key attributes.

This normalization process ensures that the database structure minimizes redundancy and supports efficient query performance

3.2 Entity-Relationship Diagram

The ER diagram was designed in Lucidchart and adheres to third normal form (3NF), with clearly mapped relationships between core entities like patients and claims.
Although some objects in the database are typically modeled as relationship tables due to the presence of many-to-many associations between entities (such as `payer_transition` and `claims_transactions`, or encounters and procedures), in the context of this project — which focuses on the implementation of a data warehouse and the design of corresponding datamarts — all objects were treated as independent entities. This approach was adopted because each table represents a meaningful source of information that can

support specific analytical needs and decision-making processes. By treating every object as an entity, we ensure greater flexibility in constructing datamarts tailored to various business questions, allowing each object to serve as a potential fact or dimension depending on the analytical context.
The Crow's Foot model was used to visually represent the schema and simplify its complexity, making relationships between entities more interpretable during the design phase.

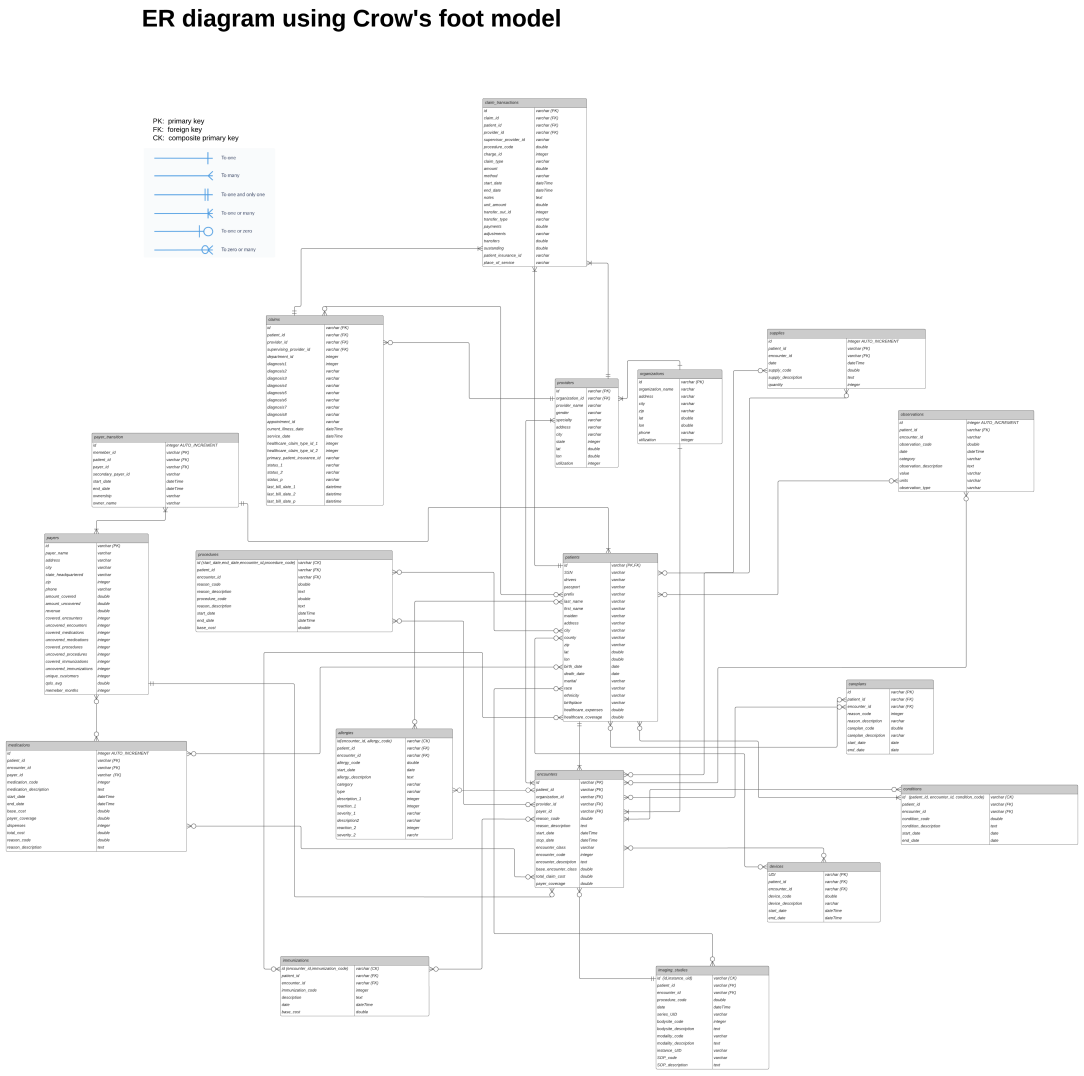


Figure 2: Entity-Relationship Diagram for Healthcare Data Warehouse

3.3 Schema Analysis and Key Descriptions

3.3.1 Cardinality Matrix and Entity Relationships

The following matrix summarizes the cardinality relationships between key entities identified in the ER diagram. Each cell indicates the nature of the relationship between the row and column entities, using standard notation (1:1, 1:N, M:N, or "—" for no direct relationship). This provides a compact view of how entities are interconnected in the conceptual schema and supports further database normalization and foreign key assignments.

	patients	claims	encounters	providers	medications	allergies	observations	procedures	immunizations	organizations
patients	—	1:N	1:N	—	M:N	M:N	M:N	M:N	M:N	—
claims	N:1	—	—	—	—	—	—	—	—	—
encounters	N:1	—	—	N:1	1:N	M:N	M:N	M:N	—	—
providers	—	—	1:N	—	—	—	—	—	—	N:1
medications	N:M	—	N:1	—	M:N	—	—	—	—	—
allergies	N:M	—	N:M	—	—	—	—	—	—	—
observations	N:M	—	N:M	—	—	—	—	—	—	—
procedures	N:M	—	N:M	—	—	—	—	—	—	—
immunizations	N:M	—	N:M	—	—	—	—	—	—	—
organizations	—	—	1:N	1:N	—	—	—	—	—	—

Table 3: Cardinality Matrix summarizing some entity relationships

3.3.2 Identifier Strategy and Referential Integrity

Each table in the schema was assigned a primary key to uniquely identify its records. In cases where no natural key existed or multiple attributes were required to ensure uniqueness, a surrogate key (such as an auto-incremented ID) was introduced for simplicity and performance.

Foreign keys were defined to maintain referential integrity between related tables, ensuring consistent relationships across the database. These include links such as `patient_id` in the encounters table referencing the patients table, and `claim_id` in `claim_insurance_details` referencing the claims table.

A few relationships required the use of composite keys, particularly in associative or junction tables, to uniquely identify records involving multiple foreign keys. These were implemented to capture many-to-many relationships accurately without introducing redundancy.

Table	**Primary Key(s)**	**Foreign Key(s)**	**Key Source**
patients	id	–	Manually Defined
patient_identity	id	patient_id	Manually Defined
encounters	id	patient_id, organization_id, provider_id, payer_id	Manually Defined
allergies	encounter_id, allergy_code	patient_id, encounter_id	Manually Defined
organizations	id	–	Manually Defined
providers	id	organization_id	Manually Defined
payers	id	–	Manually Defined
payer_transitions	id	patient_id, payer_id	Auto-Generated
careplans	id	patient_id, encounter_id	Manually Defined
claims	id	patient_id, provider_id, supervising_provider_id	Manually Defined
claim_insurance_details	claim_id	claim_id	Manually Defined
claim_transaction	id	claim_id, patient_id	Manually Defined
conditions	patient_id, encounter_id, condition_code	patient_id, encounter_id	Manually Defined
devices	udi	patient_id, encounter_id	Manually Defined
procedures	start_date, end_date, encounter_id, procedure_code	patient_id, encounter_id	Manually Defined
imaging_studies	id, instance_uid	patient_id, encounter_id	Manually Defined
immunizations	encounter_id, immunization_code	patient_id, encounter_id	Manually Defined
medications	id	patient_id, encounter_id, payer_id	Auto-Generated
observations	id	patient_id, encounter_id	Auto-Generated
supplies	id	patient_id	Auto-Generated

Table 4: Summary of Keys per Table

3.4 Database Setup and Environment

This section describes the setup of the database environment, the tools used for database management, and the reasoning behind selecting MySQL as the database management system for this project.

3.4.1 MySQL Server Setup

For this project, **MySQL Server** was selected as the relational database management system (RDBMS) due to its reliability, scalability, and widespread use in both academic and industry environments. The version of **MySQL Server** used is `9.3.0`.

MySQL was installed on a local machine. The server configuration was adjusted to support both local and remote connections, facilitating easy access to the database from MySQL Workbench and other tools.

Some key configuration steps include:

- **Max Connections**: The maximum number of simultaneous connections was adjusted to optimize server performance.
- **Buffer Pool Size**: The InnoDB buffer pool size was tuned for better memory utilization, especially when dealing with large datasets.
- **Data Directory**: The data directory path was set to ensure efficient data storage and management.

3.4.2 MySQL Workbench Setup

To interact with MySQL Server and manage the database schema and queries, **MySQL Workbench** was used. This tool provides an integrated environment for database design, query writing, and server management.

The version of **MySQL Workbench** used is `9.3`. In MySQL Workbench, the following tasks were performed:

- **Schema Creation**: The database schema and tables were created using both the graphical interface and SQL scripts.
- **Query Execution**: SQL queries were written and executed for data manipulation, table creation, index generation, and complex SELECT statements.

3.4.3 Reasons for Choosing MySQL

- **Scalability**: MySQL is a highly scalable RDBMS, capable of efficiently handling large datasets.
- **ACID Compliance**: MySQL ensures that data operations are atomic, consistent, isolated, and durable, ensuring data integrity in critical applications.
- **Community Support**: As an open-source RDBMS, MySQL has a vast community of developers and comprehensive documentation, making it easy to resolve issues and improve the system.

3.4.4 Database Schema Initialization

Before defining the in dividual tables, a dedicated database schema named `synthea_db` was created to organize all the entities relevant to the Electronic Health Records (EHR) system.

```
-- Create the database schema
CREATE DATABASE synthea_db;

-- Use the schema
USE synthea_db;
```

This schema serves as the central container for all subsequent tables such as `Patients`, `Encounters`, `Claim_Transactions`, `Procedures`, and `Payers`. Organizing the tables under a dedicated schema improves manageability and data isolation.

3.4.5 Table Creation

The EHR database contains several tables, each structured to store specific information. Below are the SQL statements used to create the core tables in the database:

1. Patients Table

The `Patients` table stores personal information about patients. It includes fields such as patient ID, first name, last name, date of birth and healthcare coverage.

```
CREATE TABLE patients (
  id VARCHAR(45) PRIMARY KEY,
  ssn VARCHAR(11),
  drivers VARCHAR(9) DEFAULT NULL,
  passport VARCHAR(10) DEFAULT NULL,
  prefix VARCHAR(5) DEFAULT NULL,
  first_name VARCHAR(25),
  last_name VARCHAR(25),
  suffix VARCHAR(7),
  maiden VARCHAR(25) DEFAULT NULL,
  gender VARCHAR(1),
  address VARCHAR(50),
  city VARCHAR(30),
  county VARCHAR(30),
  zip VARCHAR(15) DEFAULT NULL,
  lat DOUBLE,
  lon DOUBLE,
  birth_date DATE DEFAULT NULL,
  death_date VARCHAR(10) DEFAULT NULL,
  marital VARCHAR(5) DEFAULT NULL,
  race VARCHAR(15),
  ethnicity VARCHAR(15),
  birth_place VARCHAR(100),
  healthcare_expenses DOUBLE,
  healthcare_coverage DOUBLE
);
```

2. Encounters Table

The `Encounters` table stores records of patient visits to healthcare providers. It includes information about the date of the visit, diagnosis, and the provider seen by the patient.

```
CREATE TABLE encounters (
```

```
  id VARCHAR(45) PRIMARY KEY,
  start_date DATETIME DEFAULT NULL,
  stop_date DATETIME DEFAULT NULL,
  patient_id VARCHAR(45),
  organization_id VARCHAR(45),
  provider_id VARCHAR(45),
  payer_id VARCHAR(45),
  encounter_class VARCHAR(15) DEFAULT NULL,
  encounter_code INT DEFAULT NULL,
  encounter_desc TEXT DEFAULT NULL,
  base_encounter_class DOUBLE DEFAULT NULL,
  total_claim_cost DOUBLE DEFAULT NULL,
  payer_coverage DOUBLE DEFAULT NULL,
  reason_code VARCHAR(20) DEFAULT NULL,
  reason_desc TEXT,
  FOREIGN KEY (patient_id) REFERENCES patients(id) ,
  FOREIGN KEY (organization_id) REFERENCES organizations(id) ,
  FOREIGN KEY (provider_id) REFERENCES providers(id) ,
  FOREIGN KEY (payer_id) REFERENCES payers(id)

);
```

3. Claim Transactions Table

The `Claim_Transactions` table captures financial transactions related to claims. It stores claim IDs, billed amounts, and claim status.

```
CREATE TABLE claim_transaction(
id VARCHAR(45) PRIMARY KEY,
claim_id VARCHAR(45),
charge_id INT,
patient_id VARCHAR(45),
claim_type VARCHAR(20),
amount VARCHAR(45) DEFAULT NULL,
method VARCHAR(20) DEFAULT NULL,
start_date DATETIME,
end_date DATETIME,
service_place VARCHAR(45),
procedure_code DOUBLE,
notes TEXT DEFAULT NULL,
unit_amount VARCHAR(45) DEFAULT NULL,
transfer_out_id VARCHAR(45) DEFAULT NULL,
transfer_type VARCHAR(10) DEFAULT NULL,
payments VARCHAR(45) DEFAULT NULL,
adjustments VARCHAR(45) DEFAULT NULL,
transfers VARCHAR(45) DEFAULT NULL,
outstanding VARCHAR(45) DEFAULT NULL,
patient_insurance_id varchar(45) DEFAULT NULL,
provider_id VARCHAR(45),
supervisor_provider_id VARCHAR(45),
FOREIGN KEY (claim_id) REFERENCES claims(id),
FOREIGN KEY (patient_id) REFERENCES patients(id),
FOREIGN KEY (provider_id) REFERENCES providers(id)

);
```

4. Procedures Table

The `Procedures` table stores information about medical procedures performed during encounters. It links procedures to specific patients and encounters.

```
CREATE TABLE procedures(
start_date DATETIME,
end_date DATETIME,
patient_id VARCHAR(45),
encounter_id VARCHAR(45),
procedure_code DOUBLE,
procedure_desc TEXT,
base_cost DOUBLE,
reason_code VARCHAR(30) DEFAULT NULL,
reason_desc TEXT DEFAULT NULL,
PRIMARY KEY (start_date,end_date,encounter_id,procedure_code),
FOREIGN KEY (patient_id) REFERENCES patients(id),
FOREIGN KEY (encounter_id) REFERENCES encounters(id)

);
```

5. Payers Table

The `Payers` table contains details about patient insurance plans, including payer information and coverage.

```
CREATE TABLE payers(
id VARCHAR(45) PRIMARY KEY,
payer_name VARCHAR(30),
address VARCHAR(30) DEFAULT NULL,
city VARCHAR(30) DEFAULT NULL,
state_headquartered VARCHAR(5) DEFAULT NULL,
zip VARCHAR(15) DEFAULT NULL,
phone VARCHAR(15) DEFAULT NULL,
amount_covered DOUBLE,
amount_uncovered DOUBLE,
revenue DOUBLE,
covered_encounters INT,
uncovered_encounters INT,
covered_medications INT,
uncovered_medications INT,
covered_procedures INT,
uncovered_proceduress INT,
covered_immunizations INT,
uncovered_immunizations INT,
unique_customers INT,
qols_avg DOUBLE,
member_months INT
);
```

3.4.6 Explanation of Keys

Each table is designed with primary and foreign keys to establish relationships between the different entities. Below is an explanation of the key constraints used in the database schema:

- **Primary Key (PK)**: The primary key is used to uniquely identify each record in a table. For example, `patient_id` is the primary key in the `Patients` table.
- **Foreign Key (FK)**: Foreign keys are used to link tables and ensure referential integrity. For example, the `patient_id` in the `Encounters`, `Claim_Transactions`, and `Procedures` tables references the `Patients` table.

3.4.7 Bulk Data Loading

To efficiently populate the database with large volumes of data, bulk loading techniques were used via SQL scripts. The cleaned and preprocessed CSV files corresponding to each main table were placed inside a dedicated subfolder named `csv`, which was moved into MySQL's default `Uploads` directory to ensure proper file access. For instance, on the system used, the full path was:

`D:\MySQL\MySQL Server 9.3\Uploads\csv`

Placing the files in this directory allowed MySQL to recognize and read them using the `LOAD DATA INFILE` command without encountering permission or path issues.

```
LOAD DATA INFILE 'D:/MySQL/MySQL Server 9.3/Uploads/csv/patients.csv'
INTO TABLE Patients
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

This method was applied to all major tables such as `Patients`, `Encounters`, `Procedures`, `Claim_Transactions`, and `Payers`. Using `LOAD DATA INFILE` significantly reduced loading time and ensured consistent and accurate data import.

Note: As was mentioned earlier, empty strings were handled **after** data loading. Since MySQL is intolerant of empty strings in columns declared as numeric types (e.g., `INT`, `DOUBLE`, `DATE`), some attributes were initially defined as `VARCHAR` to avoid import errors.

Although several of these columns were not critical for analysis, others—especially date fields and codes—required precise handling. The chosen solution was to post-process the tables by converting empty strings to `NULL`, followed by altering the column types to their intended numeric or date formats. Below are some examples:

```
-- For the 'conditions' table
UPDATE conditions
SET end_date = NULL
WHERE end_date = '';

ALTER TABLE conditions
MODIFY COLUMN end_date DATE;

-- For the 'medications' table
UPDATE medications
SET reason_code = NULL
WHERE reason_code = '';

ALTER TABLE medications
MODIFY COLUMN reason_code DOUBLE;
```

This approach ensured compatibility during import while preserving schema consistency and supporting accurate analysis in subsequent stages.

IV Data Warehouse Design and Implementation

Data warehousing is a fundamental component of modern decision support systems. Its primary goal is to consolidate data from various operational sources into a unified, con-

sistent, and query-optimized environment that supports analytical processing, strategic decision-making, and long-term trend analysis. By structuring data in a way that facilitates efficient querying and reporting, data warehouses empower organizations to extract meaningful insights from vast volumes of historical and real-time data.

In this project, the data warehouse is structured as a collection of subject-oriented datamarts, each designed to address a specific analytical need within the healthcare domain. Instead of physically materializing fact and dimension tables, the implementation relies on SQL views to define the logic of each datamart. This virtualized approach provides flexibility, reduces storage overhead, and allows for easier maintenance.

To drive the design of these datamarts, we identified a set of eleven decisional queries that represent key business questions related to clinical interactions, claims, observations, medications, procedures, and immunizations. Each query informed the creation of a corresponding datamart, ensuring that the warehouse is aligned with actual analytical use cases. The result is a modular, query-driven data warehouse architecture capable of delivering timely and relevant insights.

4.1 Decisional Queries

To ensure that the data warehouse meets real-world analytical needs, a set of eleven decisional queries was defined. These queries represent key business questions that guide healthcare organizations in monitoring performance, understanding patient behavior, and optimizing resource allocation. Each query directly influenced the design of a specific datamart within the warehouse.

The queries are grouped below according to the datamart they correspond to:

- **Clinical Interactions (Encounters) Datamart**
  - Which organizations have performed the highest number of medical encounters in 2020 and 2021, grouped by quarter?
  - Which encounter classes were most common among female patients under 40 years old during the current year grouped by total claim cost?
  - What are the most frequent care locations for women under 40 in the current year (2021)?
- **Claim Transactions Datamart**
  - What is the total and average transaction amount for each claim type per provider in the past year, grouped by month?
  - What is the annual distribution of insured vs. uninsured patients per age group (2021)?
- **Observations Datamart**
  - Count of adult patients with elevated body weight in 2021, by gender.
  - What were the average observed glucose values in 2021 by patient age group (under 30, 30–60, over 60)?
- **Medications Datamart**

  - What is the average base cost of medications prescribed to patients over 65 years old in 2021, grouped by payer name?
  - What is the average number of dispenses per medication in 2021, grouped by patient gender and ethnicity?

- **Procedures Datamart**
  - How many procedures were performed in 2021 on female patients under 30 years old, grouped by procedure description/code?

- **Immunizations Datamart**
  - What percentage of patients aged 65 and over received at least one influenza-related immunization during the winter months (Dec–Feb) in the past decade, and what was the average base cost of those immunizations?

4.2 Datamart Attribute Tree, Fact and Star Schema

To design a data warehouse that effectively supports the defined decisional queries, each datamart was carefully modeled by identifying the necessary analytical attributes and organizing them into a structured schema. For each datamart, an attribute tree was generated to capture all relevant paths from the central fact to associated dimensions. This tree was refined using techniques such as pruning (to eliminate irrelevant attributes) and grafting (to simplify relationships and resolve converging paths).

The resulting attribute trees informed the construction of each fact schema, which defines the central measurements (facts) and their analytical context. Finally, star schemas were derived to represent the logical organization of facts and dimensions in a query-optimized structure. These schemas serve as the blueprint for the SQL views implemented in the virtualized data warehouse.

The following subsections present the complete design for each of the six datamarts: Encounters, Claim Transactions, Observations, Medications, Procedures, and Immunizations.

4.2.1 Encounters Datamart

Related Decisional Queries

- Which organizations have performed the highest number of medical encounters in 2020 and 2021, grouped by quarter?
- Which encounter classes were most common among female patients under 40 years old during the current year grouped by total claim cost?
- What are the most frequent care locations for women under 40 in the current year (2021)?

1. Attribute Tree and Fact Schema The attribute tree for the Encounters datamart is constructed by tracing all relevant attributes from the central fact — encounters — toward the associated entities via foreign key. To simplify and focus the model, attributes not required by the decisional queries were pruned, while essential relationships (e.g.,

between encounters, patients, and organizations) were retained. Converging paths were resolved using grafting to ensure schema normalization and clarity.

Notably, certain dimensions follow hierarchical structures to support multi-level aggregations. For instance, the **Date** dimension allows navigation from individual days to months and quarters, enabling time-based analysis such as grouping encounters by quarter. Similarly, patient **age** is categorized into predefined age groups (e.g., under 30, 30–60, over 60), allowing filtering and segmentation in analytical queries. These hierarchies are embedded in the fact schema and reflected in the dimension design.

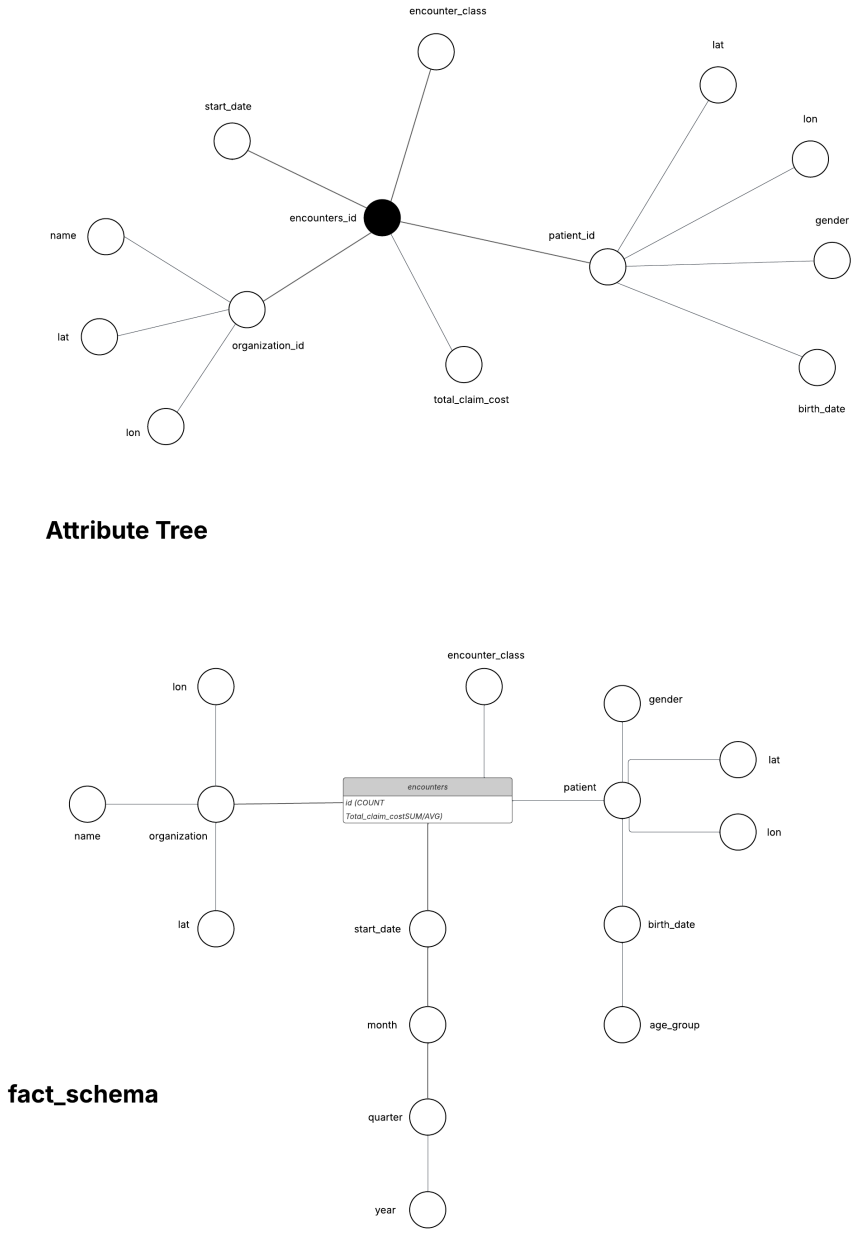


Figure 3: Attribute Tree & Fact Schema for Encounters Datamart

The fact schema is derived by identifying the measures (e.g., total claim cost) and connecting dimension keys from the pruned attribute tree.

2. Star Schema The star schema provides a denormalized, query-efficient structure. For the Encounters datamart, the central fact table references the following dimension tables:

- **Date** — to enable temporal analysis (by quarter/year).
- **Organization** — providing details about the healthcare facility.
- **Patient** — including demographic filters (gender, age).
- **Encounter Class** — categorizing the type of medical visit.

Figure 4: Star Schema for Encounters Datamart

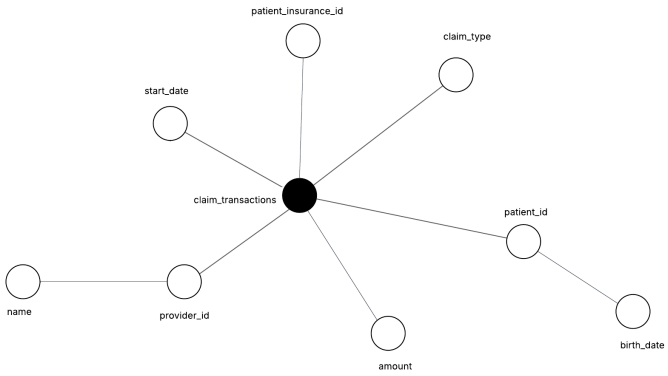
4.2.2 Claim Transactions Datamart

Related Decisional Queries

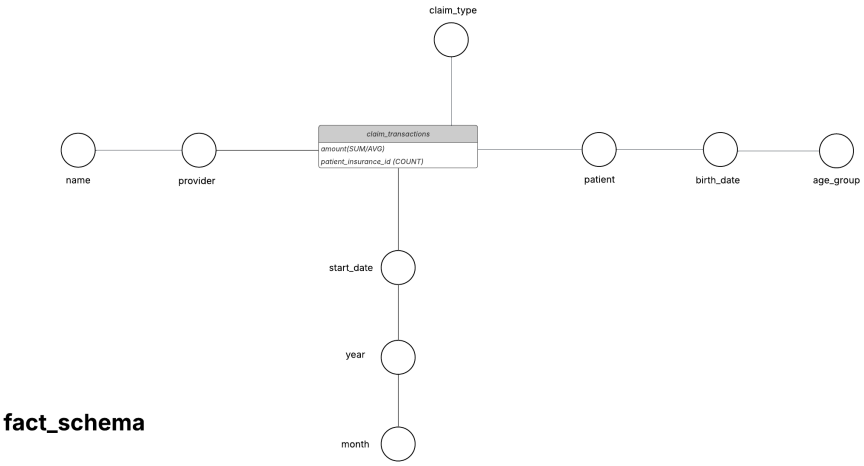
- What is the total and average transaction amount for each claim type per provider in the past year, grouped by month?
- What is the annual distribution of insured vs. uninsured patients per age group (2021)?

1. Attribute Tree and Fact Schema The attribute tree for the Claim Transactions datamart is constructed by following foreign key relationships from the claim transactions toward related entities such as patients, providers, and claim types. Redundant or unrelated paths were pruned, and intersecting paths were unified to maintain consistency and clarity.

As with other datamarts, hierarchical dimensions such as **Date** (day → month → year) and **Age Group** (e.g., under 18, 18–64, 65+) are embedded to enable time-series and demographic analyses. These support filtering and aggregation directly within the fact schema.



Attribute Tree



fact_schema

Figure 5: Attribute Tree & Fact Schema for Claim Transactions Datamart

2. Star Schema The fact table in the star schema is connected to multiple dimensions relevant to transaction analytics:

- **Date** — supporting monthly and yearly aggregations.
- **Provider** — identifying the care provider.
- **Claim Type** — categorizing transactions.
- **Patients** — enabling demographic breakdowns.

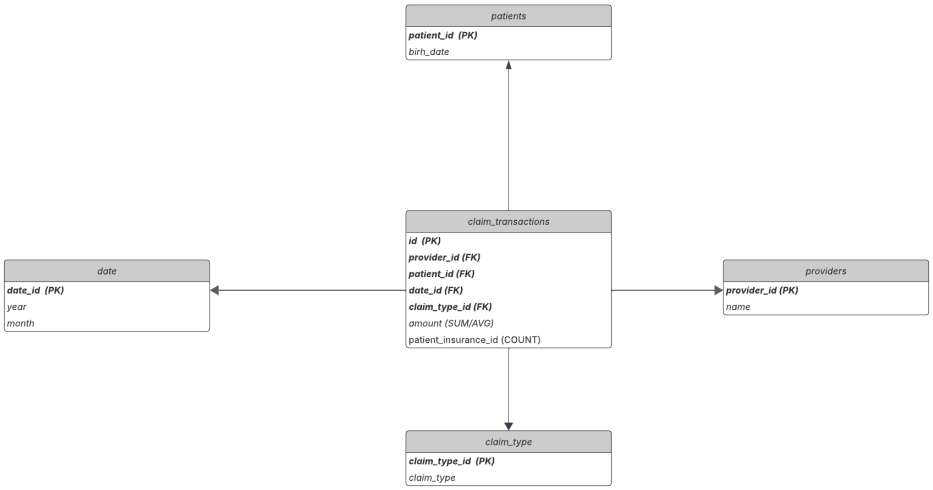


Figure 6: Star Schema for Claim Transactions Datamart

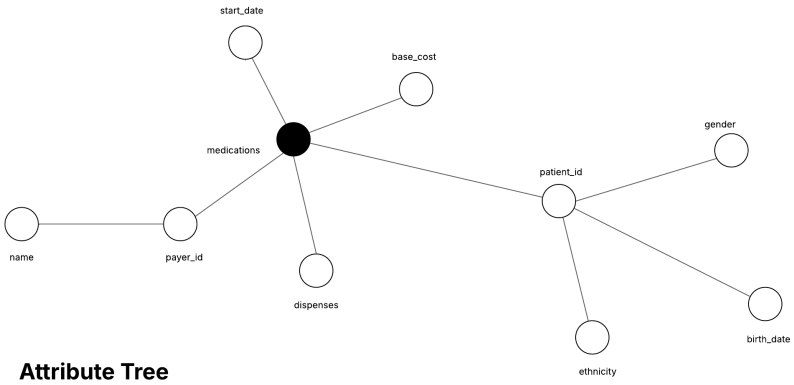
4.2.3 Medications Datamart

Related Decisional Queries

- What is the average base cost of medications prescribed to patients over 65 years old in 2021, grouped by payer name?
- What is the average number of dispenses per medication in 2021, grouped by patient gender and ethnicity?

1. Attribute Tree and Fact Schema The attribute tree for this datamart includes medication prescriptions, patient demographics, and payer information. Unused clinical data were pruned, and demographic references were grafted to consolidate ethnicity, gender, and age.

The **Date**, **Age Group**, and **Payer Name** dimensions follow hierarchical logic (e.g., year, month; age ranges), which are embedded into the fact schema to enable grouped reporting and filtering.



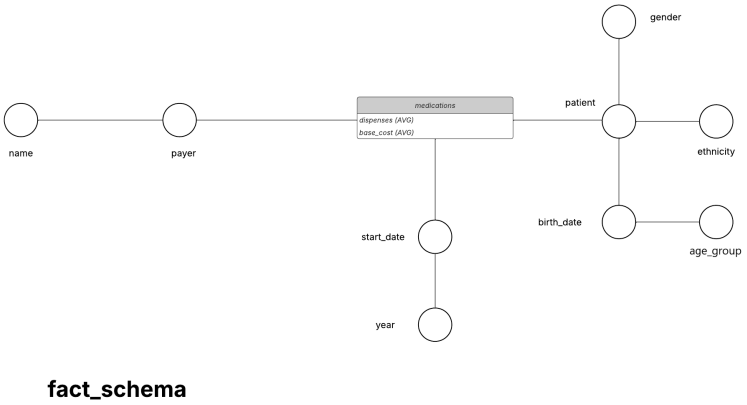


Figure 7: Attribute Tree & Fact Schema for Medications Datamart

2. Star Schema Key analytical dimensions include:

- **Date** — linked to prescription dates.
- **Patient** — including gender, age group, and ethnicity.
- **Payer** — source of payment for medication.

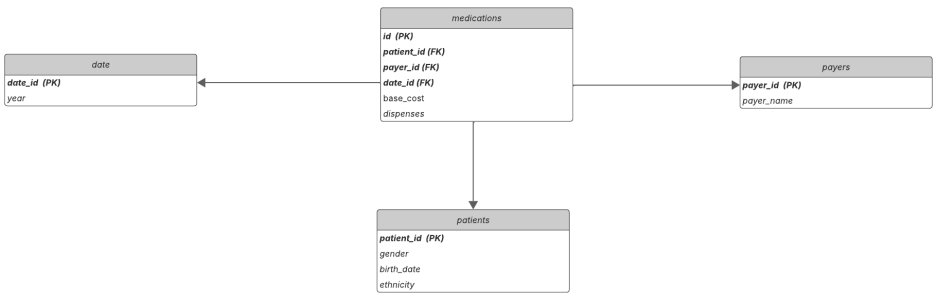


Figure 8: Star Schema for Medications Datamart

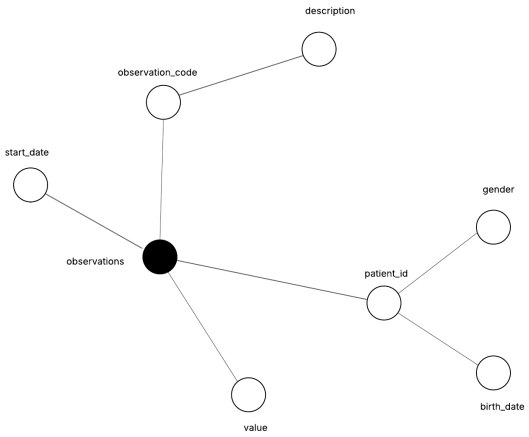
4.2.4 Observations Datamart

Related Decisional Queries

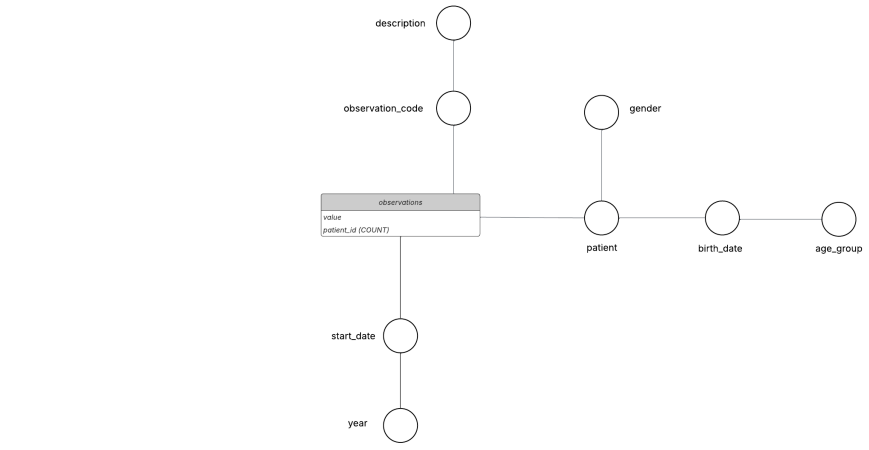
- Count of adult patients with elevated body weight in 2021, by gender.
- What were the average observed glucose values in 2021 by patient age group (under 30, 30–60, over 60)?

1. Attribute Tree and Fact Schema The Observations datamart's attribute tree is centered on recorded health observations linked to patients and observation types. Pruning was applied to discard unrelated metadata while preserving gender, age, and observation values. Grafting was used to manage multiple references to patient demographics.

Hierarchies in this datamart include **Date** and **Age Group**, which allow aggregation of metrics over time or population segments. In addition, a second hierarchy exists between **code** and **description**, where the description provides a more readable dimension for analytical grouping. This hierarchy is especially relevant in medical classification systems (e.g., procedure codes, immunizations codes) and is reflected in the dimension design. All hierarchical structures are embedded within the fact schema to support flexible and intuitive querying.



Attribute Tree



fact_schema

Figure 9: Attribute Tree & Fact Schema for Observations Datamart

2. **Star Schema** Key dimensions include:

   - **Date** — identifying observation periods.
   - **Patient** — including gender and age grouping.
   - **Description** — such as glucose or body weight.

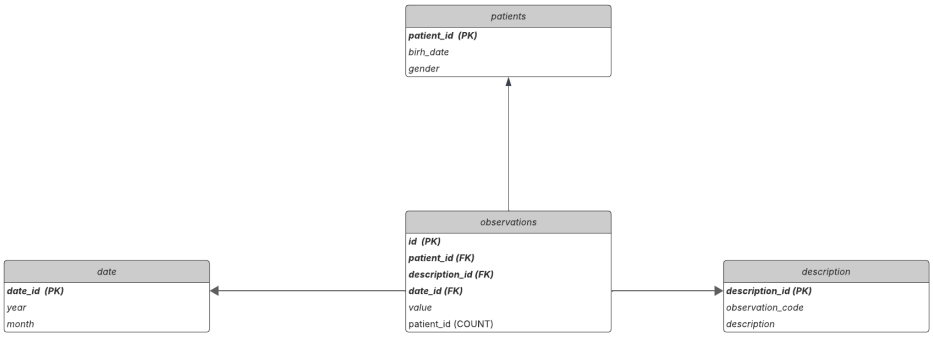


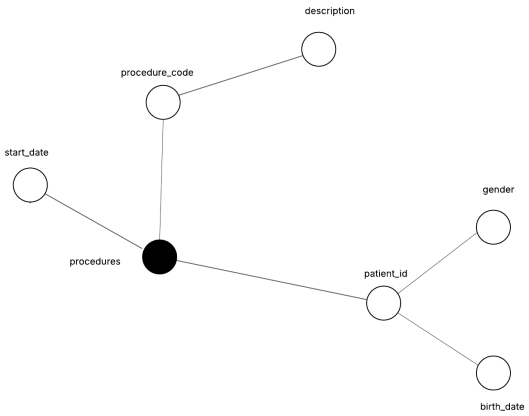
Figure 10: Star Schema for Observations Datamart

4.2.5 Procedures Datamart

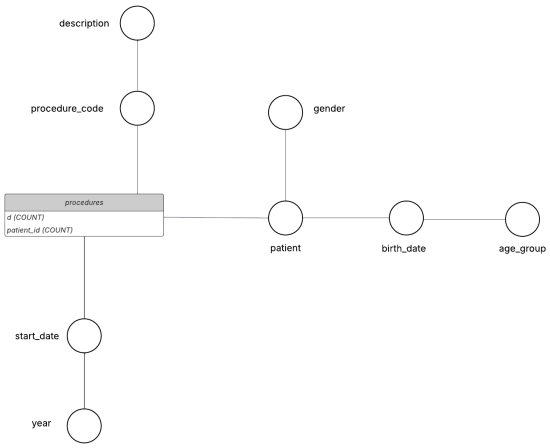
Related Decisional Queries

- How many procedures were performed in 2021 on female patients under 30 years old, grouped by procedure description/code?

1. Attribute Tree and Fact Schema The attribute tree maps procedures to patients and procedure descriptions, while demographic attributes such as gender and age are preserved. Pruning eliminated attributes not influencing query logic. The **Date** and **Age Group** dimensions support temporal and cohort analysis, structured hierarchically.



Attribute Tree



fact_schema

Figure 11: Attribute Tree & Fact Schema for Procedures Datamart

2. **Star Schema** The star schema supports aggregation by:

   - **Date** — when the procedure occurred.
   - **Patient** — with demographic breakdowns.
   - **Description** — identified by code or description.

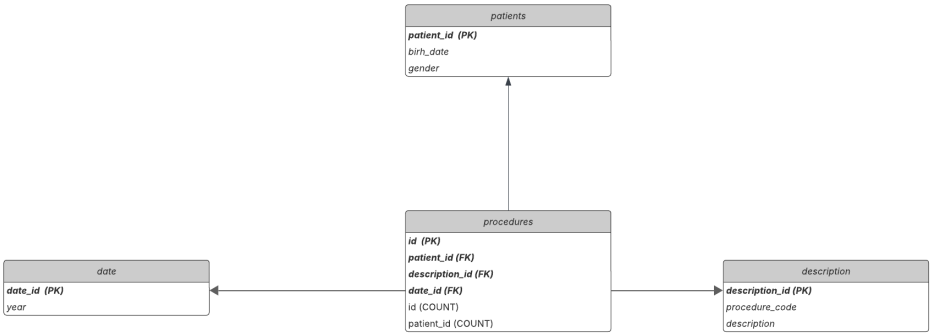


Figure 12: Star Schema for Procedures Datamart

4.2.6 Immunizations Datamart

Related Decisional Queries

- What percentage of patients aged 65 and over received at least one influenza-related immunization during the winter months (Dec–Feb) in the past decade, and what was the average base cost of those immunizations?

1. Attribute Tree and Fact Schema The attribute tree includes immunization records joined to patients and immunization types. Demographics and dates are retained, while non-contributing attributes are pruned. Date-based filtering supports seasonal analysis (e.g., Dec–Feb), while **Age Group** enables focus on older populations.

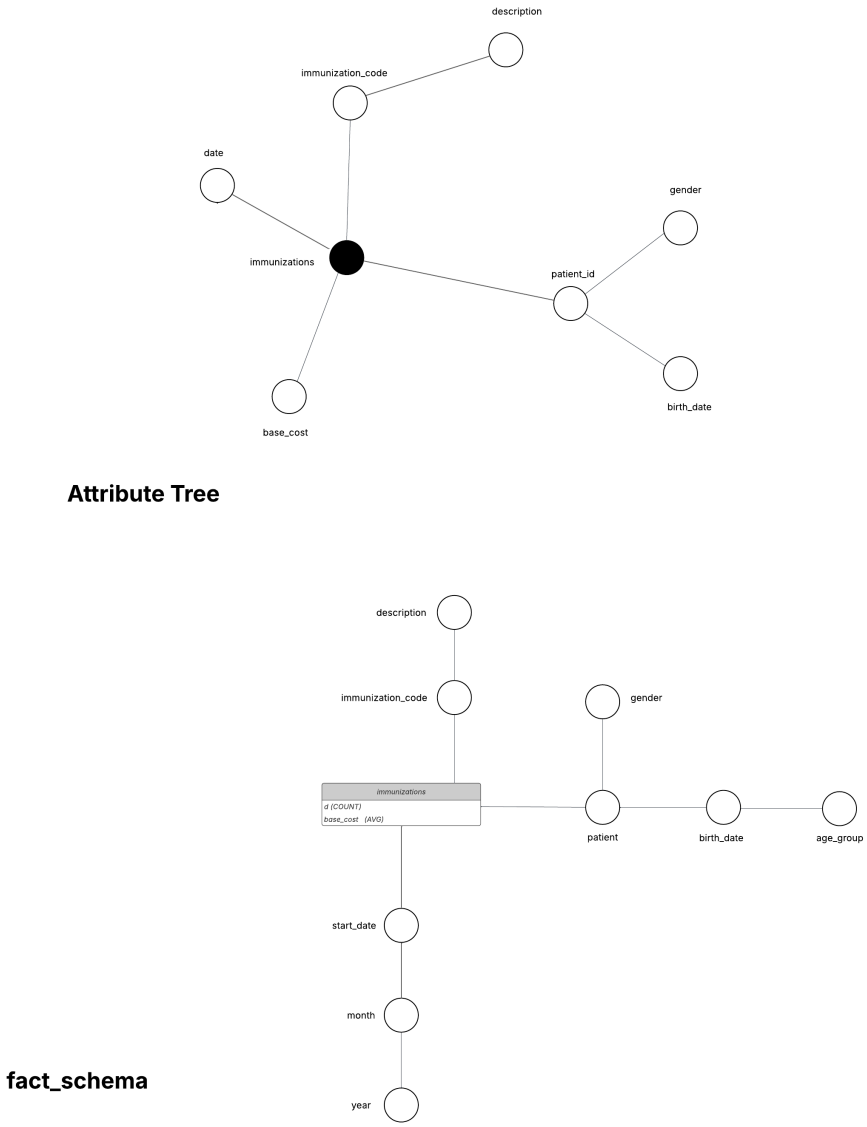


Figure 13: Attribute Tree & Fact Schema for Immunizations Datamart

2. **Star Schema** Dimensions supporting the analysis include:

   - **Date** — including season-based hierarchies.
   - **Patient** — with age group and gender.
   - **Description** — to isolate influenza-related vaccines.

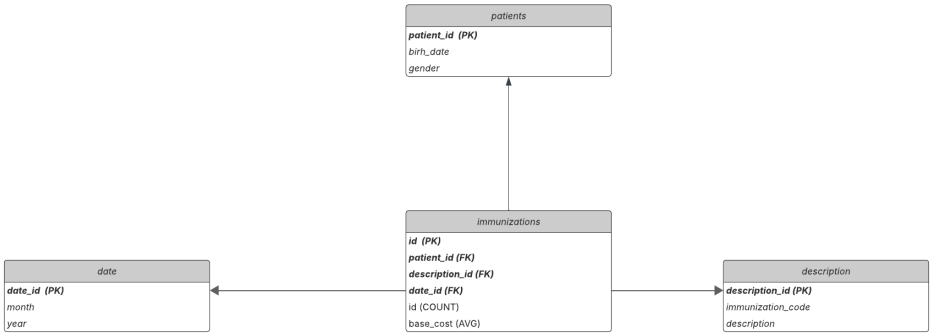


Figure 14: Star Schema for Immunizations Datamart

Summary of Measurement Functions by Datamart

Datamart	**Measurement Functions Used**
Encounters	COUNT([Encounter ID])
Claim Transactions	SUM([Amount]), AVG([Amount]), COUNTD([Patient ID])
Observations	AVG([Value]), COUNT([Patient ID])
Medications	AVG([Base Cost]), AVG([Dispenses])
Procedures	COUNT([Patient ID])
Immunizations	COUNTD([Patient ID]), AVG([Base Cost])

Table 5: Measurement Functions Used per Datamart

V Datamart Implementation and Analytical Interface

5.1 Implementation of SQL Views for Datamarts

To implement the designed data warehouse, each datamart was constructed as a SQL view instead of being materialized into physical fact and dimension tables. This architectural choice ensures flexibility, reduces storage requirements, and simplifies data maintenance.

Each view encapsulates the logic derived from the corresponding attribute tree and fact schema. Filtering conditions, joins, and transformations are applied directly in SQL to calculate the required measures and define the analytical context for each datamart.

Hierarchical dimensions such as **Date** (day → month → quarter) and **Age Group** (e.g., under 30, 30–60, over 60) are computed within the views to support grouped analysis. For the **Medications**, **Procedures**, and **Immunizations** datamarts, an additional semantic hierarchy **Code** → **Description** was introduced. This allows analysts to work with more interpretable labels (descriptions) while keeping internal consistency with coded data.

All views were developed and tested to ensure full alignment with the decisional queries and the logical star schema models discussed previously.

5.1.1 Encounters Datamart

Listing 1: SQL View for Encounters Datamart

```
CREATE OR REPLACE VIEW v_encounters_base AS
SELECT
    e.id AS encounter_id,
    e.patient_id,
    p.gender,
    TIMESTAMPDIFF(YEAR, p.birth_date, '2021-12-31') AS age_in_2021,
    p.address AS patient_address,
    p.city AS patient_city,
    p.lat AS patient_lat,
    p.lon AS patient_lon,
    e.start_date,
    YEAR(e.start_date) AS year,
    QUARTER(e.start_date) AS quarter,
    MONTH(e.start_date) AS month,
    e.encounter_class,
    e.total_claim_cost,
    pr.id AS provider_id,
    pr.specialty,
    o.id AS organization_id,
    o.organization_name,
    o.lat,
    o.lon
FROM encounters e
JOIN patients p ON e.patient_id = p.id
JOIN providers pr ON e.provider_id = pr.id
JOIN organizations o ON e.organization_id = o.id;
```

5.1.2 Claim Transactions Datamart

Listing 2: SQL View for Claim Transactions Datamart

```
CREATE OR REPLACE VIEW v_claim_transaction_base AS
SELECT
    ct.provider_id,
    p.provider_name,
    ct.claim_type,
    YEAR(ct.start_date) AS year,
    MONTH(ct.start_date) AS month,
    CAST(ct.amount AS DECIMAL(10, 2)) AS amount,
    CASE
        WHEN ct.patient_insurance_id IS NULL OR ct.patient_insurance_id
            ↪ = '' THEN 'Uninsured'
        ELSE 'Insured'
    END AS insurance_status,
    ct.patient_id,
    pm.birth_date,
    TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31')) AS age_2021,
    CASE
        WHEN TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31')) < 30
            ↪  THEN 'Under 30'
```

```
        WHEN TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31'))
           ↪ BETWEEN 30 AND 60 THEN '30  60  '
        WHEN TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31')) > 60
           ↪  THEN 'Over 60'
        ELSE 'Unknown'
    END AS age_group
FROM claim_transaction ct
JOIN providers p ON ct.provider_id = p.id
LEFT JOIN patients pm ON ct.patient_id = pm.id
WHERE ct.amount IS NOT NULL;
```

5.1.3 Observations Datamart

Listing 3: SQL View for Observations Datamart

```
CREATE OR REPLACE VIEW v_observations_base AS
SELECT
  o.id AS observation_id,
  o.patient_id,
  pm.gender,
  pm.birth_date,
  o.date AS observation_date,
  YEAR(o.date) AS observation_year,
  -- New age relative to 2021-12-31
  TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31')) AS age_2021,
  CASE
    WHEN TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31')) < 30
       ↪ THEN 'Under 30'
    WHEN TIMESTAMPDIFF(YEAR, pm.birth_date, DATE('2021-12-31')) BETWEEN
       ↪ 30 AND 60 THEN '30  60  '
    ELSE 'Over 60'
  END AS age_group_2021,
  o.observation_desc,
  o.observation_type,
  o.value,
  CAST(o.value AS DECIMAL(10,2)) AS value_numeric
FROM observations o
JOIN patients pm ON o.patient_id = pm.id
WHERE o.value IS NOT NULL;
```

5.1.4 Medications Datamart

This view includes medication prescriptions, payer data, and patient info. Code → description hierarchy is used for drug classification.

Listing 4: SQL View for Medications Datamart

```
CREATE OR REPLACE VIEW v_medications_base AS
SELECT
  m.id AS medication_id,
  m.patient_id,
  pm.gender,
  pm.ethnicity,
  pm.birth_date,
  p.payer_name,
  m.medication_desc,
```

```
  m.base_cost,
  m.total_cost,
  m.payer_coverage,
  m.dispenses,
  m.start_date,
  YEAR(m.start_date) AS medication_year
FROM medications m
JOIN patients pm ON m.patient_id = pm.id
JOIN payers p ON m.payer_id = p.id
WHERE m.base_cost IS NOT NULL
  AND m.dispenses IS NOT NULL
  AND m.start_date IS NOT NULL;
```

5.1.5 Procedures Datamart

Listing 5: SQL View for Procedures Datamart

```
CREATE OR REPLACE VIEW v_procedures_base AS
SELECT
  pr.procedure_desc,
  pr.start_date,
  YEAR(pr.start_date) AS procedure_year,
  pr.patient_id,
  pm.gender,
  pm.birth_date
FROM procedures pr
JOIN patients pm ON pr.patient_id = pm.id
WHERE pr.start_date IS NOT NULL
  AND pr.procedure_desc IS NOT NULL;
```

5.1.6 Immunizations Datamart

This view summarizes immunizations, including vaccine codes/descriptions and patient age groups.

Listing 6: SQL View for Immunizations Datamart

```
CREATE OR REPLACE VIEW v_immunizations_base AS
SELECT
  pm.id AS patient_id,
  pm.birth_date,
  pm.gender,
  YEAR(i.date) AS year,
  MONTH(i.date) AS month,
  i.immunization_desc,
  i.base_cost
FROM immunizations i
JOIN patients pm ON i.patient_id = pm.id;
```

5.2 Analytical Exploration with Tableau Desktop

To facilitate interactive analysis, each SQL view was imported into Tableau Desktop as a separate data source, corresponding to a single datamart. Tableau acts as the front-end analytical interface, enabling users to perform visual exploration and drill-down analyses on the warehouse data.

Within each Tableau data source, several **calculated fields** were added to enhance analytical capabilities.

In practice, due to the large volume of data involved, it was preferable and more interpretable to visualize only the **top records** (e.g., top 3 organizations or most frequent classes) rather than the entire dataset. This approach enhances clarity and supports comparative analysis without overwhelming the visual interface.

Moreover, as many of the queries were centered around the most recent year in the dataset (2021), a calculated column for **Patient Age in 2021** was included during the SQL view creation phase. This decision ensured consistency across views and simplified filter and group logic in Tableau.

5.2.1 Encounters Datamart Visualizations

This set of visualizations corresponds to the decisional queries defined for the Encounters datamart.

5.2.2 Query 1: Number of Encounters per Organization per Quarter (2020–2021)

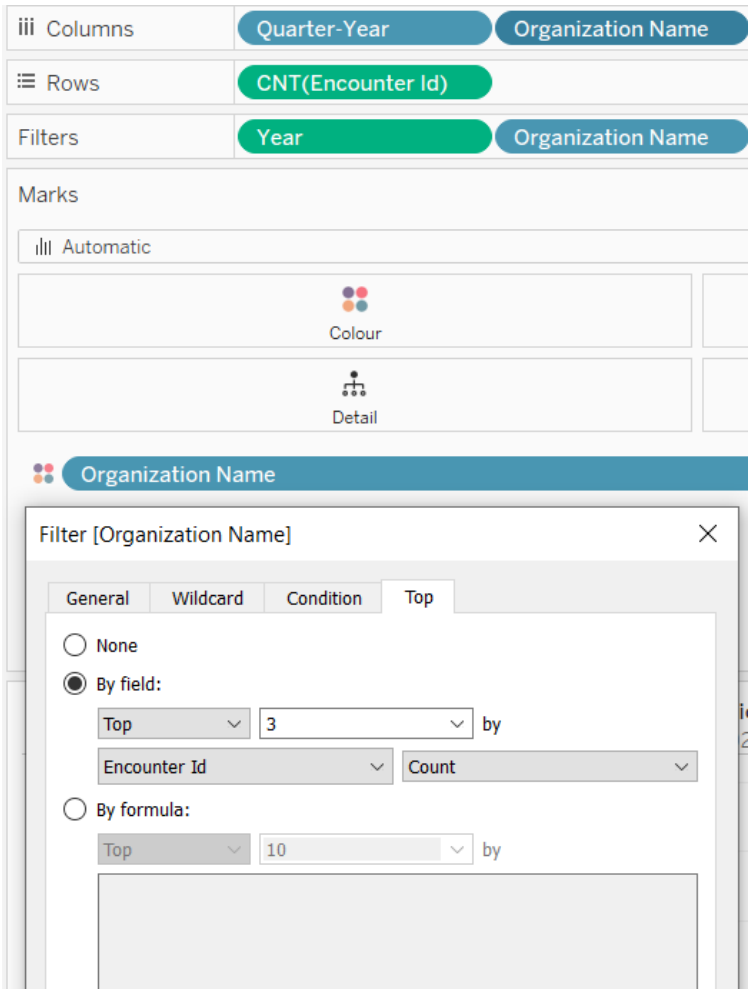


Figure 15: Shelf Configuration for Query 1

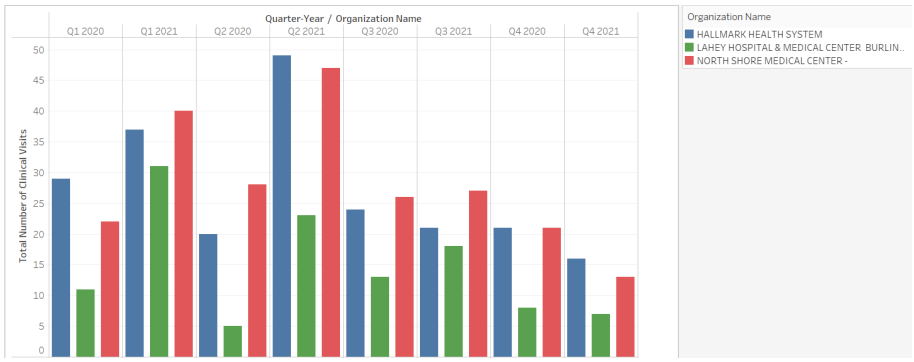


Figure 16: Top 3 organizations with the highest number of medical encounters in the two years (2021 and 2020), grouped by quarter

5.2.3 Query 2: Most Common Encounter Classes (Females under 40)

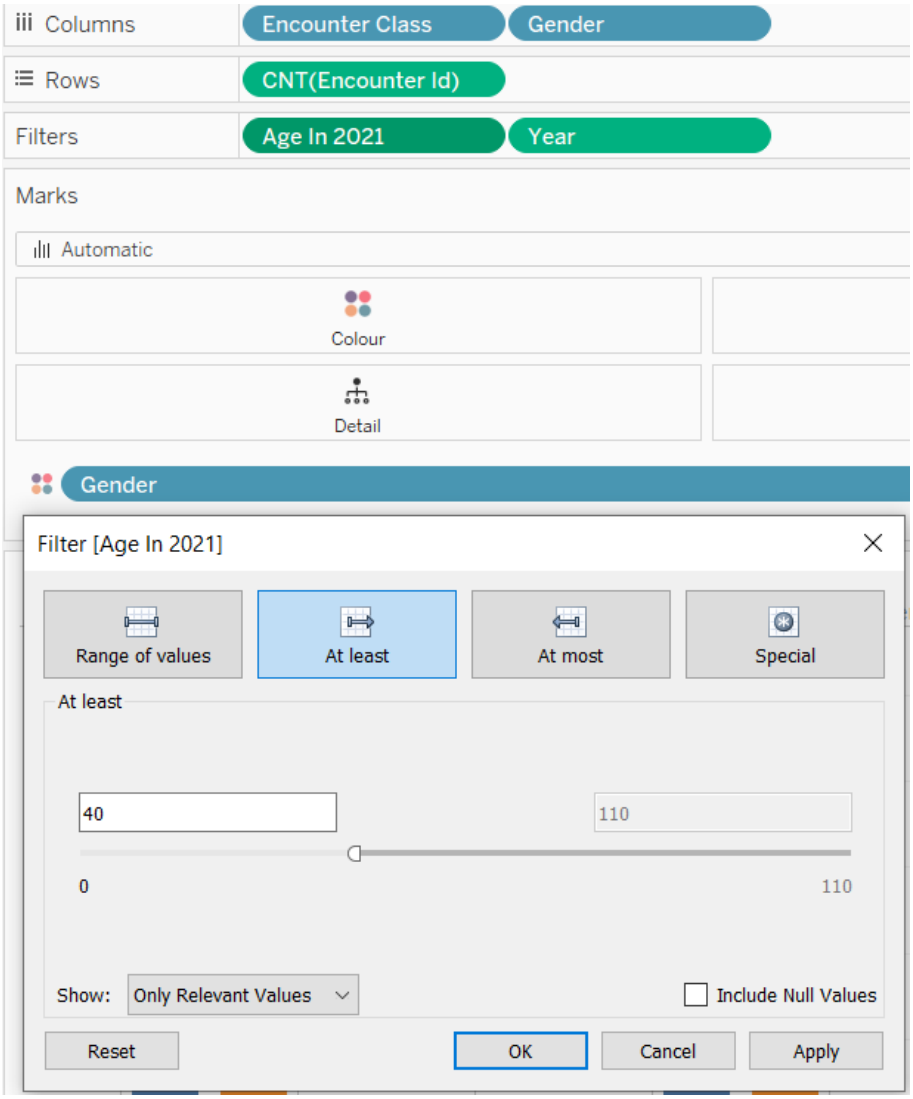


Figure 17: Shelf Configuration for Query 2

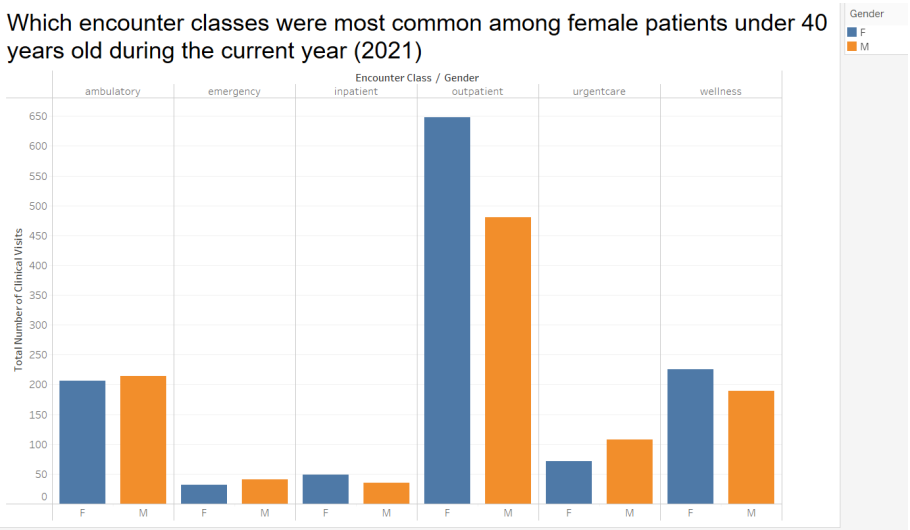


Figure 18: Top Encounter Classes for Female Patients Under 40

5.2.4 Query 3: Location of young women seeking medical care during the current year(2021)

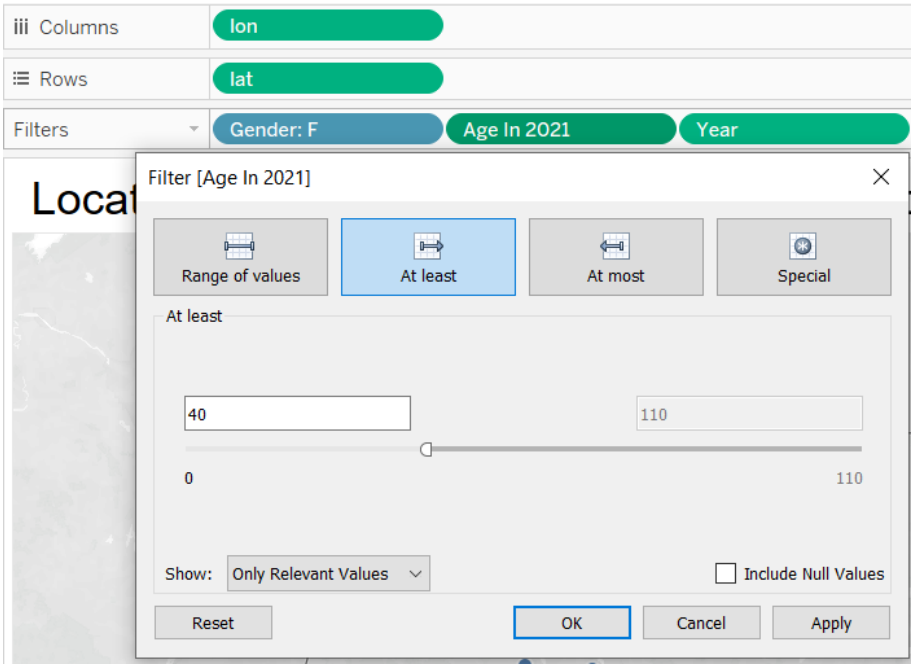


Figure 19: Shelf Configuration for Query 3

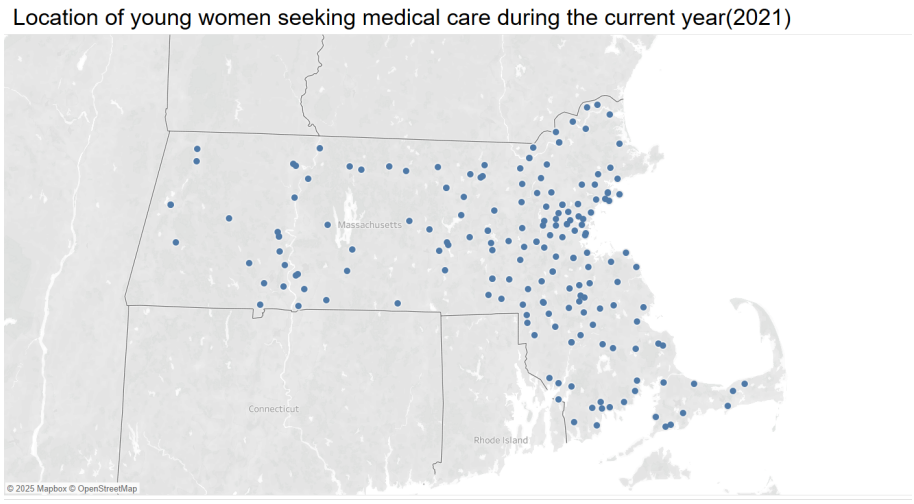


Figure 20: Location of young women seeking medical care during 2021

5.2.5 Claim Transactions Datamart Visualizations

This section presents the visualizations related to the Claim Transactions datamart.

5.2.6 Query 1: Total and Average Transaction Amount per Claim Type and Provider by Month (2021)

Given the complexity of this query — which combines grouping across both providers and claim types over time — it was more effective to represent the results using **two separate visualizations**. One chart displays the *The total and average transaction amount trends for each claim type in 2021, grouped by month*, and the other shows the *the total transaction amount per top 5 providers with the highest total amount in 2021, grouped by month*, both segmented by month. This split improves readability and allows more focused interpretation.

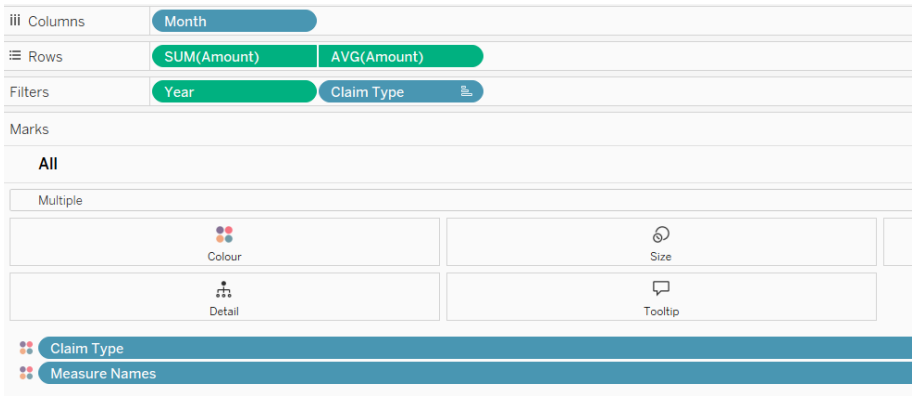


Figure 21: Shelf Configuration for Query 1 First Part

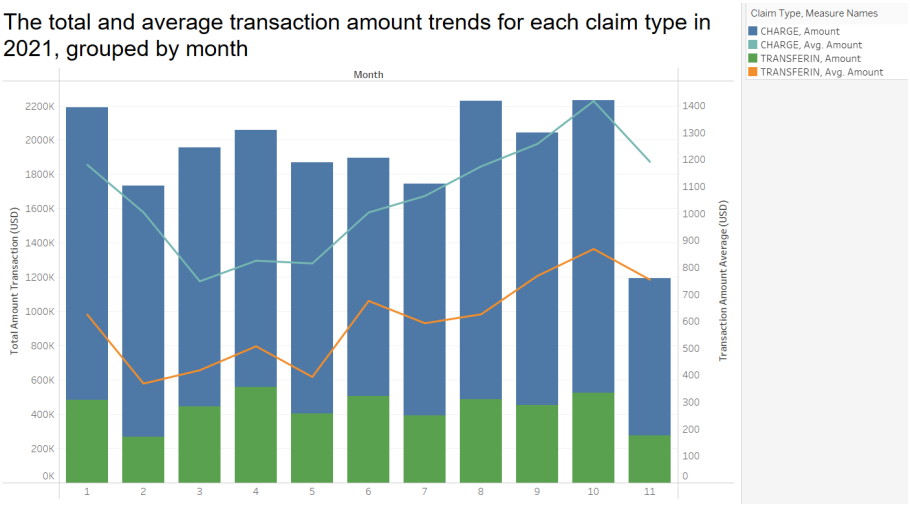


Figure 22: Total and Average Transaction Amount Trends grouped by Claim Type and Month in 2021

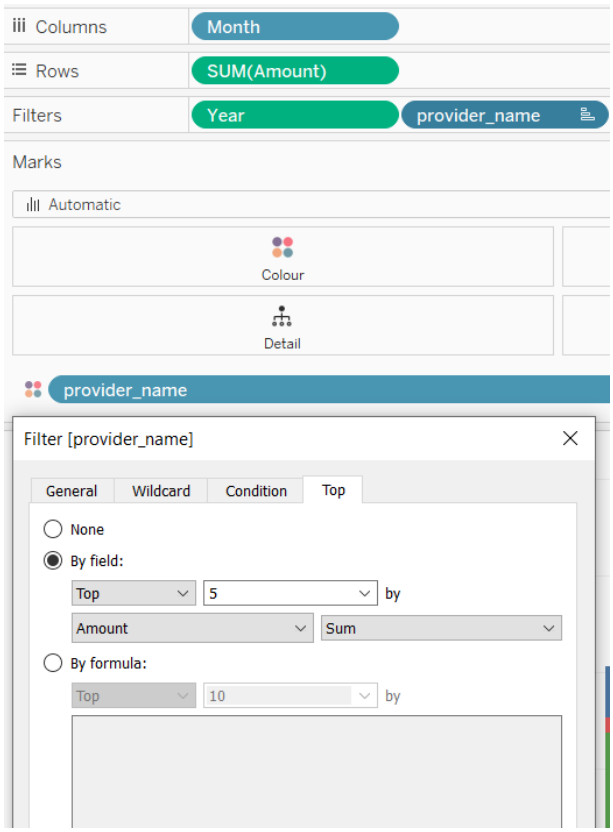


Figure 23: Shelf Configuration for Query 1 Second Part

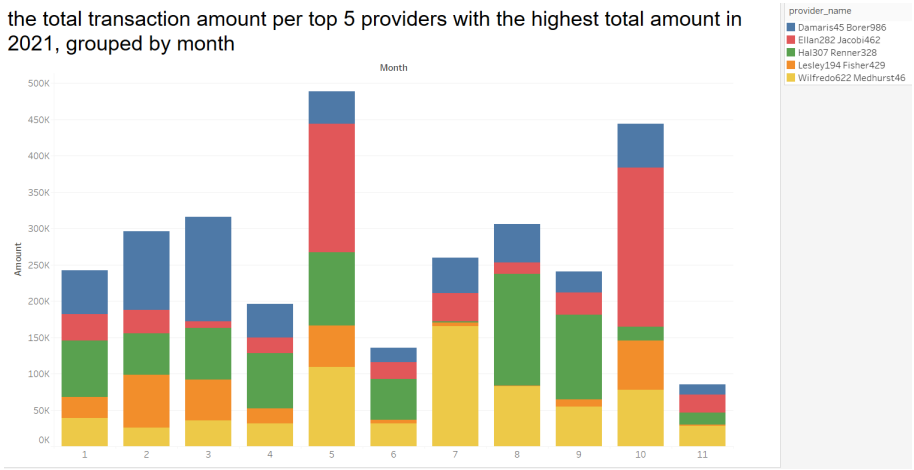


Figure 24: Total Transaction Amount per Top 5 Providers with the Highest Total amount in 2021, grouped by month

5.2.7 Query 2: 2021 Patient Coverage Analysis by Age Segmentr

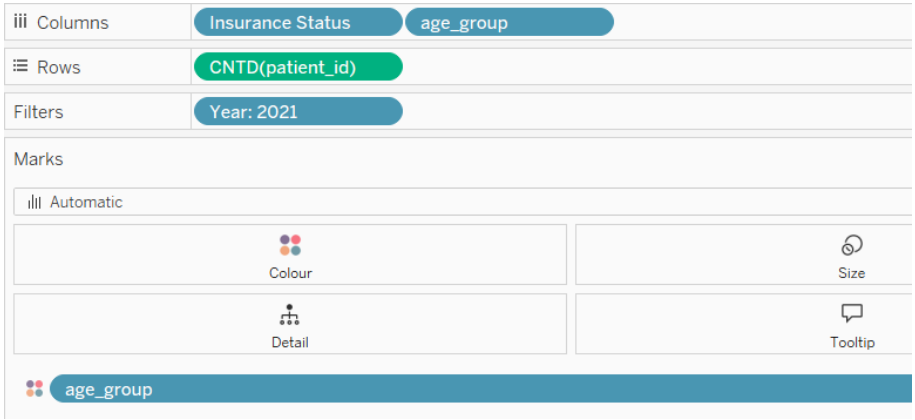


Figure 25: Shelf Configuration for Query 2

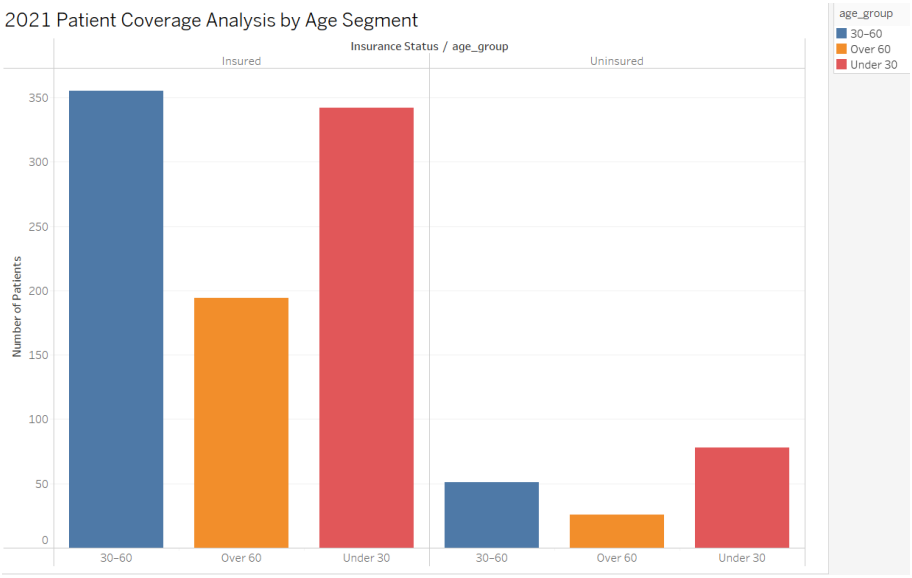


Figure 26: 2021 Patient Coverage Analysis by Age Segment

5.2.8 Observations Datamart Visualizations

5.2.9 Query 1: Count of Adult Patients with Elevated Body Weight in 2021, Grouped by Gender

To identify patients with elevated body weight, we relied on the Body Mass Index (BMI) metric, which depends on both a patient's latest body weight and height measurements. These values were stored in the dataset as observations with specific descriptions.

To extract valid inputs for BMI, we filtered the dataset by **Observation Description** equal to either "Body Height" or "Body Weight" and retrieved the most recent value for each patient in 2021. Since height was recorded in centimeters, it was converted to meters before the BMI calculation.

The following calculated fields were created in Tableau:

- **Body Height (in meters)** — the latest recorded value in 2021, divided by 100:

Listing 7: Calculated Field: Body Height (2021)

```
{ FIXED [Patient Id] :
    MAX(
        IF [Observation Desc] = 'Body Height'
           AND [Observation Date] =
              { FIXED [Patient Id] : MAX(
                  IF [Observation Desc] = 'Body Height'
                     AND YEAR([Observation Date]) = 2021
                  THEN [Observation Date]
                  ELSE NULL
                  END
              )}
        THEN FLOAT([Value])/100
        ELSE NULL
        END
    )
}
```

- **Body Weight (in kg)** — similarly filtered for the most recent 2021 value:

Listing 8: Calculated Field: Body Weight (2021)

```
{ FIXED [Patient Id] :
    MAX(
        IF [Observation Desc] = 'Body Weight' AND ISNULL([Value]) =
           ↪ FALSE
           AND [Observation Date] =
              { FIXED [Patient Id] : MAX(
                  IF [Observation Desc] = 'Body Weight'
                     AND YEAR([Observation Date]) = 2021
                  THEN [Observation Date]
                  ELSE NULL
                  END
              )}
        THEN FLOAT([Value])
        ELSE NULL
        END
    )
}
```

- **BMI** — computed using the standard formula:

Listing 9: Calculated Field: BMI

```
[Body Weight] / ([Body Height]^2)
```

Finally, a filter was applied to include only patients with:

- **BMI 25** (classified as overweight or obese),
- **Age 18** as of 2021 (adults only),
- and known **gender** values.

The result is a clean and focused visualization showing the count of adult patients with elevated body weight by gender.

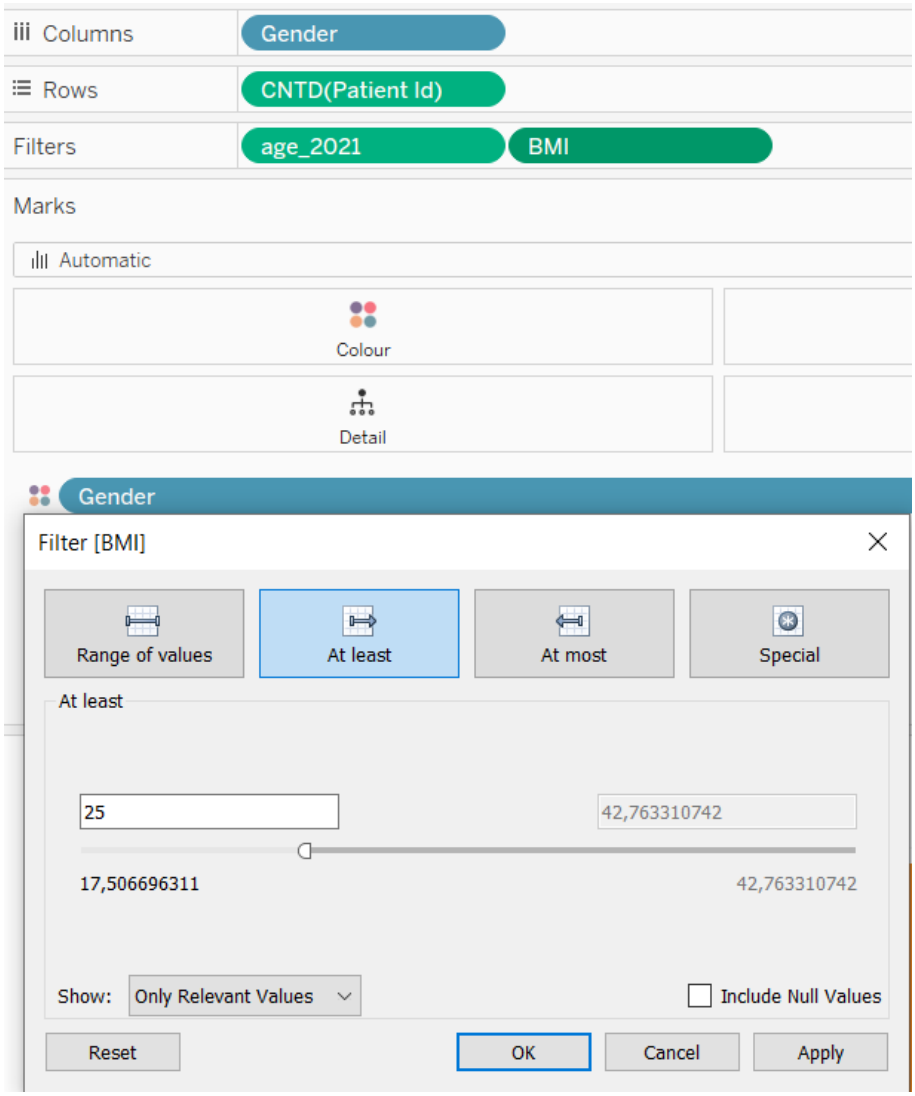


Figure 27: Shelf Configuration for Query 1

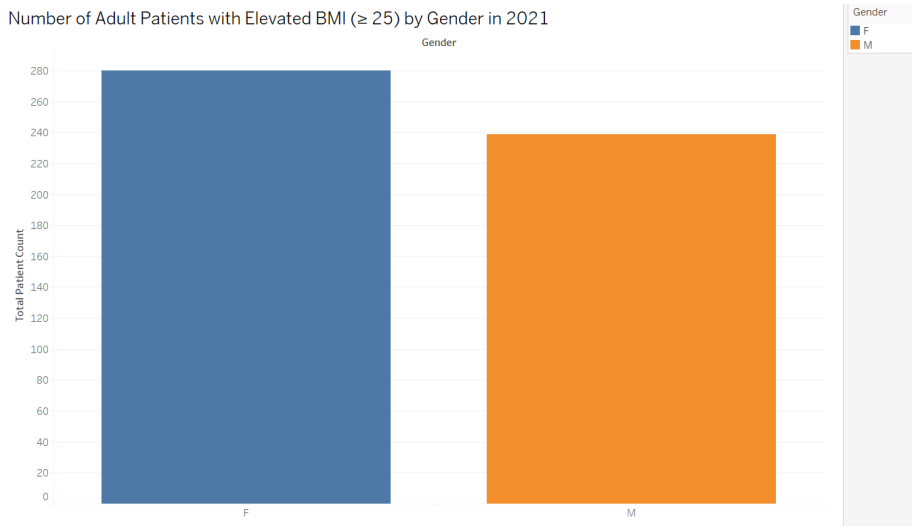


Figure 28: Adult Patients with Elevated Body Weight in 2021, by Gender

5.2.10 Query 2: Average Glucose Distribution over different patients in 2021r

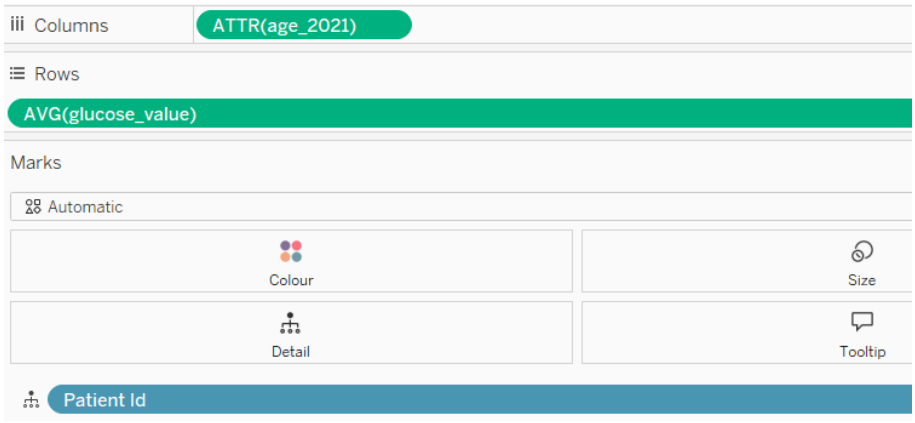


Figure 29: Shelf Configuration for Query 2

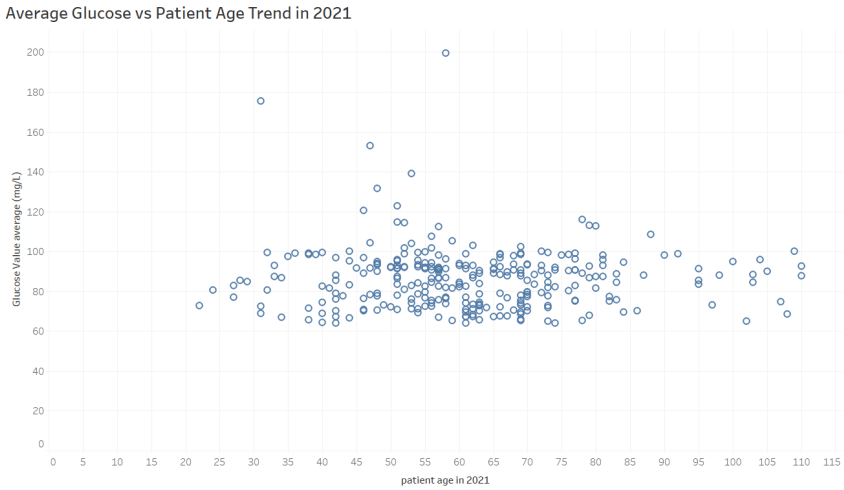


Figure 30: Average Glucose Distribution over different patients in 2021

5.2.11 Medications Datamart Visualizations

This section presents the visualizations for the Medications datamart. Given the complexity of prescription and cost data, only the most relevant groupings were visualized to ensure clarity and usability.

5.2.12 Query 1: Average Base Cost of Medications Prescribed to Patients Over 65 in 2021, Grouped by Payer Name

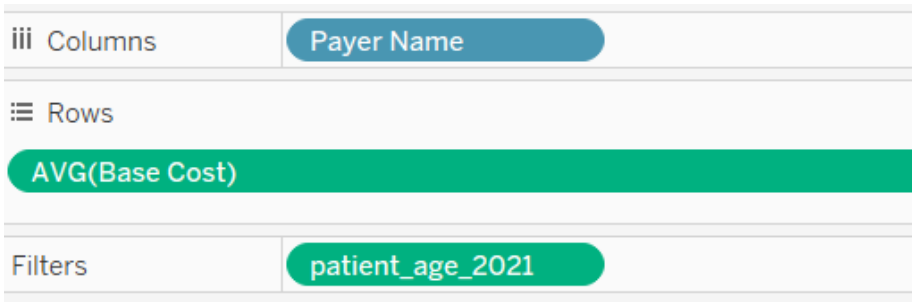


Figure 31: Shelf Configuration for Query 1

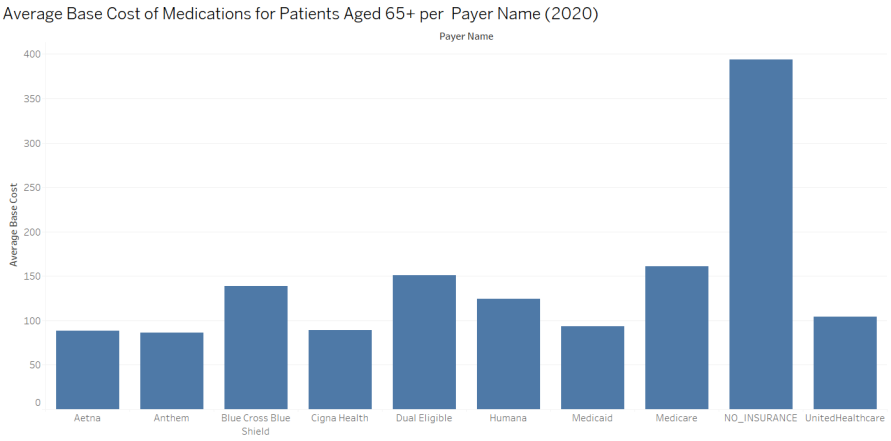


Figure 32: Average Base Cost of Medications by Payer Name for Patients Over 65 in 2021

5.2.13 Query 2: Top Prescribed Medications: Average Dispenses by Gender and Ethnicity (2021)

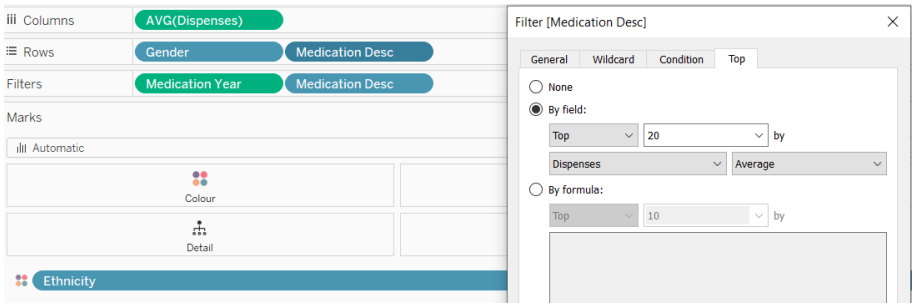


Figure 33: Shelf Configuration for Query 2

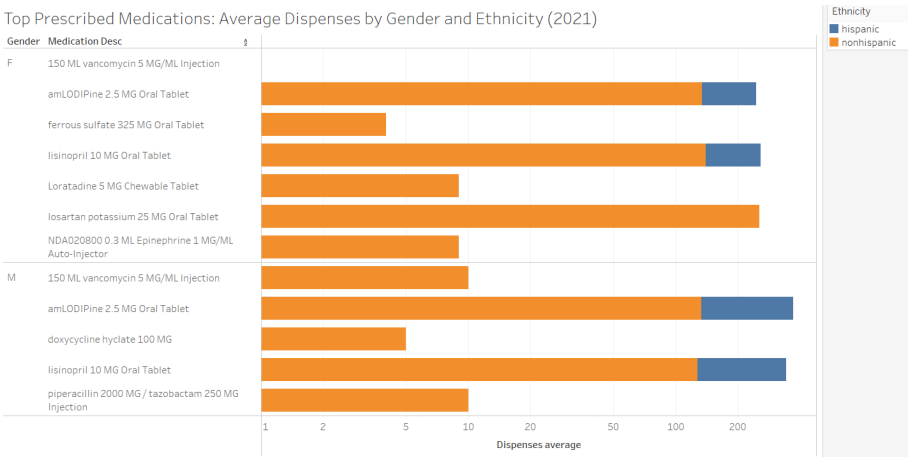


Figure 34: Top Prescribed Medications: Average Dispenses by Gender and Ethnicity (2021)

5.2.14 Procedures Datamart Visualizations

This section presents the visualization for the Procedures datamart. The analysis centers on the distribution of procedures among a specific demographic segment — young female patients — and relies on filtering both by gender and computed age in 2021.

5.2.15 Query: Number of Procedures Performed in 2021 on Female Patients Under 30, Grouped by Procedure Description

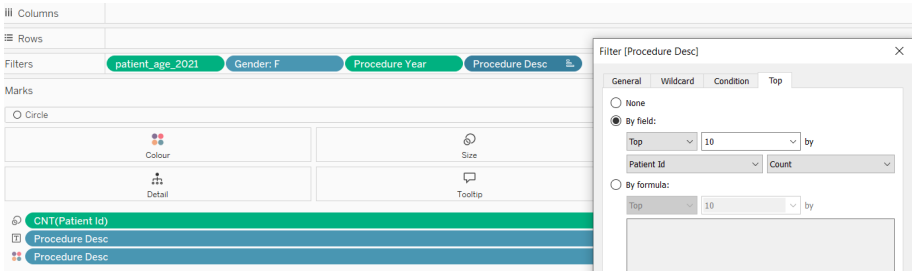


Figure 35: Shelf Configuration for Procedures by Description

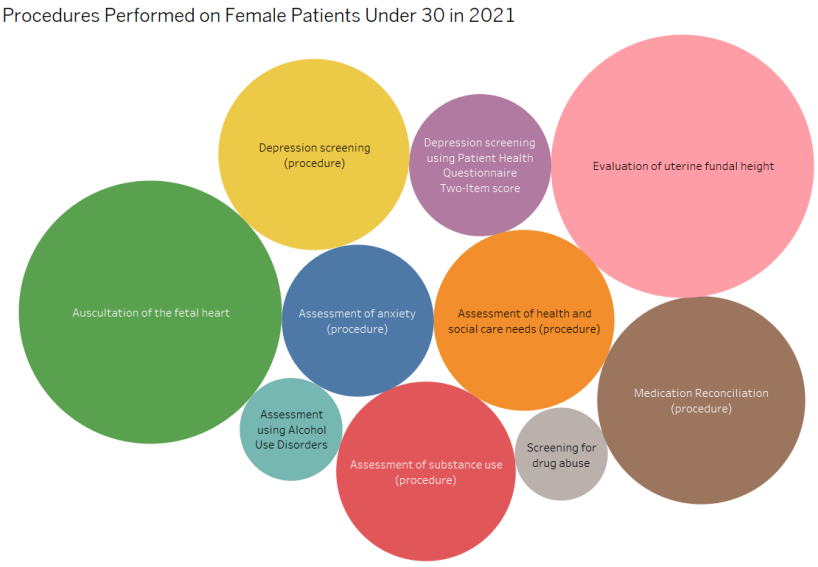


Figure 36: Most Common Procedures for Female Patients Under 30 in 2021

5.2.16 Immunizations Datamart Visualizations

This section presents the visual analysis for the Immunizations datamart. The query focuses on identifying elderly patients who received influenza-related immunizations during the winter season, as well as the associated average base cost.

To support this, a calculated field named **Is Influenza Immunization** was introduced. This field checks whether the immunization description contains the word "influenza", allowing the aggregation to target only flu-related vaccines regardless of code variations.

Listing 10: Calculated Field: Is Influenza Immunization

```
CONTAINS(LOWER([Immunization Desc]), "influenza")
```

The aggregation also relies on:

- **Age filtering**: only patients aged 65 or older in 2021. - **Date filtering**: only immunizations performed during winter months (December, January, February). - **Year grouping**: analysis over the past decade.

5.2.17 Query: Percentage of Patients Aged 65+ Receiving an Influenza Immunization in Winter (2011-2021), and Average Cost

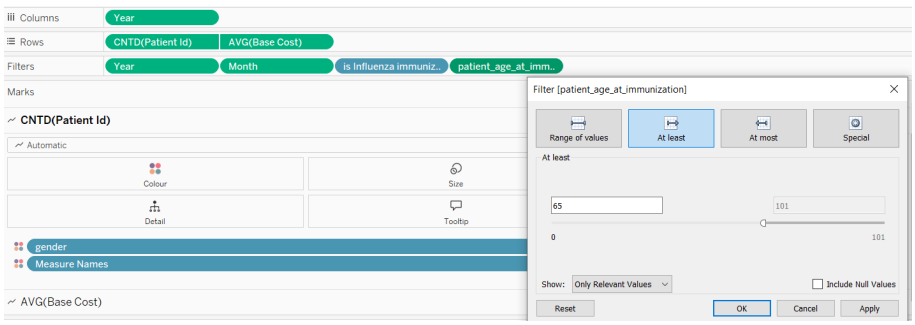


Figure 37: Shelf Configuration for Influenza Immunization Query

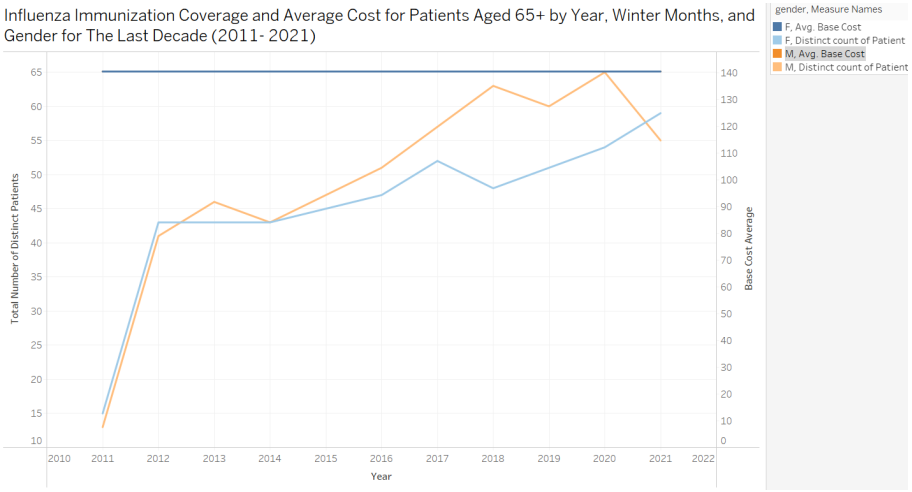


Figure 38: Influenza Immunization Coverage and Cost for Elderly Patients During Winter (2011-2021)

VI Discussion and Conclusion

6.1 Challenges and Limitations

While this project successfully demonstrates the design and implementation of a healthcare data warehouse, several limitations were encountered throughout the process, many of which are rooted in the nature of the dataset and the environment in which it was developed:

- **Synthetic Dataset**: The data was generated using Synthea, a synthetic patient generator. Although useful for simulation purposes, this data lacks the nuance and variability of real-world clinical information. As a result, it was often difficult to assign meaningful context to specific codes, values, or relationships — especially when validating query logic or assessing the representativeness of results.
- **Null Handling in MySQL**: MySQL posed limitations in dealing with NULL values during data transformation. Calculated fields involving aggregates, filters, and conditions often required workarounds to avoid unintended behavior or loss of data during joins or filtering operations.
- **Residual Data Quality Issues**: Despite preprocessing and cleaning efforts, the dataset remained noisy and redundant. This was reflected in the final visualizations, where inconsistent attribute values, duplicated entities, and outlier measures produced charts that are not always representative or analytically sound.
- **Analytical Misalignment**: Since the data is not grounded in a real operational context, some queries—particularly those involving trends, costs, or patient distributions—yielded results that lack real-world interpretability or strategic value.

These challenges are typical in academic or simulated environments but are important to document to frame the analytical results appropriately.

6.2 Conclusion

This project provided a comprehensive walkthrough of designing and implementing a healthcare data warehouse tailored to a set of defined analytical requirements. The process began by identifying eleven decisional queries and deriving six corresponding datamarts using techniques such as attribute tree pruning, fact schema modeling, and star schema design.

Rather than creating physical fact and dimension tables, the warehouse was implemented through SQL views to support a flexible and storage-efficient architecture. The Extract-Transform-Load (ETL) process was conceptually applied by transforming and joining tables within views based on relevant business logic.

To facilitate decision-making, each datamart view was connected to Tableau Desktop. There, visualizations were built using OLAP concepts such as **slicing** (filtering by age group, gender, or year), **dicing** (cross-tabulating dimensions like provider and claim type), **drill-down** (analyzing by month or quarter), and **roll-up** (aggregating by year or patient group).

Despite the constraints of working with synthetic and imperfect data, the project illustrates the full lifecycle of a data warehouse solution — from business requirement identification to schema modeling, SQL-based implementation, and interactive data visualization.

Future extensions of this work could include integration with real clinical data, more advanced ETL pipelines, data cleaning automation, and support for real-time analytics through materialized views or OLAP cubes.