MILESTONE-7

DONE BY
ARAVIND SWAMY
AND UMA
MAHESHWARI
DEIVASIGAMANI

Table of Contents:

Dashboards & KPIs:

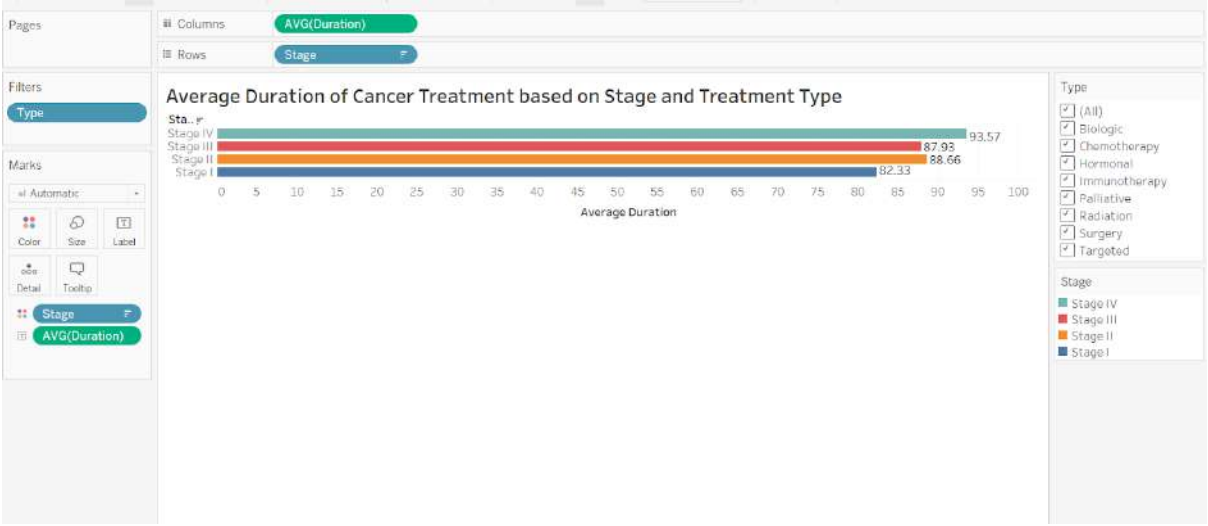
We have chosen the same project idea for both on-prem and cloud projects. In the on-prem project we have 3 fact tables : Treatment Fact Table, Supplies Fact Table and Reviews Fact Table.

In the cloud project we removed the supplies part of it. And we have only two fact tables in it: Treatment Fact Table and Reviews Fact Table.

We have **3 dashboards** for this project, one each for each of the fact table.i.e., one for treatment KPIs, one for reviews KPIs and one for supplies KPIs. In addition to KPIs mentioned in OLAP operations we have also done some extra KPIs

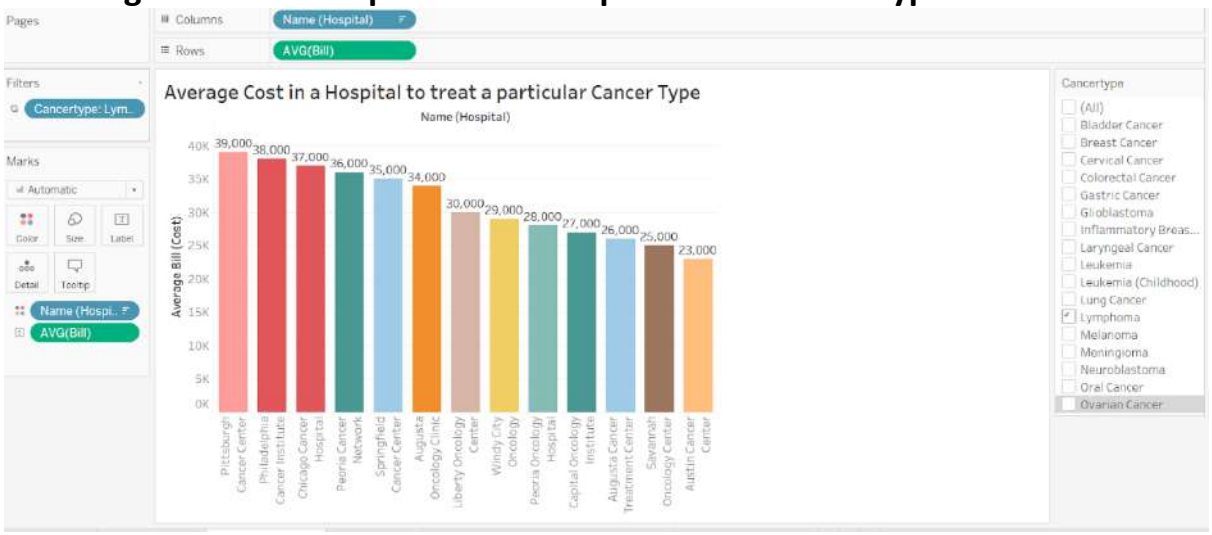
Treatment KPIs & Dashboard:

1. **1.Average Duration of Cancer Treatment based on Stage and Treatment Type:**



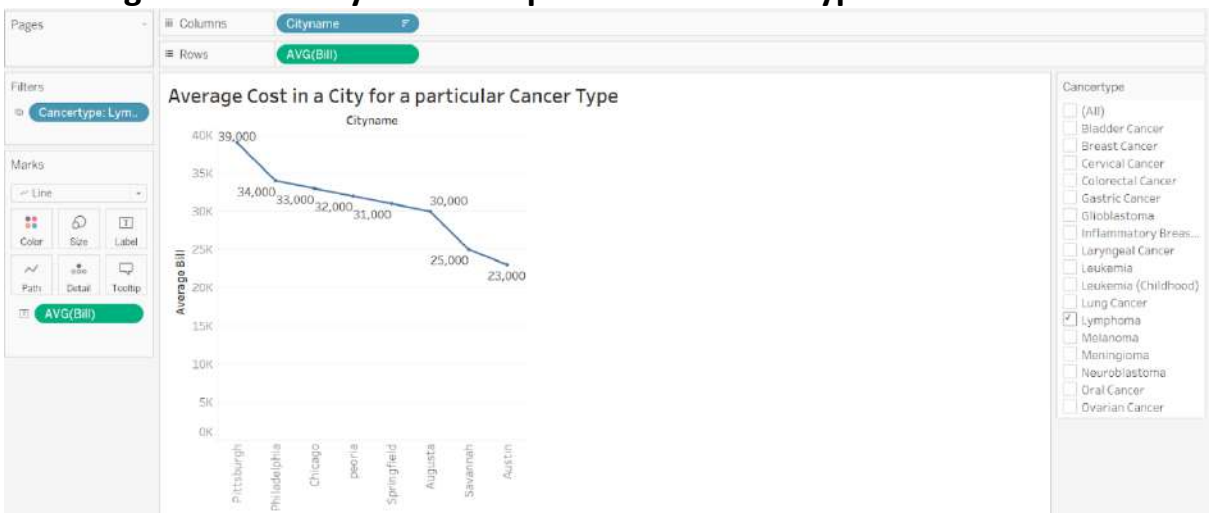
The above graph shows the average duration to treat each stage of cancer using a particular treatment type.(This follows from the OLAP operation mentioned: Treatment duration based on cancer stage (measure involved:Duration))

2.Average Cost in a Hospital to treat a particular Cancer Type



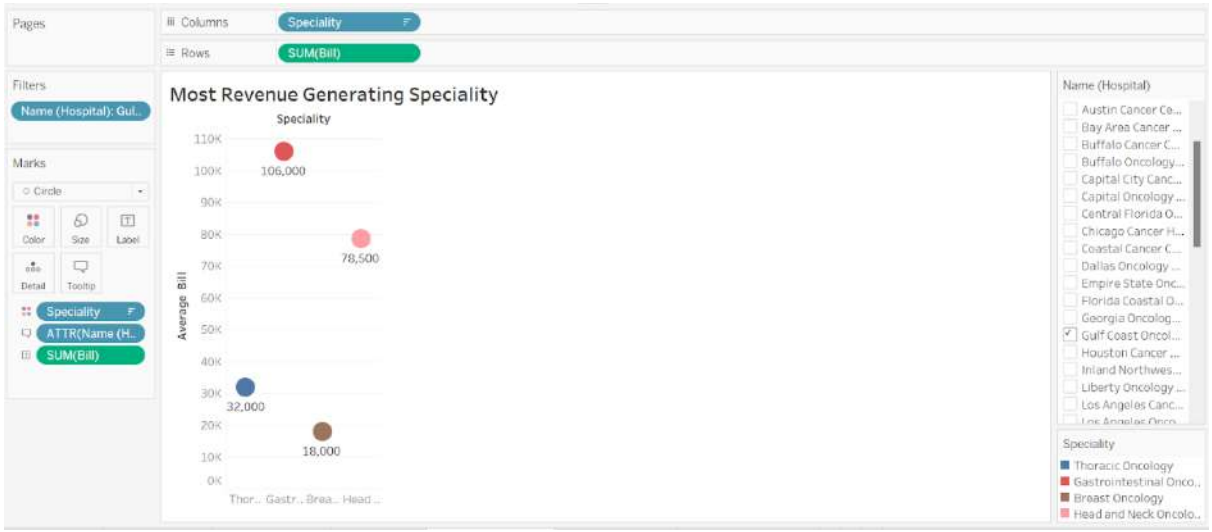
The above graph shows the average cost to treat a particular cancer in a hospital. This could help the patients to take an informed decision and choose a hospital.
(This follows from the OLAP operation mentioned: Average cost in hospital to treat a particular cancer type (measure involved:Bill))

3.Average Cost in a city to treat a particular Cancer Type



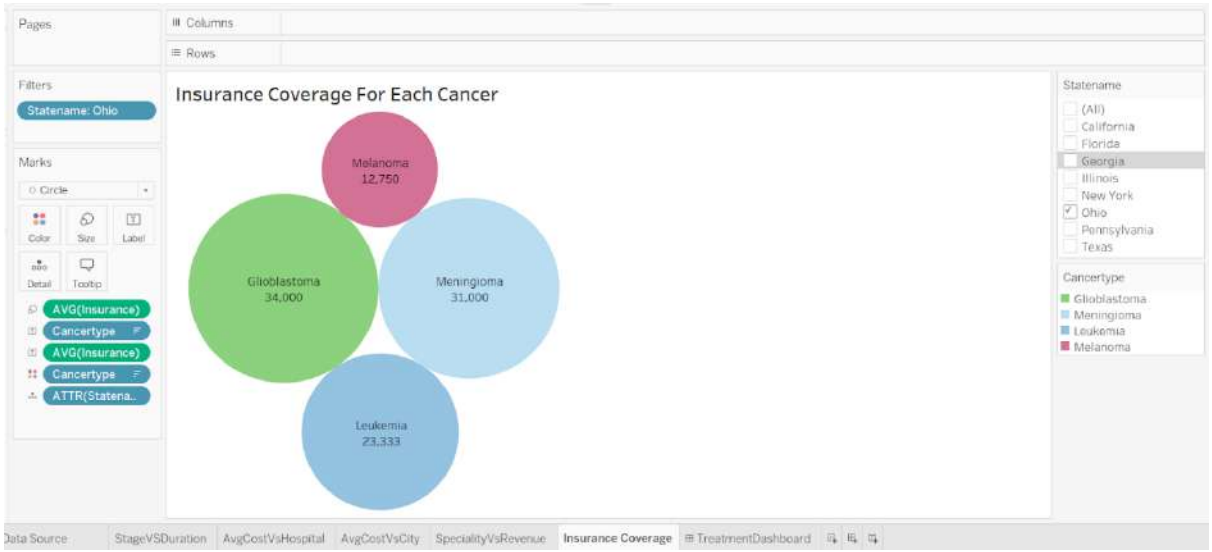
The above graph shows the average cost to treat a particular cancer in a city. This could help the patients to take an informed decision and choose a particular city.
(This follows from the OLAP operation mentioned: Average cost in city to treat a particular cancer type(measure involved:Bill))

4. Most Revenue Generating Speciality



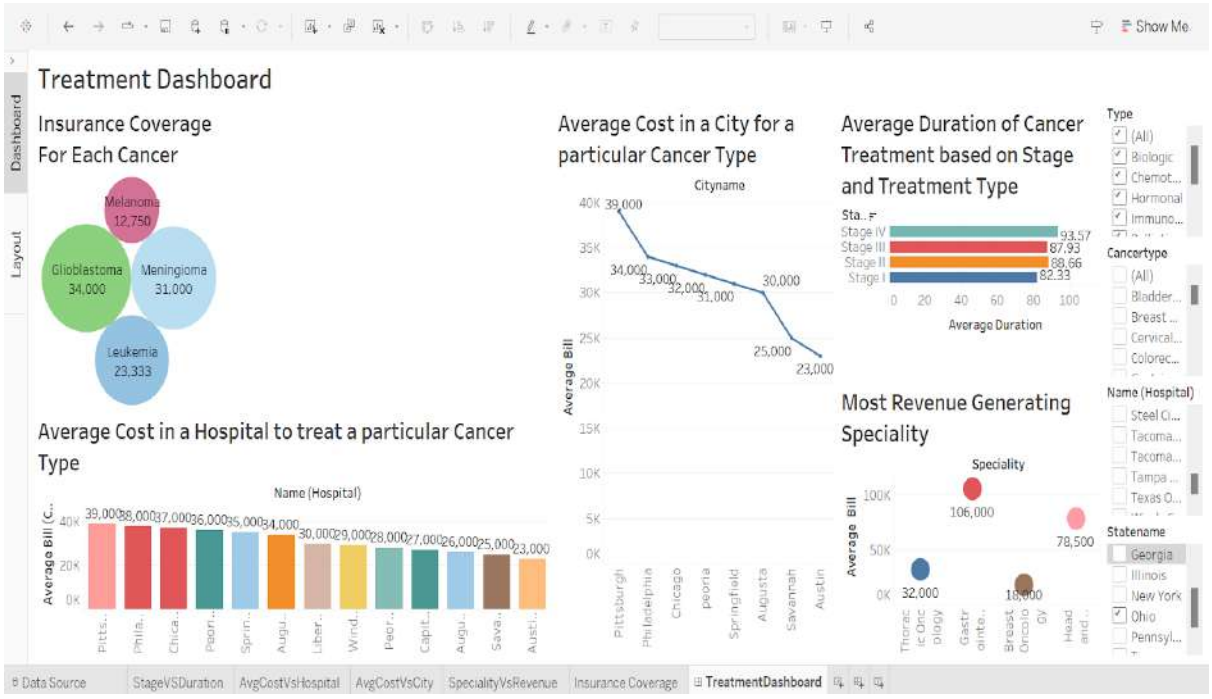
The above graph shows which speciality generates the most revenue for a chosen hospital.(This follows from the OLAP operation mentioned: Which specialty of doctors generates the most revenue(measure involved: Bill))

5. Insurance Coverage For a Cancer Type in a chosen state:



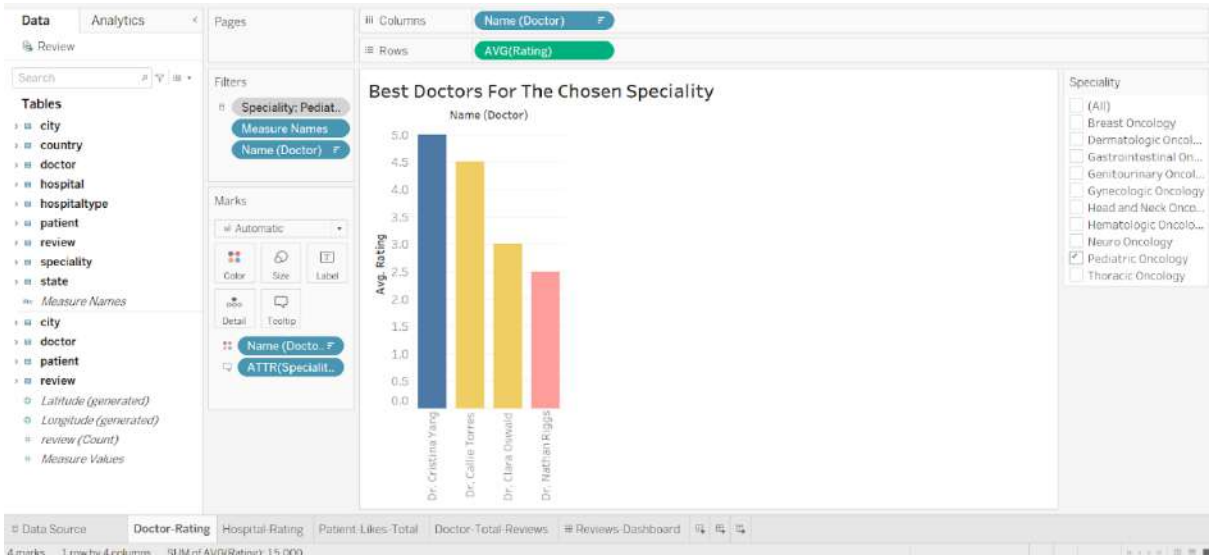
The above graph shows the average insurance coverage for a cancer type in a chosen state. Using this information the patients can choose a state to get treated. (This follows from the OLAP operation mentioned: Sort by highest average insurance coverage based on cancer type(measure involved:insurance))

Treatment Dashboard:



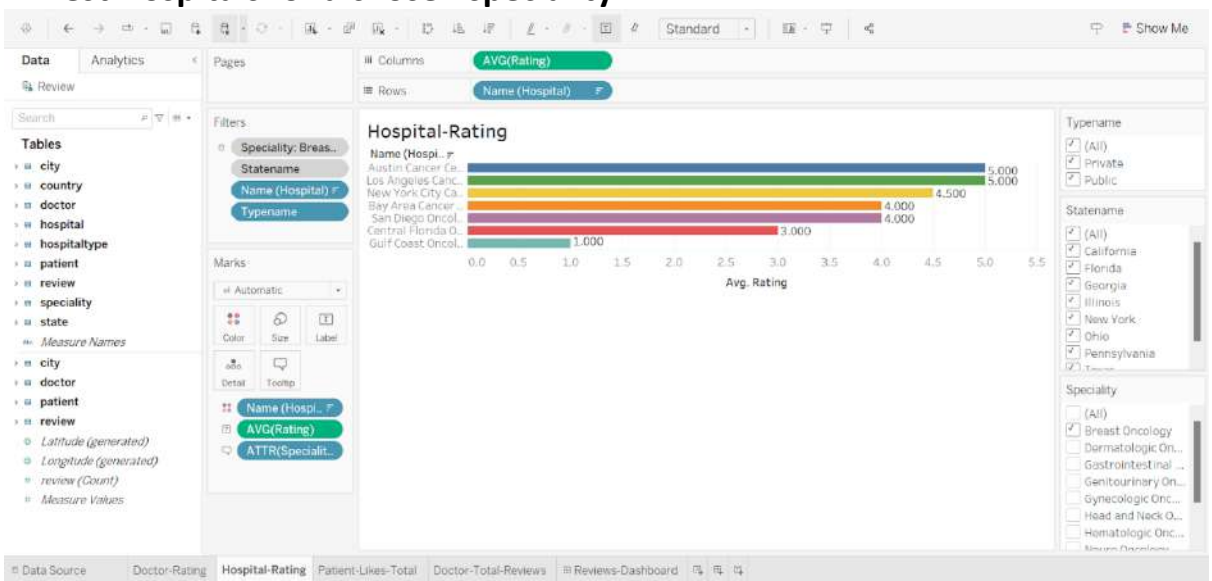
Reviews KPI & Dashboard: (FROM CLOUD)

1.Best Doctors for a chosen speciality based on ratings



The above graph shows the best top doctors available for a chosen speciality based on this info the patients can take a decision. (This follows from the OLAP operation: Sort doctors by highest rating based on specialty (measure involved:rating))

2. Best Hospitals for a chosen speciality:



The above graph shows which hospital is best for treating cancers of a particular speciality based on rating and using the filters we can see the information for each state and hospital type.

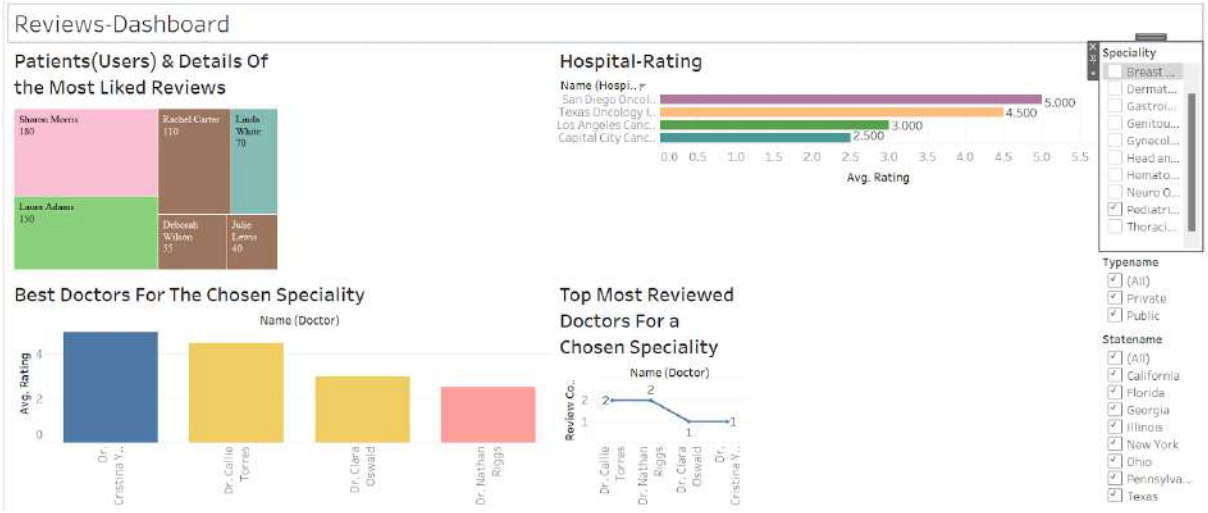
3. Details of the patients with most liked reviews for each speciality

The above graph shows the most liked reviews for a chosen specialty. New users can sort by this and gain valuable information about the reviews. (This follows from the olap operation: Sort patients by the number of likes on their comments (measure involved: review helpfulness score/likes))

4. Most Reviewed Doctors in a chosen specialty (Total Number of reviews for a doctor):

The above graph shows the most reviewed doctors in each speciality. This information can serve as valuable information for patients trying to choose a doctor.(This follows from the OLAP operation: Find the total number of reviews for a specific doctor)

Reviews Dashboard:



Supplies KPIs & Dashboard:

1. Average Spending for Medicine in Each Hospital in a chosen state (Avg Selling price to each hospital):

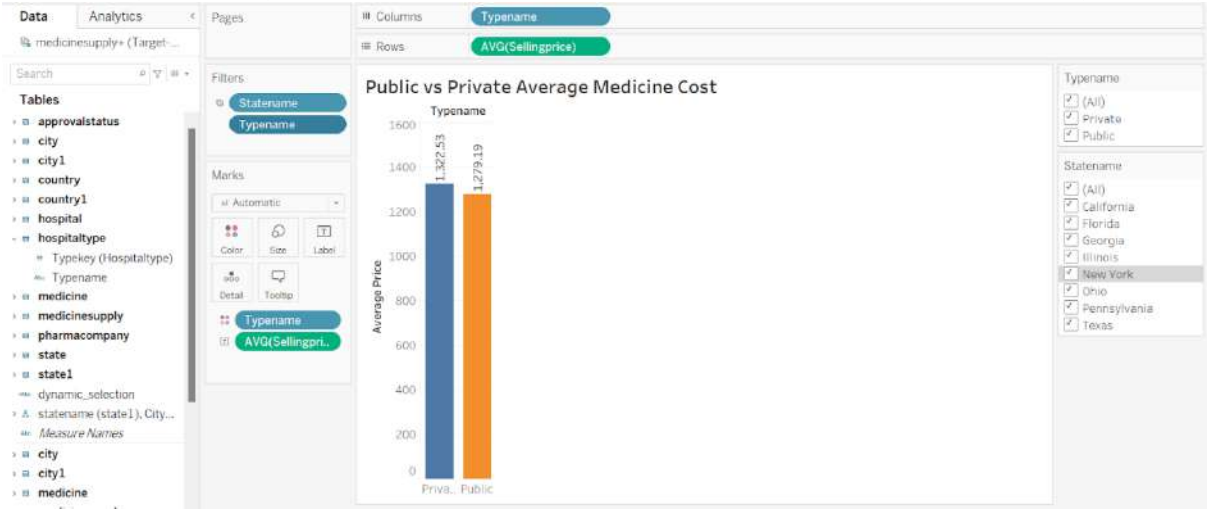
This graph gives the information on how much a specific hospital spends on medicines (follows from OLAP operation mentioned in milestone 3: The average

selling price of a medicine supplied to a particular hospital(measure involved: selling price))

2.Average Price of Medicine in Each State in Public & Private Hospitals:

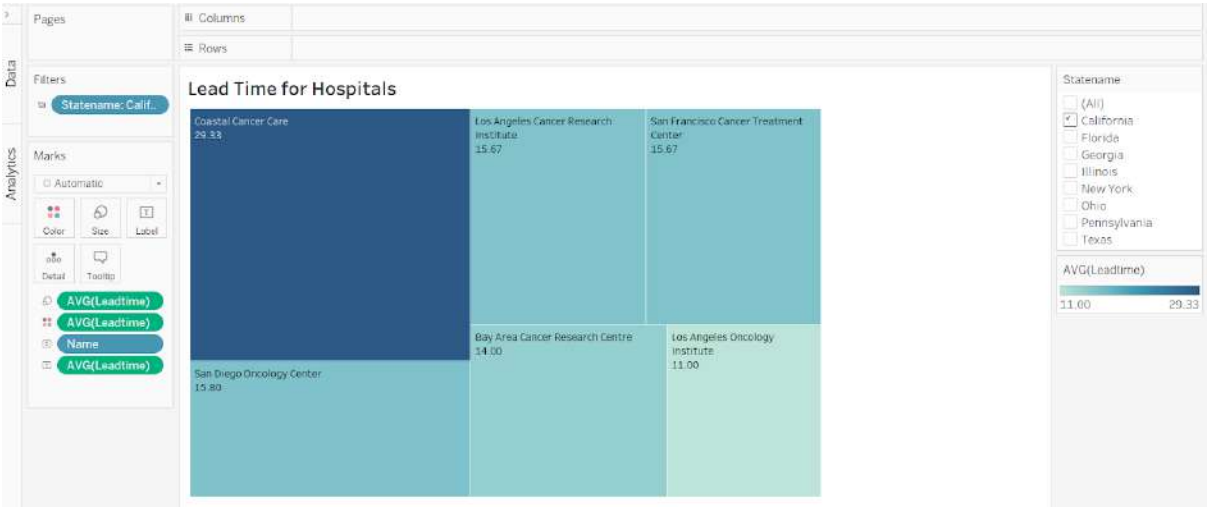
This Graph gives more insights since we can choose the hospital type.

3. Average Selling Price of Medicine in Public & Private Hospitals (Hospital Type):



This graph gives information on how much the selling price for each hospital type (This follows from the OLAP operation mentioned: Average Selling Price of Medicines for a Specic Hospital Type(measure involved: selling price))

4.LeadTime for Hospitals in a chosen state:

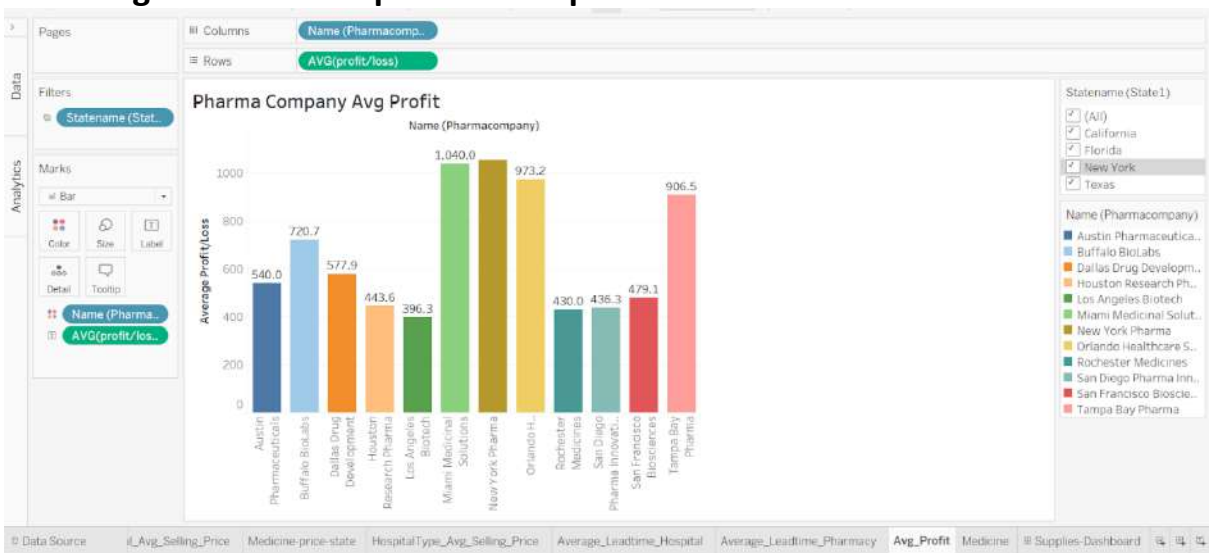


This graph gives the information on the average time it takes for a medicine to be delivered to an hospital

5.Average Lead Time for pharmacies to deliver their medicines:

This graph gives an information on the average time a pharma company takes to deliver its medicines.

6. Average Profit of the pharma companies:

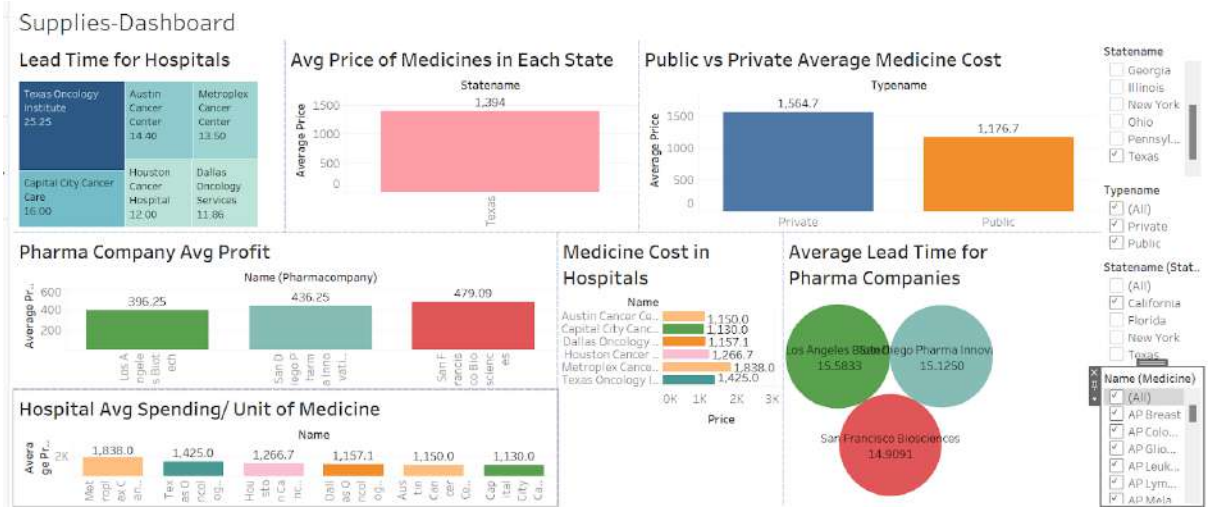


This gives the average profit a pharma company makes with each transaction

7. Avg Cost of a chosen medicine in each hospital where its available:

This gives the varying cost of the same medicine in different hospital.

Supplies Dashboard:



Complete Documentation of both projects

INTRODUCTION :

The healthcare sector, particularly in cancer treatment, faces numerous challenges in transparency, treatment effectiveness, and decision-making for patients. These challenges highlight the importance of providing accurate and accessible information to cancer patients, especially previvors, to make well-informed decisions. The CanCure initiative is a project designed to address these challenges by developing a comprehensive data warehouse system for cancer treatment. This system integrates static reference data (such as hospital details, doctor specialties, and patient demographics) and transactional data (including treatment records, patient reviews, and medicine logistics). We have implemented the ETL process on-prem using Talend and by leveraging AWS tools like Glue, S3, and Athena, this project establishes an effective data pipeline for processing, transforming, and analyzing this information using cloud. The ultimate goal is to enable data-driven decisions that improve treatment success rates, enhance patient satisfaction, and optimize hospital operations.

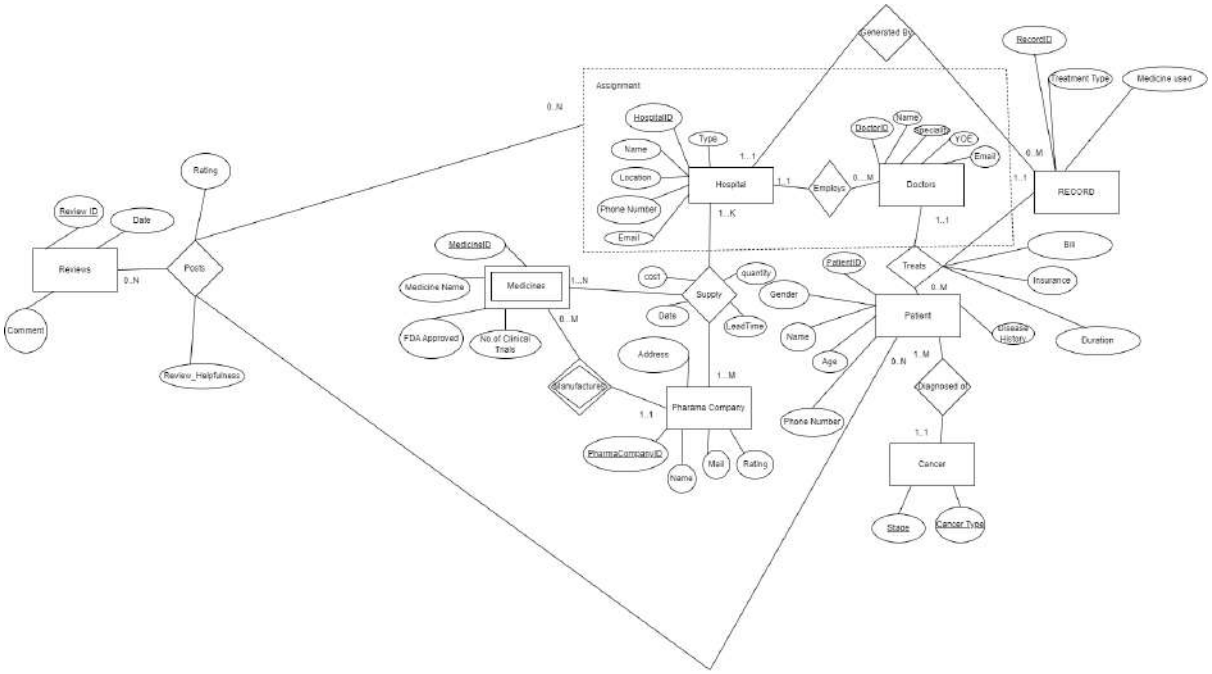
Project 1: On-Prem

Problem Definition:

The healthcare sector, particularly in cancer treatment, faces significant challenges related to transparency, treatment effectiveness,especially when it comes to patient's decision making . The need for knowledge, awareness on various improvements in treatments is largely unknown for a cancer previvor .CanCure is a government-funded initiative designed to regulate and monitor cancer institute hospitals operations while empowering citizens to make well-informed decisions regarding their cancer treatment.This project aims to develop a comprehensive database system for the Cancer Institute, capturing detailed static reference data (including but not limited to hospital details, doctor specialties, and patient demographics,and Pharma company details , medicine details, and cancer details) and transactional data (including but not limited to medical history, treatment records, patient reviews, medicine supply logistics and so on.). By establishing this system, we will enable thorough analysis of treatment success patterns, identification of influential factors, and evaluation of their effects on patients and hospital performance.

Operational Business Context: In the context of cancer treatment, the accuracy of diagnosis, treatment efficacy, and hospital performance significantly affect patient survival and overall experience and patients often lack the information needed to make fully informed decisions. CanCure seeks to bridge this gap by providing data on hospital and doctor performance, treatment success rates, and the effectiveness of specific drugs and treatments by various doctors. Initially, we plan to assess and analyze success rate based on hospital type (public or private),hospital ratings,doctor performance and years of experience, patient reviews and the medicine efficacy to identify areas for improvement.This analysis will empower the stakeholders to make data-driven decisions that enhance cancer care, improve patient satisfaction, and streamline hospital operations.

ERD:



Relational Model of relational database:

Primary Key-Underlined
Foreign key- Italics,#

Hospital (HospitalID, Name, Location, Ph-No, Email)
Primary Key: HospitalID
Foreign Keys: None

Doctor (DoctorID, Name, Specialty, YOE, Email, *#HospitalID*)
Primary Key: DoctorID
Foreign Key: HospitalID (references Hospital.HospitalID)

Patient (PatientID, Gender, Name, Age, ph.No, Symptoms, Diagnosis date, Diseasehistory,*#CancerStage, #CancerType*)
Primary Key: PatientID
Foreign Keys:CancerStage and CancerType together (reference Cancer.Stage and Cancer.CancerType)

Cancer (Stage, CancerType)
Primary Key: Composite key of (Stage, CancerType)
Foreign Keys: None

Record (RecordID, TreatmentType, Status, StartDay, EndDay, Bill, #HospitalID)
Primary Key: RecordID
Foreign Key:HospitalID (references Hospital.HospitalID)

Treats (*#PatientID, #DoctorID, #RecordID,#medicineused*)
Primary Key: Composite key of (PatientID, DoctorID, RecordID, MedicineUsed)
Foreign Keys:

- PatientID (references Patient.PatientID)
- DoctorID (references Doctor.DoctorID)
- RecordID (references Record.RecordID)
- MedicineUsed (references Medicines.MedicineID)

Pharma-Company (CompanyID, Name, Mail, Address, Rating)
Primary Key: CompanyID
Foreign Keys: None

Medicine (MedicineID, Name, FDA, No. of Clinical Trials, *#CompanyID*)
Primary Key: MedicineID
Foreign Key: CompanyID (references PharmaCompany.CompanyID)

Supplies (*#HospitalID, #MedicineID, #PharmaID*, Cost, Quantity, Unit, Date)
Primary Key: Composite key of (HospitalID, MedicineID, PharmaID)
Foreign Keys:

- HospitalID (references Hospital.HospitalID)
- MedicineID (references Medicines.MedicineID)
- PharmaID (references PharmaCompany.CompanyID)

Assignment (*#HospitalID, #DoctorID*)
Primary Key: Composite key of (HospitalID, DoctorID)
Foreign Keys:

- HospitalID (references Hospital.HospitalID)
- DoctorID (references Doctor.DoctorID)

Posts (*#ReviewID, #PatientID, #HospitalID, #DoctorID*)
Primary Key: Composite key of (ReviewID, PatientID, HospitalID, DoctorID)
Foreign Keys:

- ReviewID (references Reviews.ReviewID)
- PatientID (references Patient.PatientID)
- HospitalID (references Hospital.HospitalID)
- DoctorID (references Doctor.DoctorID)

Reviews (ReviewID, Date, Rating, Comments)
Primary Key: ReviewID
Foreign Keys: None

SQL Implementation of the Relational Model:

```
CREATE DATABASE Cancure;

CREATE TABLE Hospital (
   HospitalID VARCHAR(10) PRIMARY KEY,
   Name VARCHAR(255),
   City VARCHAR(255),
   State VARCHAR(50),
   PhoneNo VARCHAR(15),
   Email VARCHAR(255)
);

CREATE TABLE Doctor (
   DoctorID VARCHAR(10) PRIMARY KEY,
   Name VARCHAR(255),
   Speciality VARCHAR(255),
   YOE INT,
   Email VARCHAR(255),
   HospitalID VARCHAR(10) REFERENCES Hospital(HospitalID)
);

CREATE TABLE Cancer (
   Stage VARCHAR(255),
   CancerType VARCHAR(255),
   PRIMARY KEY (Stage, CancerType)
);

CREATE TABLE Patient (
   PatientID VARCHAR(10) PRIMARY KEY,
   Gender VARCHAR(10),
   Name VARCHAR(255) NOT NULL,
   Age INT,
   PhoneNo VARCHAR(15),
   Symptoms TEXT,
   DiagnosisDate DATE,
   CancerStage VARCHAR(255),
   CancerType VARCHAR(255),
   FOREIGN KEY (CancerStage, CancerType) REFERENCES Cancer(Stage, CancerType)
);
```

```
CREATE TABLE Record (
   RecordID VARCHAR(10) PRIMARY KEY,
   TreatmentType VARCHAR(255),
   Status VARCHAR(255),
   StartDay DATE,
   EndDay DATE,
   Bill NUMERIC(10, 2),
   HospitalID VARCHAR(10) REFERENCES Hospital(HospitalID)
);

CREATE TABLE PharmaCompany (
   CompanyID VARCHAR(10) PRIMARY KEY,
   Name VARCHAR(255),
   Mail VARCHAR(255),
   City VARCHAR(100),
   State VARCHAR(100),
   Rating INT CHECK (Rating >= 1 AND Rating <= 5)
);

CREATE TABLE Medicines (
   MedicineID VARCHAR(10) PRIMARY KEY,
   Name VARCHAR(255),
   FDA BOOLEAN,
   NoOfClinicalTrials INT,
   Cost NUMERIC(10, 2),
   PharmaID VARCHAR(10) REFERENCES PharmaCompany(CompanyID)
);

CREATE TABLE Treats (
   PatientID VARCHAR(10) REFERENCES Patient(PatientID),
   DoctorID VARCHAR(10) REFERENCES Doctor(DoctorID),
   RecordID VARCHAR(10) REFERENCES Record(RecordID),
   MedicineUsed VARCHAR(10) REFERENCES Medicines(MedicineID),
   PRIMARY KEY (PatientID, DoctorID, RecordID, MedicineUsed)
);

CREATE TABLE Supplies (
   HospitalID VARCHAR(10) REFERENCES Hospital(HospitalID),
   MedicineID VARCHAR(10) REFERENCES Medicines(MedicineID),
```

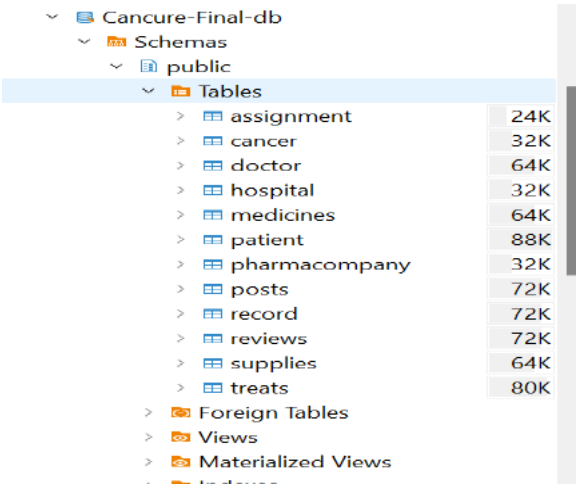
```
    PharmaID VARCHAR(10) REFERENCES PharmaCompany(CompanyID),
    CostPerUnit NUMERIC(10, 2),
    Quantity INT,
    SupplyDate DATE,
    PRIMARY KEY (HospitalID, MedicineID, PharmaID)
);

CREATE TABLE Assignment (
    HospitalID VARCHAR(10) REFERENCES Hospital(HospitalID),
    DoctorID VARCHAR(10) REFERENCES Doctor(DoctorID),
    PRIMARY KEY (HospitalID, DoctorID)
);

CREATE TABLE Reviews (
    ReviewID VARCHAR(10) PRIMARY KEY,
    Date DATE,
    Rating INT,
    Comments TEXT
);

CREATE TABLE Posts (
    ReviewID VARCHAR(10) REFERENCES Reviews(ReviewID),
    PatientID VARCHAR(10) REFERENCES Patient(PatientID),
    HospitalID VARCHAR(10) REFERENCES Hospital(HospitalID),
    DoctorID VARCHAR(10) REFERENCES Doctor(DoctorID),
    PRIMARY KEY (ReviewID, PatientID, HospitalID, DoctorID)
);
```

Data Population Methodology:

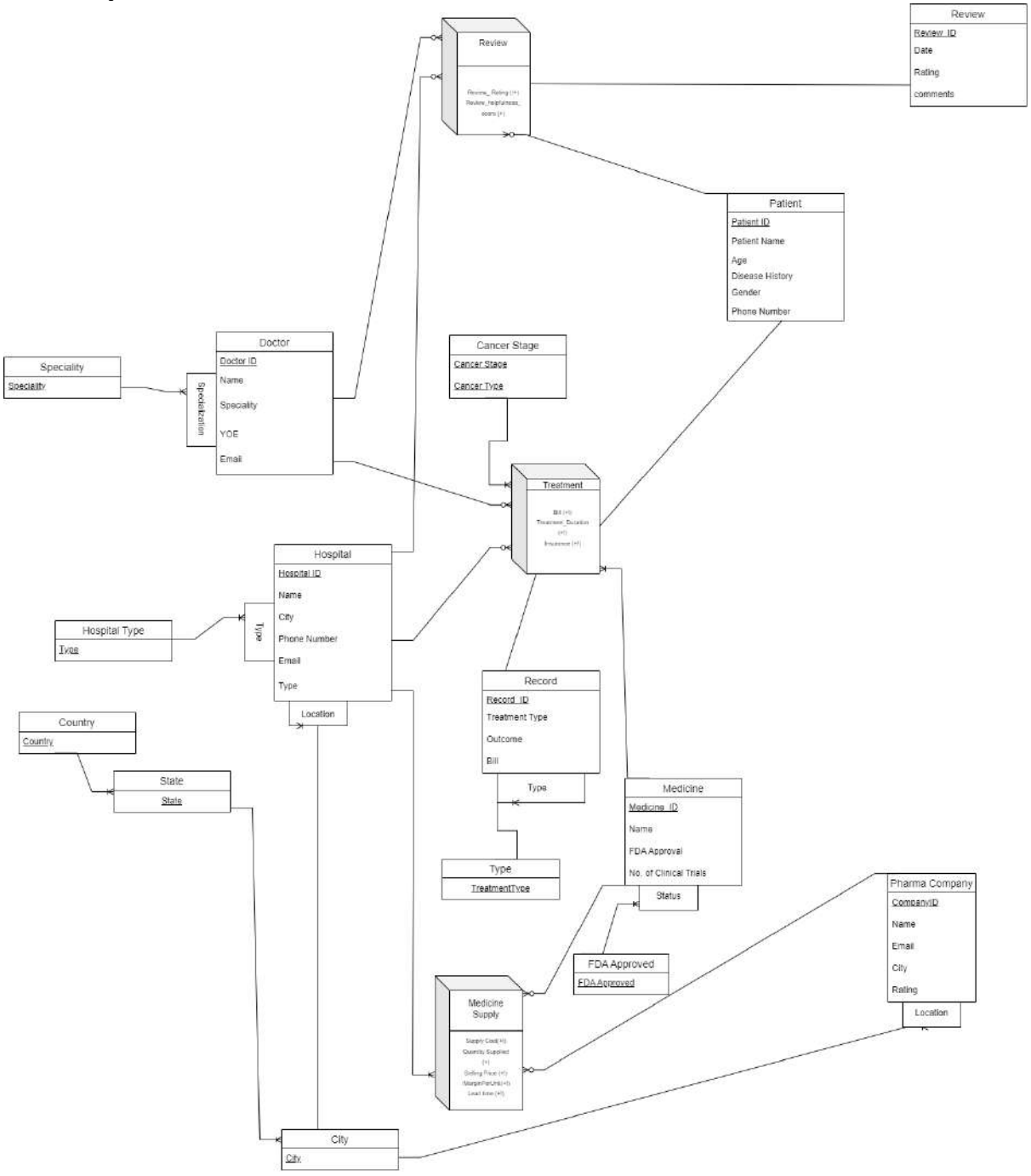
We generated our data using various LLM in batches and filled the static data like Hospital,Doctor,Cancer,PharmaCompany and Medicines first and then using that static data we generated the data for our transactional data like Record,Treats,Supplies.

And with this generated data we populated the data into our tables in our database name Cancure.

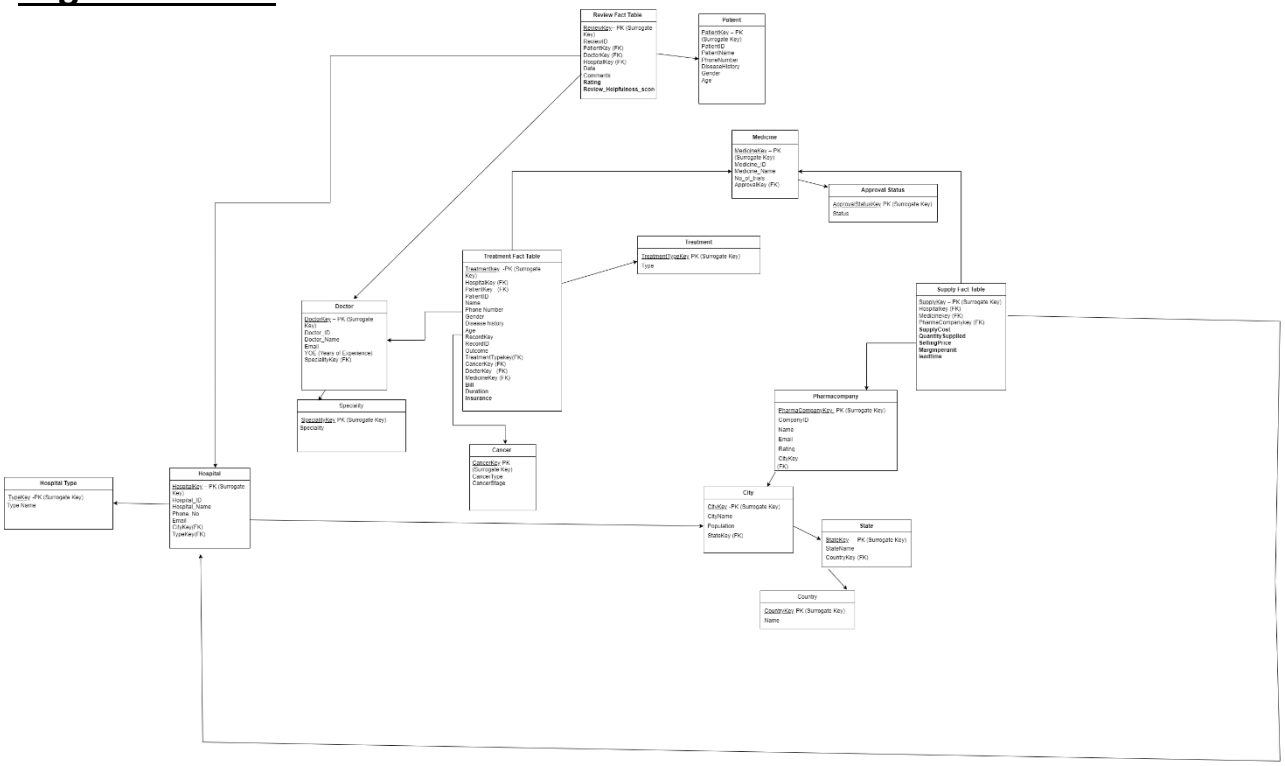
The database is set up in the dbeaver and it is named Cancure and the database totally has 12 tables.



Conceptual Model:



Logical Model:



DB Implementation:

Hospital(HospitalKey,HospitalID,Name,Phone Number, Email, #citykey,#Typekey)
Primary Key: HospitalKey
Foreign Keys:

- #citykey references City.CityKey
- #Typekey references HospitalType.TypeKey
-

City(CityKey, CityName,Population, #StateKey)
Primary Key: CityKey
Foreign Keys: #StateKey references State.StateKey

State(StateKey, StateName,#CountryKey)
Primary Key: StateKey
Foreign Keys: #CountryKey references Country.CountryKey
Country(CountryKey, Name)
Primary Key: CountryKey

HospitalType(TypeKey, TypeName)
Primary Key: TypeKey
Speciality(SpecialityKey,Speciality)
Primary Key: SpecialityKey
Doctor(DoctorKey, DoctorID, Name,Email, YOE,#SpecialityKey)
Primary Key: DoctorKey
Foreign Keys: #SpecialityKey references Speciality.SpecialityKey
Patient(PatientKey,PatientID, Name, Ph.NO,Gender, DiseaseHistory, Age)
Primary Key: PatientKey
PharmaCompany (PharmaCompanyKey, CompanyID, Name, Email, Rating,#CityKey)
Primary Key: PharmaCompanyKey
Foreign Keys: #CityKey references City.CityKey
ApprovalStatus(ApprovalStatusKey, Status)
Primary Key: ApprovalStatusKey
Medicine(MedicineKey,MedicineID, Name, No.of Trials, #ApprovalStatuskey)
Primary Key: MedicineKey
Foreign Keys: #ApprovalStatuskey references ApprovalStatus.ApprovalStatusKey
TreatmentType(TypeKey,Type)
Primary Key: TypeKey
CancerType(CancerKey,CancerType,Stage)
Primary Key: CancerKey

Fact Tables:
Review(Review key,#Patientkey,#DoctorKey,#Hospitalkey,
ReviewID,Date,Comments,Rating, review_helpfulness_score)
Primary Key: Reviewkey
Foreign Keys:

- Patientkey references Patient.PatientKey
- DoctorKey references Doctor.DoctorKey
- Hospitalkey references Hospital.HospitalKey

Treatment(TreatmentKey, #Hospitalkey,#PatientKey,PatientID, Name,
Ph.NO,Gender, DiseaseHistory, AgeGroupKey, RecordKey,RecordID,
,Outcome,TreatmentTypeKey,#Cancerkey,#DoctorKey,Medicinekey,*Bill,Duration, insurance,dosage*)
Primary Key: TreatmentKey
Foreign key:

- Cancerkey references CancerType.Cancerkey
- DoctorKey references Doctor.DoctorKey
- PatientKey references Patient.PatientKey
- HospitalKey references Hospital.HospitalKey

MedicineSupply(SupplyKey,#Hospitalkey,#Medicinekey,#PharmaCompanykey,SupplyCost,Quantity,SellingPrice,MarginPerUnit, lead_time)
Primary Key: SupplyKey
Foreign Keys:

- Hospitalkey references Hospital.Hospitalkey
- Medicinekey references Medicine.Medicinekey
- PharmaCompanykey references PharmaCompany.Companykey

SQL Implementation of Warehouse Design:

```
CREATE TABLE Country (
  CountryKey INTEGER PRIMARY KEY,
  Name VARCHAR(255) NOT NULL
);
CREATE TABLE State (
  StateKey INTEGER PRIMARY KEY,
  StateName VARCHAR(255) NOT NULL,
  CountryKey INTEGER NOT NULL,
  FOREIGN KEY (CountryKey) REFERENCES Country(CountryKey)
);
CREATE TABLE City (
  CityKey INTEGER PRIMARY KEY,
  CityName VARCHAR(255) NOT NULL,
  Population INTEGER,
  StateKey INTEGER NOT NULL,
  FOREIGN KEY (StateKey) REFERENCES State(StateKey)
);
CREATE TABLE HospitalType (
  TypeKey INTEGER PRIMARY KEY,
  TypeName VARCHAR(255) NOT NULL
);
CREATE TABLE Hospital (
  HospitalKey INTEGER PRIMARY KEY,
  HospitalID VARCHAR(255) NOT NULL,
  Name VARCHAR(255) NOT NULL,
  Phone_Number VARCHAR(255),
  Email VARCHAR(255),
  CityKey INTEGER NOT NULL,
  TypeKey INTEGER NOT NULL,
  FOREIGN KEY (CityKey) REFERENCES City(CityKey),
  FOREIGN KEY (TypeKey) REFERENCES HospitalType(TypeKey)
```

```
);
CREATE TABLE PharmaCompany (
  PharmaCompanyKey INTEGER PRIMARY KEY,
  CompanyID VARCHAR(255) NOT NULL,
  Name VARCHAR(255) NOT NULL,
  Email VARCHAR(255),
  Rating DECIMAL(3,2),
  CityKey INTEGER NOT NULL,
  FOREIGN KEY (CityKey) REFERENCES City(CityKey)
);
CREATE TABLE CancerType (
  CancerKey INT PRIMARY KEY,
  CancerType VARCHAR(100) NOT NULL,
  Stage VARCHAR(50) NOT NULL
);
CREATE TABLE Speciality (
  SpecialityKey INT PRIMARY KEY,
  Speciality VARCHAR(100) NOT NULL
);
CREATE TABLE Doctor (
  DoctorKey INT PRIMARY KEY,
  DoctorID VARCHAR(50) NOT NULL,
  Name VARCHAR(100) NOT NULL,
  Email VARCHAR(100) NOT NULL,
  YOE INT NOT NULL,  -- Years of Experience
  SpecialityKey INT,
  FOREIGN KEY (SpecialityKey) REFERENCES Speciality(SpecialityKey)
);
CREATE TABLE ApprovalStatus (
  ApprovalStatusKey INT PRIMARY KEY,
  Status VARCHAR(100) NOT NULL
);
CREATE TABLE TreatmentType (
  TypeKey INT PRIMARY KEY,
  Type VARCHAR(50) NOT NULL
);
```

```
CREATE TABLE Medicine (
  MedicineKey INT PRIMARY KEY,
  MedicineID VARCHAR(50) NOT NULL,
  Name VARCHAR(100) NOT NULL,
  No_of_Trials INT,
  ApprovalStatusKey INT,
  FOREIGN KEY (ApprovalStatusKey) REFERENCES
ApprovalStatus(ApprovalStatusKey)
);
CREATE TABLE Patient (
  PatientKey INT PRIMARY KEY,
  PatientID VARCHAR(50) NOT NULL,
  Name VARCHAR(100) NOT NULL,
  PhNo VARCHAR(20),
  Gender VARCHAR(10),
  DiseaseHistory TEXT,
  Age INT
);
CREATE TABLE Record (
  RecordKey INT PRIMARY KEY,
  RecordID VARCHAR(50) NOT NULL,
  Outcome VARCHAR(10),
  TreatmentTypeKey INT,
  FOREIGN KEY (TreatmentTypeKey) REFERENCES TreatmentType(TypeKey)

);
```

--Fact Tables

```
CREATE TABLE Review (
  ReviewKey INT PRIMARY KEY,
  PatientKey INT,
  DoctorKey INT,
  HospitalKey INT,
  ReviewID VARCHAR(50),
  Date DATE,
  Comments TEXT,
  Rating INT CHECK (Rating BETWEEN 1 AND 5),
  ReviewHelpfulnessScore INT,
  FOREIGN KEY (PatientKey) REFERENCES Patient(PatientKey),
  FOREIGN KEY (DoctorKey) REFERENCES Doctor(DoctorKey),
```

```
  FOREIGN KEY (HospitalKey) REFERENCES Hospital(HospitalKey)
);
CREATE TABLE MedicineSupply (
  SupplyKey INT PRIMARY KEY,
  HospitalKey INT,
  MedicineKey INT,
  PharmaCompanyKey INT,
  SupplyCost DECIMAL(10, 2),
  Quantity INT,
  SellingPrice DECIMAL(10, 2),
  MarginPerUnit DECIMAL(10, 2),
  LeadTime INT,
  FOREIGN KEY (HospitalKey) REFERENCES Hospital(HospitalKey),
  FOREIGN KEY (MedicineKey) REFERENCES Medicine(MedicineKey),
  FOREIGN KEY (PharmaCompanyKey) REFERENCES
PharmaCompany(PharmaCompanyKey)
);
CREATE TABLE TreatmentFact (
  TreatmentKey INT PRIMARY KEY,
  HospitalKey INT,
  PatientKey INT,
  PatientID VARCHAR(50),
  Name VARCHAR(100),
  PhNo VARCHAR(15),
  Gender VARCHAR(10),
  DiseaseHistory TEXT,
  Age INT,
  RecordKey INT,
  RecordID VARCHAR(50),
  Outcome VARCHAR(10),
  TreatmentTypeKey INT,
  CancerKey INT,
  DoctorKey INT,
  MedicineKey INT,
  Bill DECIMAL(10, 2),
  Duration INT,
  Insurance DECIMAL(10, 2),
  FOREIGN KEY (HospitalKey) REFERENCES Hospital(HospitalKey),
  FOREIGN KEY (PatientKey) REFERENCES Patient(PatientKey),
  FOREIGN KEY (RecordKey) REFERENCES Record(RecordKey),
  FOREIGN KEY (TreatmentTypeKey) REFERENCES TreatmentType(TypeKey),
```

```
  FOREIGN KEY (DoctorKey) REFERENCES Doctor(DoctorKey),
  FOREIGN KEY (MedicineKey) REFERENCES Medicine(MedicineKey)
);
```

Primary Events:

Taking in patient records and Hospitals ordering medicines from pharma companies and patients posting their reviews would be the primary events for the project because these are the critical first steps in ensuring that the system has accurate and comprehensive data on patient transactions,treatments,supply interactions and billing,payments, and review ratings which will drive the subsequent analysis and reporting processes

OLAP Operations:

Treatment duration based on cancer stage
ROLLUP(Treatment, Cancer Stage, AVG(Duration))

Average cost in hospital to treat a particular cancer type
ROLLUP(Treatment, Hospital→Hospital, Cancer Type, AVG(Bill))

Average cost in a city to treat a particular cancer type
ROLLUP(Treatment, Hospital→City, Cancer Type, AVG(Cost))

Which specialty of doctors generates the most revenue
ROLLUP(Treatment, Doctor→Specialty, SUM(Bill))

Sort by highest average insurance coverage based on cancer type
SORT(Treatment, Cancertype, (AVG(Insurance) DESC))

The average selling price of a medicine supplied to a particular hospital
ROLLUP(MedicineSupply, Hospital, Medicine, AVG(Selling Price))

Average Selling Price of Medicines for a Specific Hospital Type
RolledUpCube<–ROLLUP(MedicineSupply, Hospital—>Type, Medicine, AVG(Selling Price))

SLICE(RolledUpCube, Type = 'SpecificType')

Find the total number of reviews for a specific doctor
SlicedReviewCube<—SLICE(Review, Doctor = 'SpecificDoctorID')
ROLLUP(Review, COUNT(ReviewID))

Sort doctors by highest rating based on specialty
RolledUpReview<—ROLLUP(Review, Doctor→Specialty, AVG(Rating))
SORT(RolledUpReview, Specialty, (AVG(Rating) DESC))

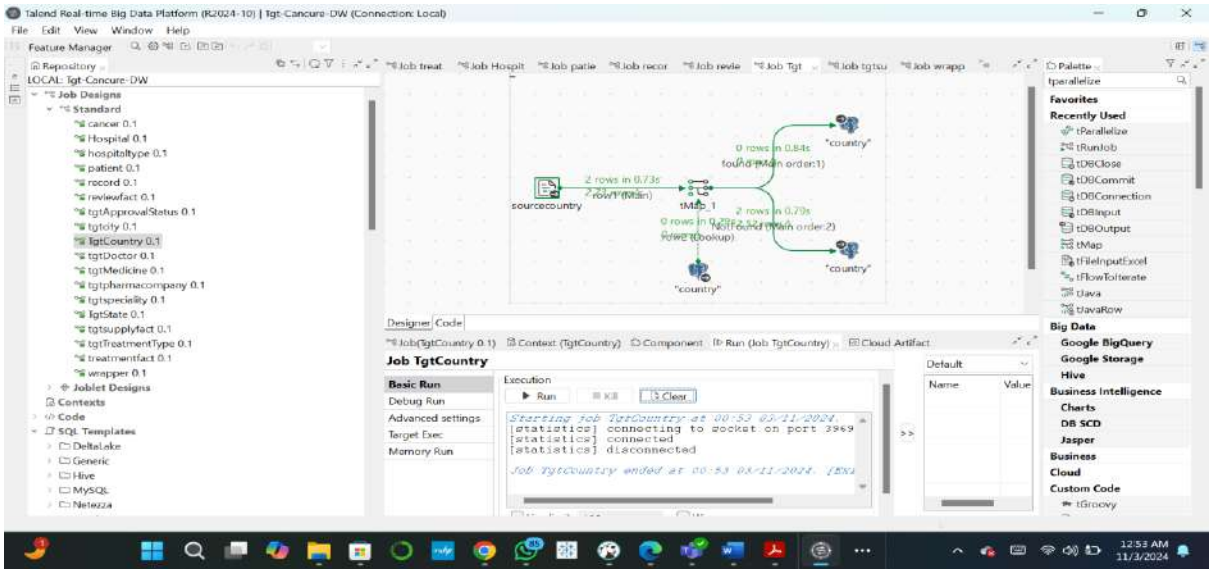
Sort patients by the number of likes on their comments
RolledUpReviews<–ROLLUP(Review, Patient, SUM(ReviewHelpfulnessScore)
SORT(RolledUpReview, Patient, (SUM(ReviewHelpfulnessScore) DESC))

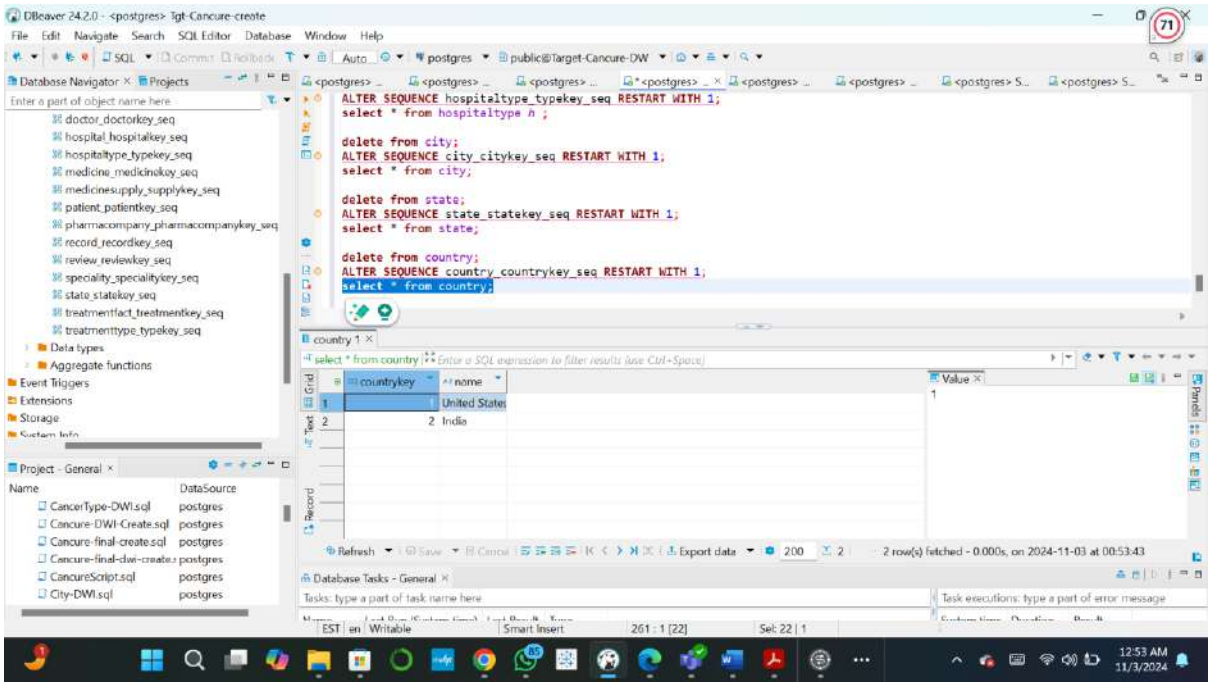
Talend Implementation:

Data Flow and Control Flow
Insert flow:

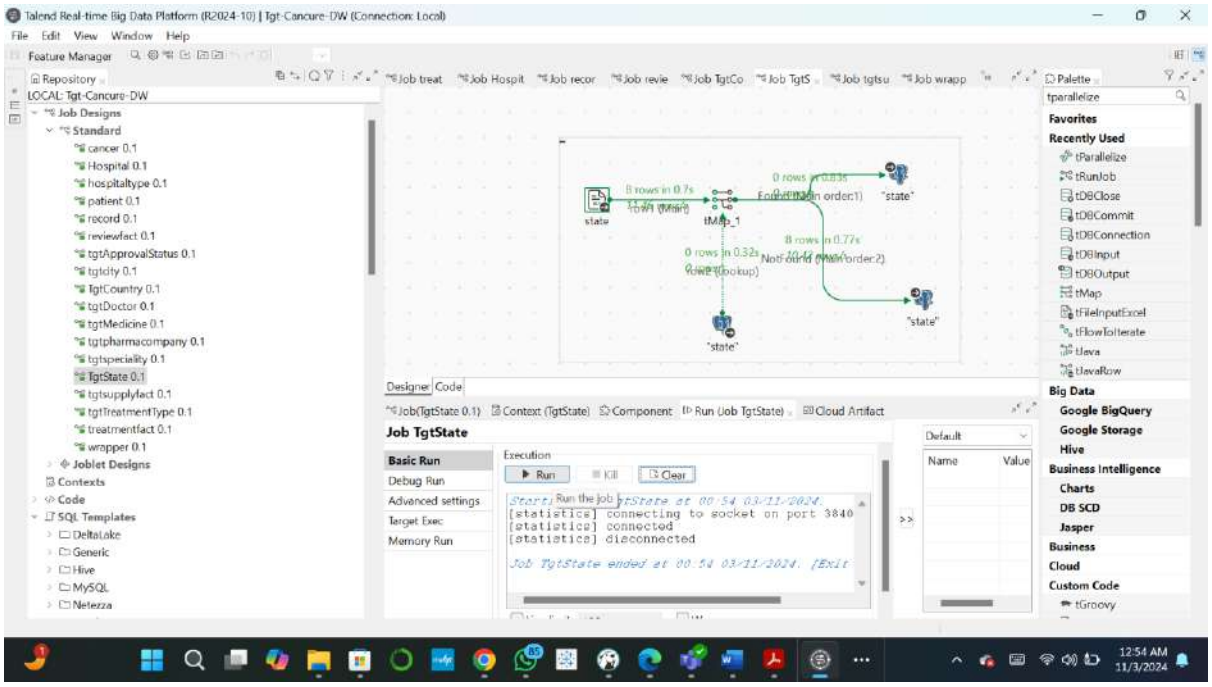
The Following Diagram shows the insert flow execution and proof of execution for each table in the data warehouse, and the screenshots below also show surrogate key has been implemented for all tables.
Not Found-Insert
Found-Update

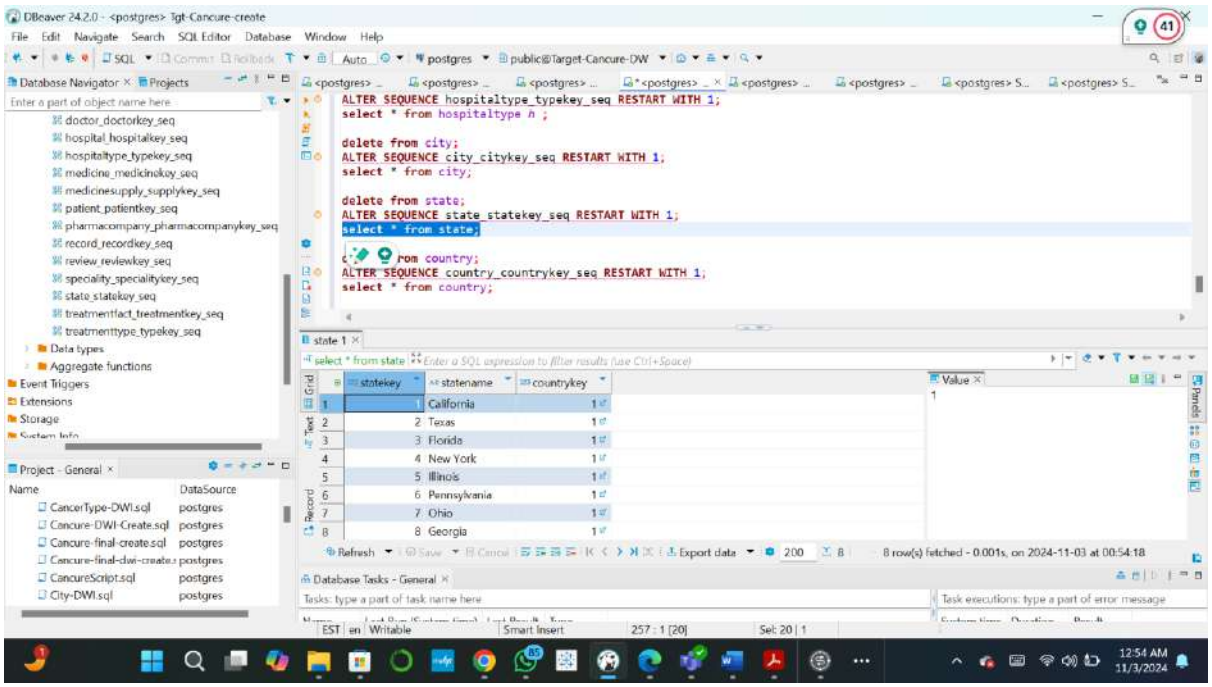
1. Country:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
File Edit View Window Help
Feature Manager
Repository
LOCAL: Tgt-Cancure-DW
Job Designs
Standard
cancer 0.1
Hospital 0.1
hospitaltype 0.1
patient 0.1
record 0.1
reviewfact 0.1
tgtApprovalStatus 0.1
tgtcity 0.1
tgtCountry 0.1
tgtDoctor 0.1
tgtMedicine 0.1
tgtpharmacompany 0.1
tgtspeciality 0.1
tgtState 0.1
tgtsupplyfact 0.1
tgtTreatmentType 0.1
treatmentfact 0.1
wrapper 0.1
Joblet Designs
Contexts
Code
SQL Templates
DeltaLake
Generic
Hive
MySQL
Netezza
sourcecountry
tMap_1
"country"
Designer Code
Job TgtCountry
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Clear
[statistics] connecting to socket on port 3969
[statistics] connected
[statistics] disconnected
Palette
tparallelize
Favorites
Recently Used
tParallelize
tRunJob
tDBClose
tDBCommit
tDBConnection
tDBInput
tDBOutput
tMap
tFileInputExcel
tFlowToIterate
tJava
tJavaRow
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
tGroovy
12:53 AM
11/3/2024

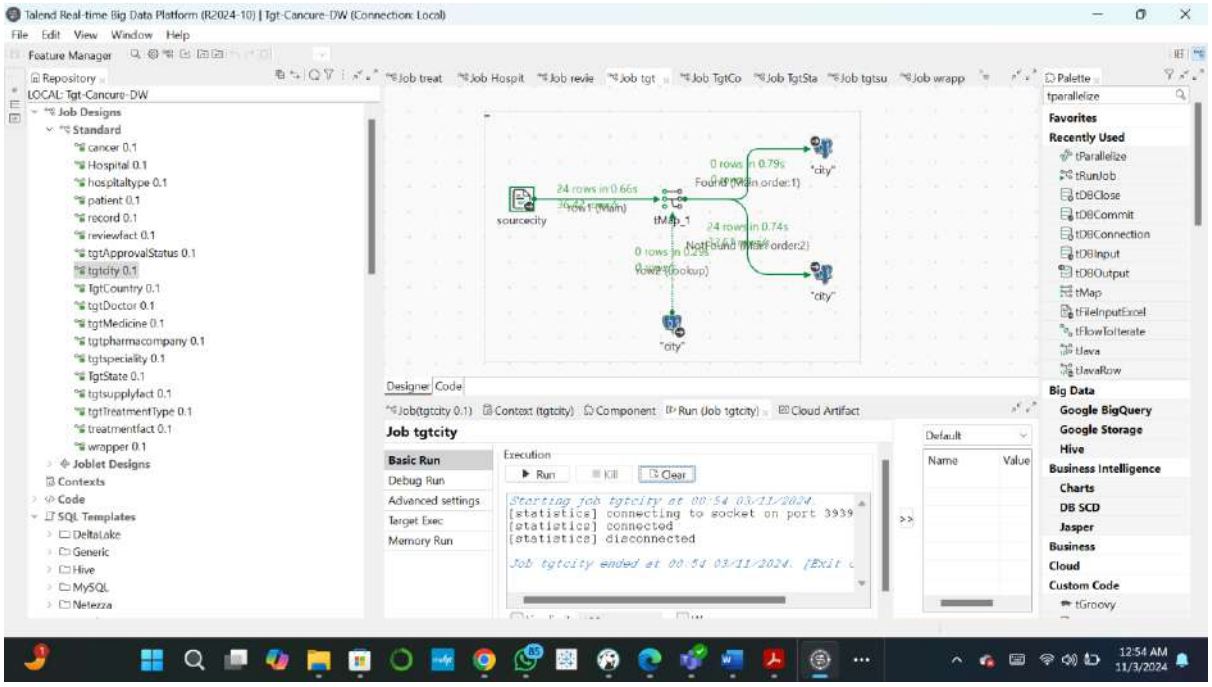

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
doctor_doctorkey_seq
hospital_hospitalkey_seq
hospitaltype_typekey_seq
medicine_medicinekey_seq
medicinesupply_supplykey_seq
patient_patientkey_seq
pharmacompany_pharmacompanykey_seq
record_recordkey_seq
review_reviewkey_seq
speciality_specialitykey_seq
state_statekey_seq
treatmentfact_treatmentkey_seq
treatmenttype_typekey_seq
Data types
Aggregate functions
Event Triggers
Extensions
Storage
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype n ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
delete from country;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
country 1
countrykey
nome
1 United States
2 India
Value
Refresh
Export data
2 row(s) fetched - 0.000s, on 2024-11-03 at 00:53:43
Project - General
Name
DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Tasks: type a part of task name here
Task executions: type a part of error message
EST | en | Writable
Smart Insert
261 : 1 [22]
Sel: 22 | 1
12:53 AM
11/3/2024

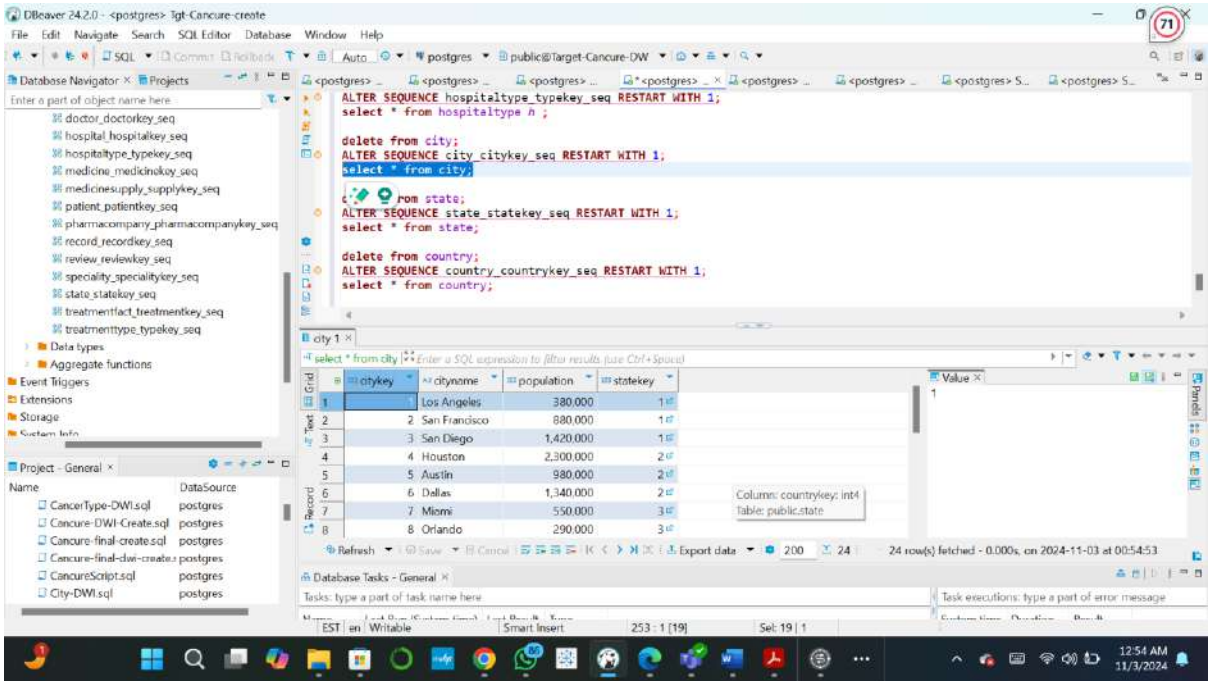
2. State:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
File Edit View Window Help
Repository
LOCAL: Tgt-Cancure-DW
Job Designs
Standard
cancer 0.1
Hospital 0.1
hospitaltype 0.1
patient 0.1
record 0.1
reviewfact 0.1
tgtApprovalStatus 0.1
tgtcity 0.1
TgtCountry 0.1
tgtDoctor 0.1
tgtMedicine 0.1
tgtpharmacompany 0.1
tgtspeciality 0.1
TgtState 0.1
tgtsupplyfact 0.1
tgtTreatmentType 0.1
treatmentfact 0.1
wrapper 0.1
Joblet Designs
Contexts
Code
SQL Templates
DeltaLake
Generic
Hive
MySQL
Netezza
Job TgtState
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Clear
Start Run the Job TgtState at 00:54 03/11/2024.
[statistics] connecting to socket on port 3840
[statistics] connected
[statistics] disconnected
Job TgtState ended at 00:54 03/11/2024. [Exit
Palette
Favorites
Recently Used
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
12:54 AM
11/3/2024

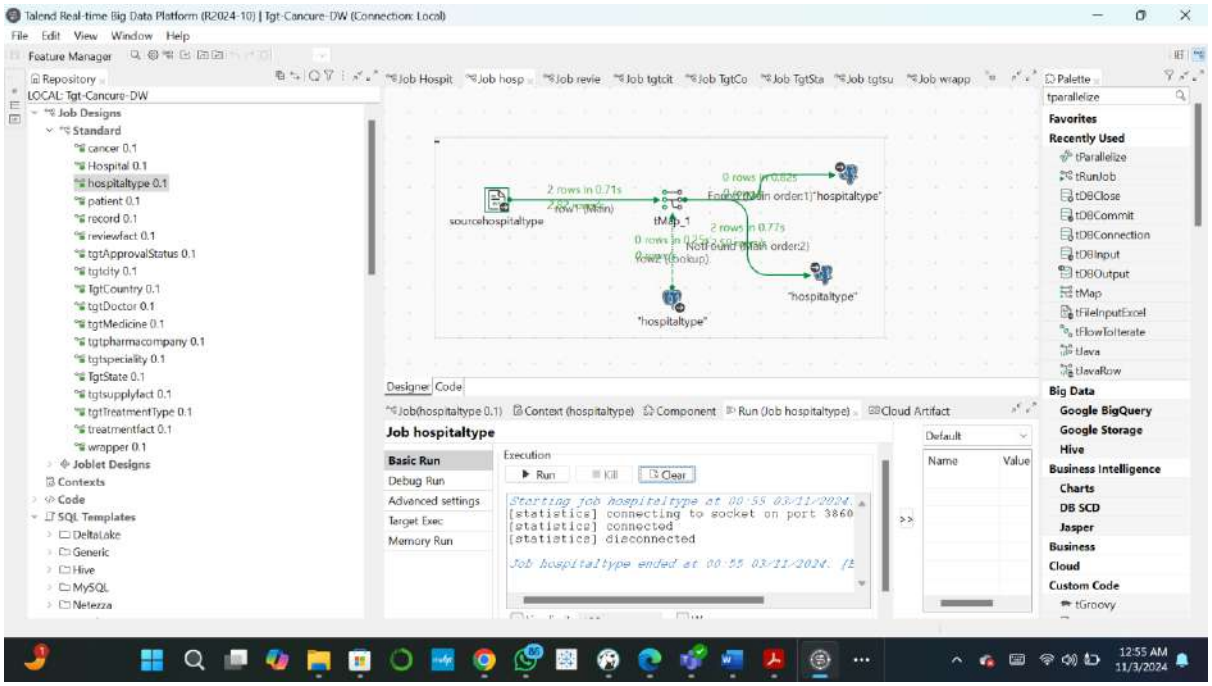

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
statekey statename countrykey
1 California 1
2 Texas 1
3 Florida 1
4 New York 1
5 Illinois 1
6 Pennsylvania 1
7 Ohio 1
8 Georgia 1
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres
8 row(s) fetched - 0.001s, on 2024-11-03 at 00:54:18
12:54 AM
11/3/2024

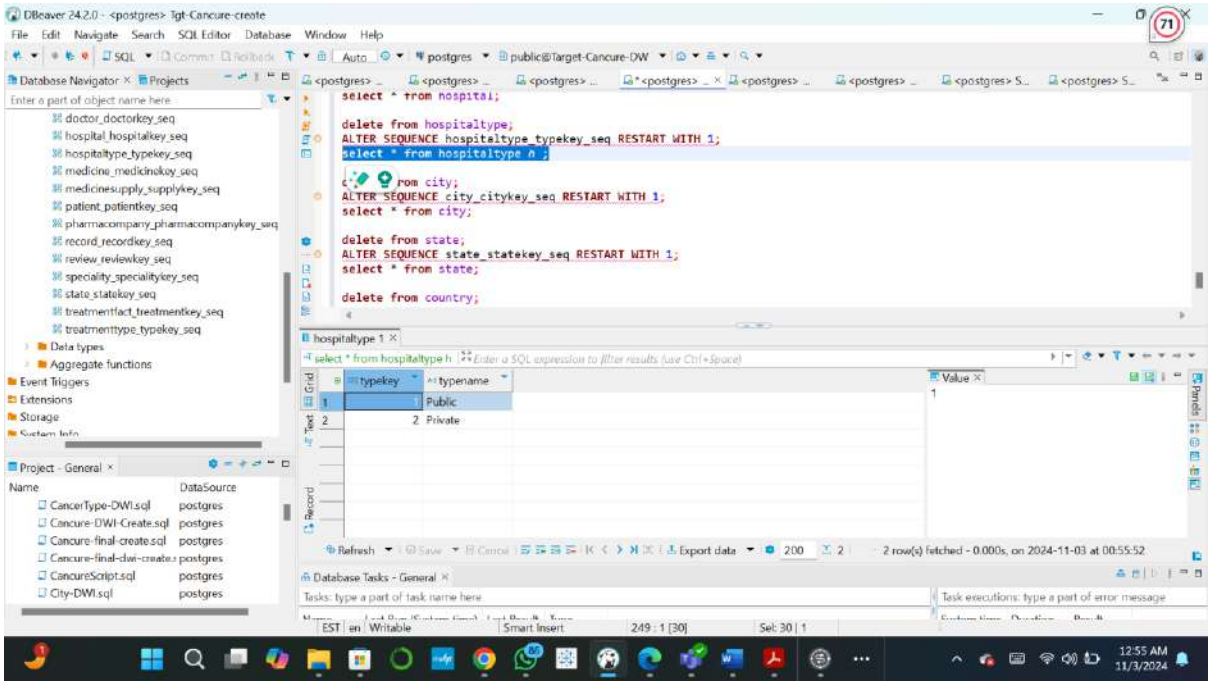
3. City:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
File Edit View Window Help
Feature Manager
Repository
LOCAL: Tgt-Cancure-DW
Job Designs
Standard
cancer 0.1
Hospital 0.1
hospitaltype 0.1
patient 0.1
record 0.1
reviewfact 0.1
tgtApprovalStatus 0.1
tgtcity 0.1
TgtCountry 0.1
tgtDoctor 0.1
tgtMedicine 0.1
tgtpharmacompany 0.1
tgtspeciality 0.1
TgtState 0.1
tgtsupplyfact 0.1
tgtTreatmentType 0.1
treatmentfact 0.1
wrapper 0.1
Joblet Designs
Contexts
Code
SQL Templates
DeltaLake
Generic
Hive
MySQL
Netezza
Job treat
Job Hospit
Job revie
Job tgt
Job TgtCo
Job TgtSta
Job tgtsu
Job wrapp
24 rows in 0.66s
sourcecity
tMap_1
0 rows in 0.79s
Found (Main order:1)
"city"
24 rows in 0.74s
NotFound (Main order:2)
"city"
"city"
Designer
Code
Job(tgtcity 0.1)
Context (tgtcity)
Component
Run (Job tgtcity)
Cloud Artifact
Job tgtcity
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Kill
Clear
Starting job tgtcity at 00:54 03/11/2024.
[statistics] connecting to socket on port 3939
[statistics] connected
[statistics] disconnected
Job tgtcity ended at 00:54 03/11/2024.
Default
Name
Value
Palette
tparallelize
Favorites
Recently Used
tParallelize
tRunJob
tDBClose
tDBCommit
tDBConnection
tDBInput
tDBOutput
tMap
tFileInputExcel
tFlowToIterate
tJava
tJavaRow
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
tGroovy
12:54 AM
11/3/2024

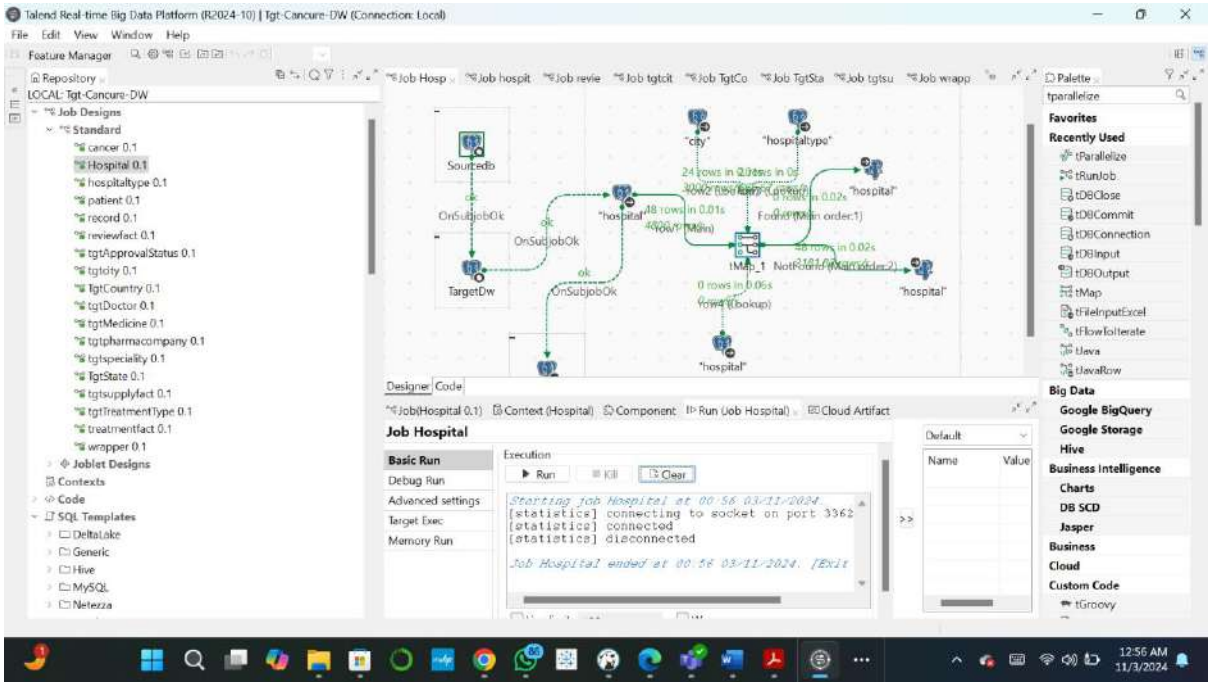

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
doctor_doctorkey_seq
hospital_hospitalkey_seq
hospitaltype_typekey_seq
medicine_medicinekey_seq
medicinesupply_supplykey_seq
patient_patientkey_seq
pharmacompany_pharmacompanykey_seq
record_recordkey_seq
review_reviewkey_seq
speciality_specialitykey_seq
state_statekey_seq
treatmentfact_treatmentkey_seq
treatmenttype_typekey_seq
Data types
Aggregate functions
Event Triggers
Extensions
Storage
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype h ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
delete from country;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
city 1
select * from city
citykey
cityname
population
statekey
1 Los Angeles 380,000 1
2 San Francisco 880,000 1
3 San Diego 1,420,000 1
4 Houston 2,300,000 2
5 Austin 980,000 2
6 Dallas 1,340,000 2
7 Miami 550,000 3
8 Orlando 290,000 3
Value
1
Column: countrykey: int4
Table: public.state
24 row(s) fetched - 0.000s, on 2024-11-03 at 00:54:53
Project - General
Name
DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
Cancure-final-dwi-create.s postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Smart Insert
253 : 1 [19]
Sel: 19 | 1
12:54 AM
11/3/2024

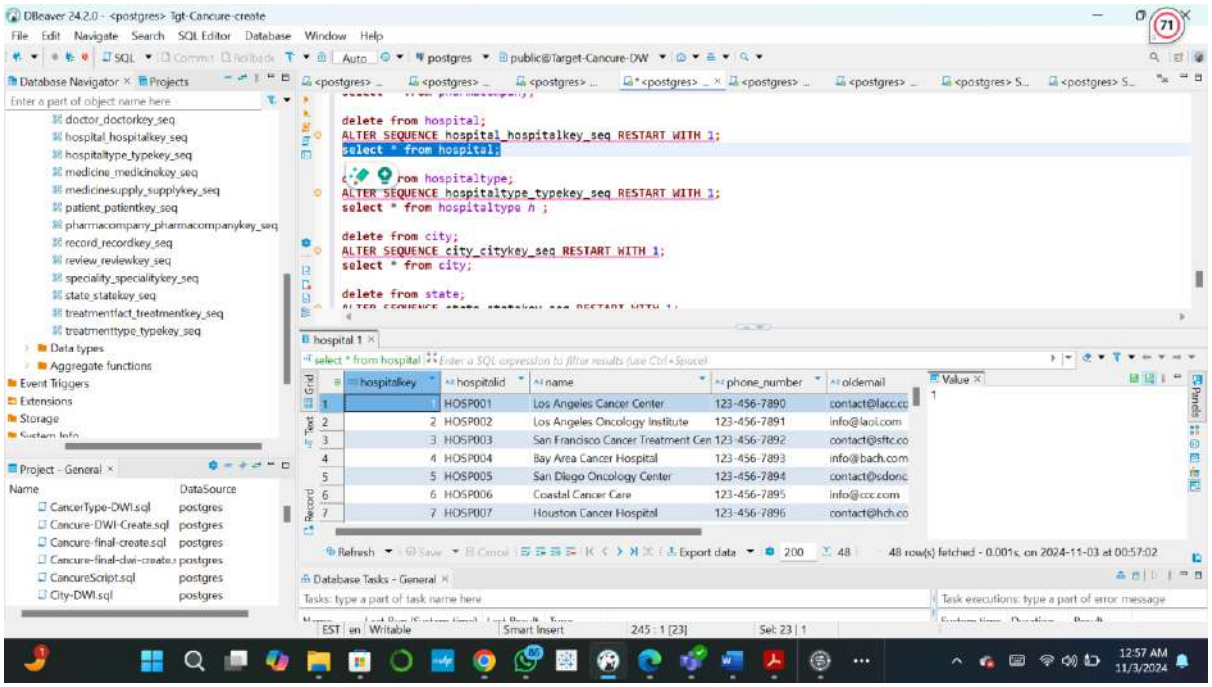
4. Hospital Type:



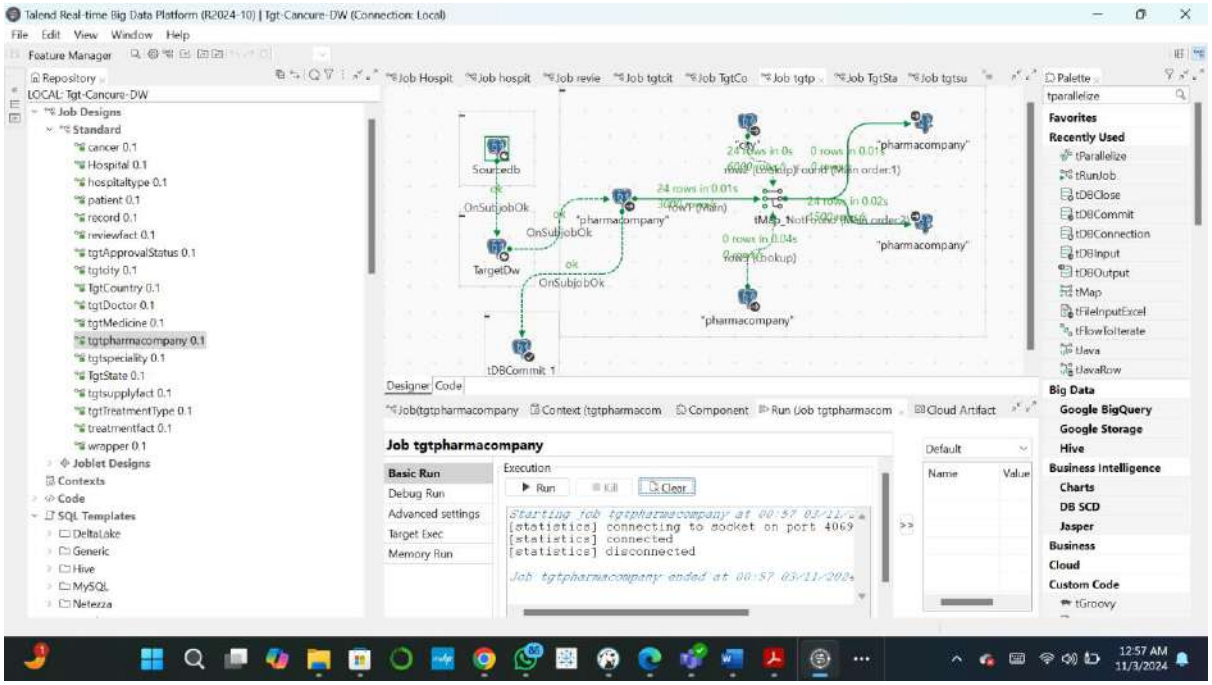


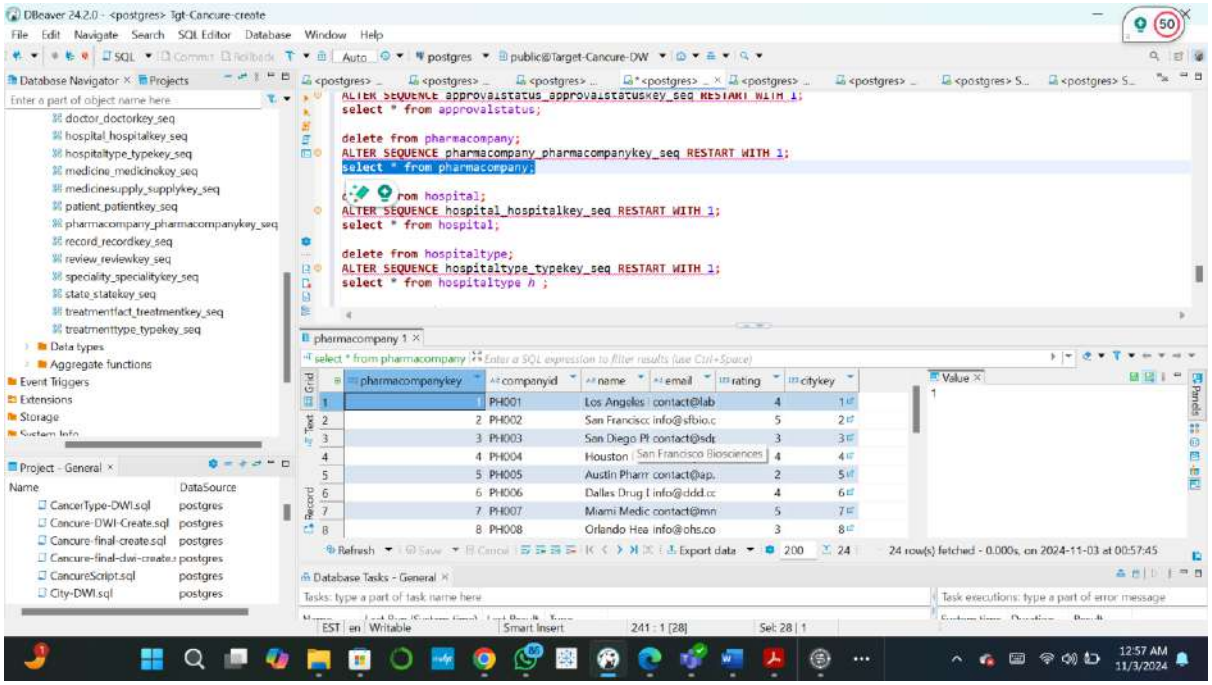
5. Hospital:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
File Edit View Window Help
Repository
LOCAL: Tgt-Cancure-DW
Job Designs
Standard
cancer 0.1
Hospital 0.1
hospitaltype 0.1
patient 0.1
record 0.1
reviewfact 0.1
tgtApprovalStatus 0.1
tgtcity 0.1
TgtCountry 0.1
tgtDoctor 0.1
tgtMedicine 0.1
tgtpharmacompany 0.1
tgtspeciality 0.1
TgtState 0.1
tgtsupplyfact 0.1
tgtTreatmentType 0.1
treatmentfact 0.1
wrapper 0.1
Joblet Designs
Contexts
Code
SQL Templates
DeltaLake
Generic
Hive
MySQL
Netezza
Job Hosp
Job hospit
Job revie
Job tgtcit
Job TgtCo
Job TgtSta
Job tgtsu
Job wrapp
SourceDb
TargetDw
OnSubjobOk
"city"
"hospitaltype"
"hospital"
24 rows in 0.03s
48 rows in 0.01s
0 rows in 0.06s
Designer
Code
Job(Hospital 0.1)
Context (Hospital)
Component
Run (Job Hospital)
Cloud Artifact
Job Hospital
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Kill
Clear
Starting job Hospital at 00:56 03/11/2024
[statistics] connecting to socket on port 3362
[statistics] connected
[statistics] disconnected
Job Hospital ended at 00:56 03/11/2024. [Exit
Default
Name
Value
Palette
tparallelize
Favorites
Recently Used
tParallelize
tRunJob
tDBClose
tDBCommit
tDBConnection
tDBInput
tDBOutput
tMap
tFileInputExcel
tFlowToIterate
tJava
tJavaRow
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
tGroovy
12:56 AM
11/3/2024

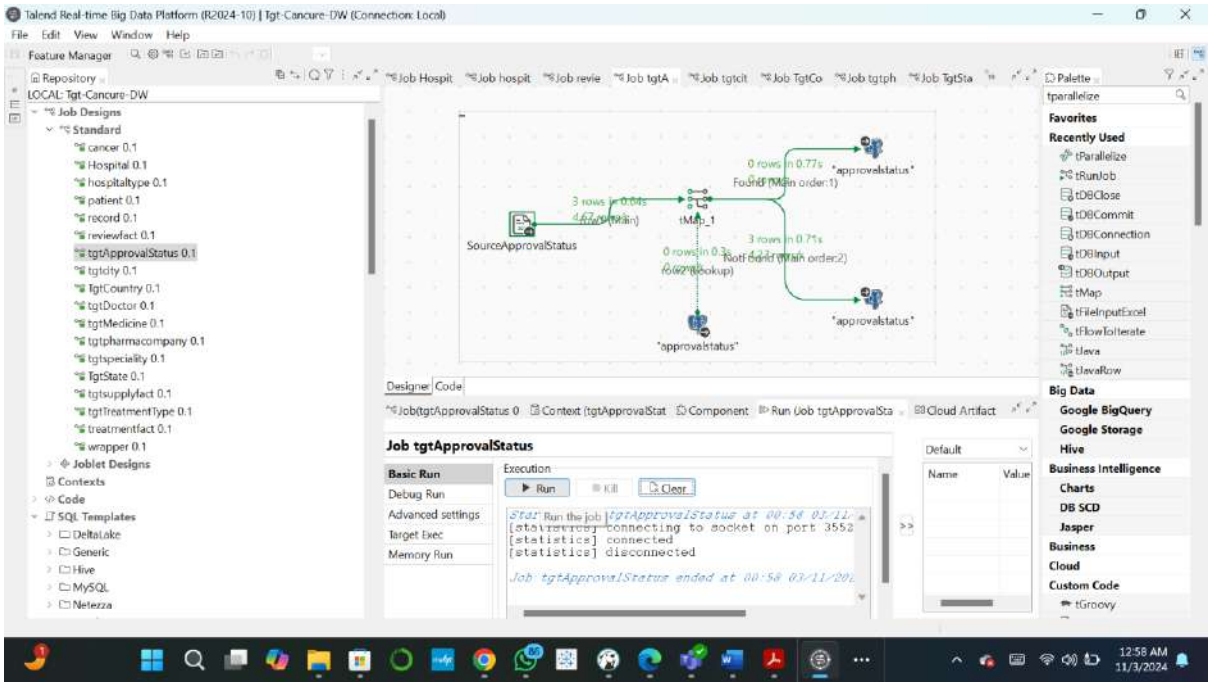

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
doctor_doctorkey_seq
hospital_hospitalkey_seq
hospitaltype_typekey_seq
medicine_medicinekey_seq
medicinesupply_supplykey_seq
patient_patientkey_seq
pharmacompany_pharmacompanykey_seq
record_recordkey_seq
review_reviewkey_seq
speciality_specialitykey_seq
state_statekey_seq
treatmentfact_treatmentkey_seq
treatmenttype_typekey_seq
Data types
Aggregate functions
Event Triggers
Extensions
Storage
Project - General
CancerType-DWI.sql
Cancure-DWI-Create.sql
Cancure-final-create.sql
CancureScript.sql
City-DWI.sql
delete from hospital;
ALTER SEQUENCE hospital_hospitalkey_seq RESTART WITH 1;
select * from hospital;
delete from hospitaltype;
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
hospital 1
select * from hospital
hospitalkey
hospitalid
name
phone_number
oldemail
1 HOSP001 Los Angeles Cancer Center 123-456-7890 contact@lacc.co
2 HOSP002 Los Angeles Oncology Institute 123-456-7891 info@laoi.com
3 HOSP003 San Francisco Cancer Treatment Cen 123-456-7892 contact@sftc.co
4 HOSP004 Bay Area Cancer Hospital 123-456-7893 info@bach.com
5 HOSP005 San Diego Oncology Center 123-456-7894 contact@sdonc
6 HOSP006 Coastal Cancer Care 123-456-7895 info@ccc.com
7 HOSP007 Houston Cancer Hospital 123-456-7896 contact@hch.co
Value
1
Refresh
Export data
48 row(s) fetched - 0.001s, on 2024-11-03 at 00:57:02
Database Tasks - General
Tasks: type a part of task name here
Task executions: type a part of error message
12:57 AM
11/3/2024

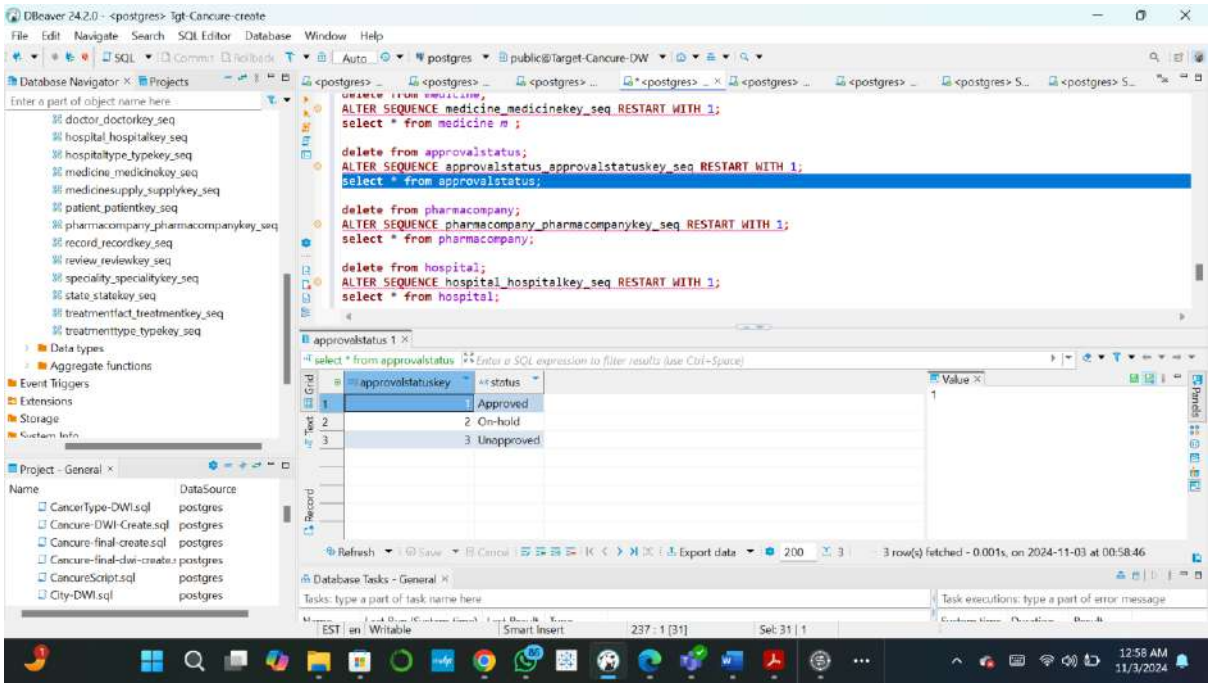
6. Pharma company:



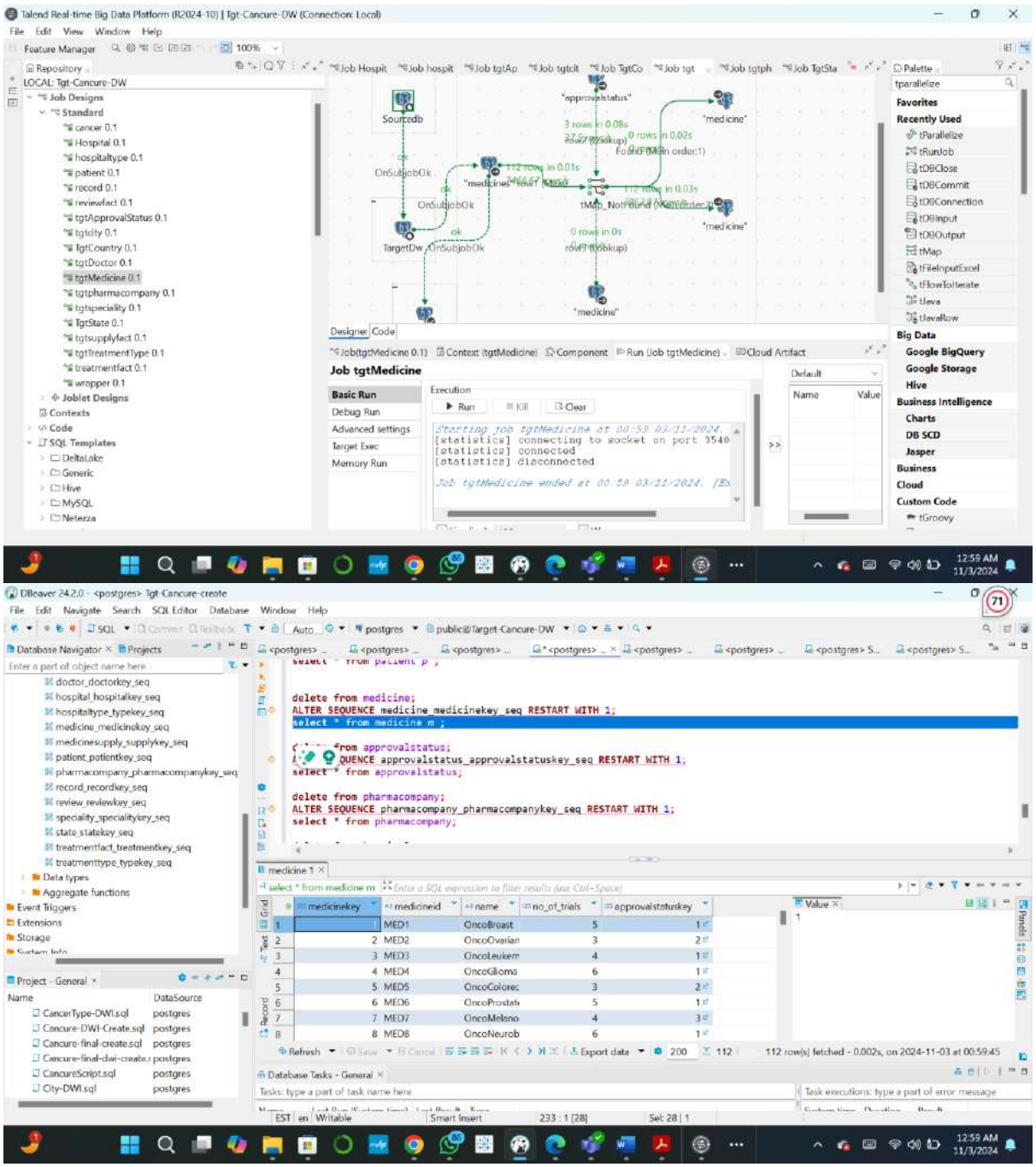


7.ApprovalStatus:

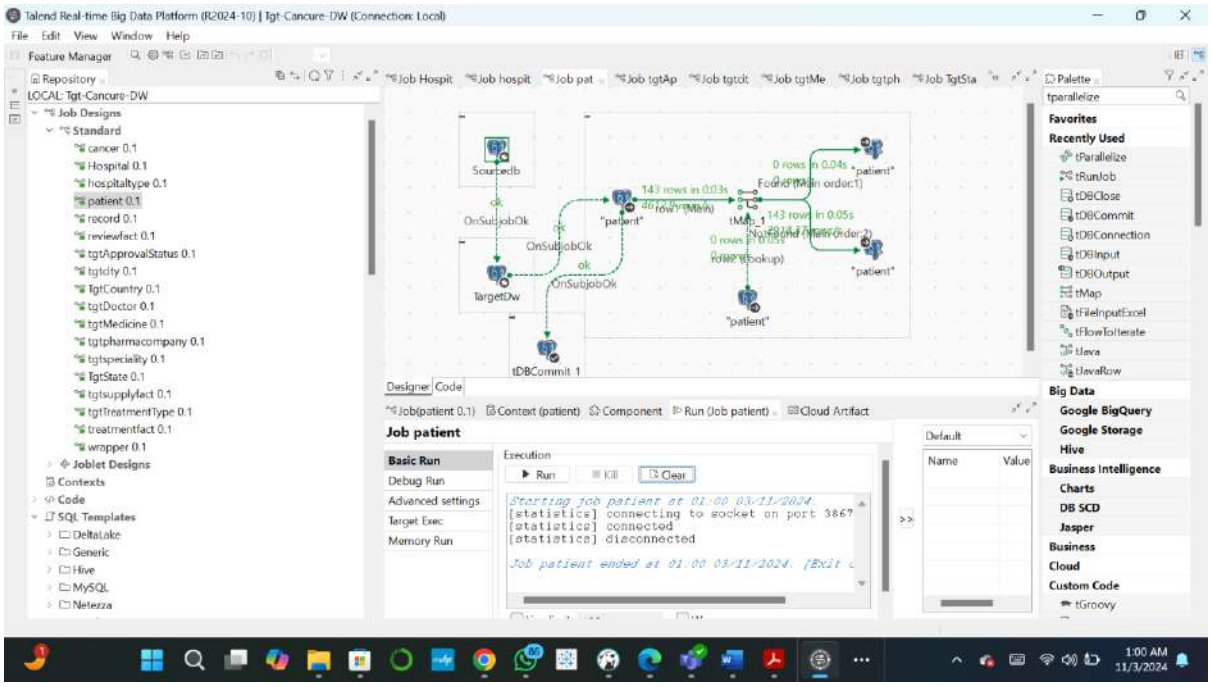


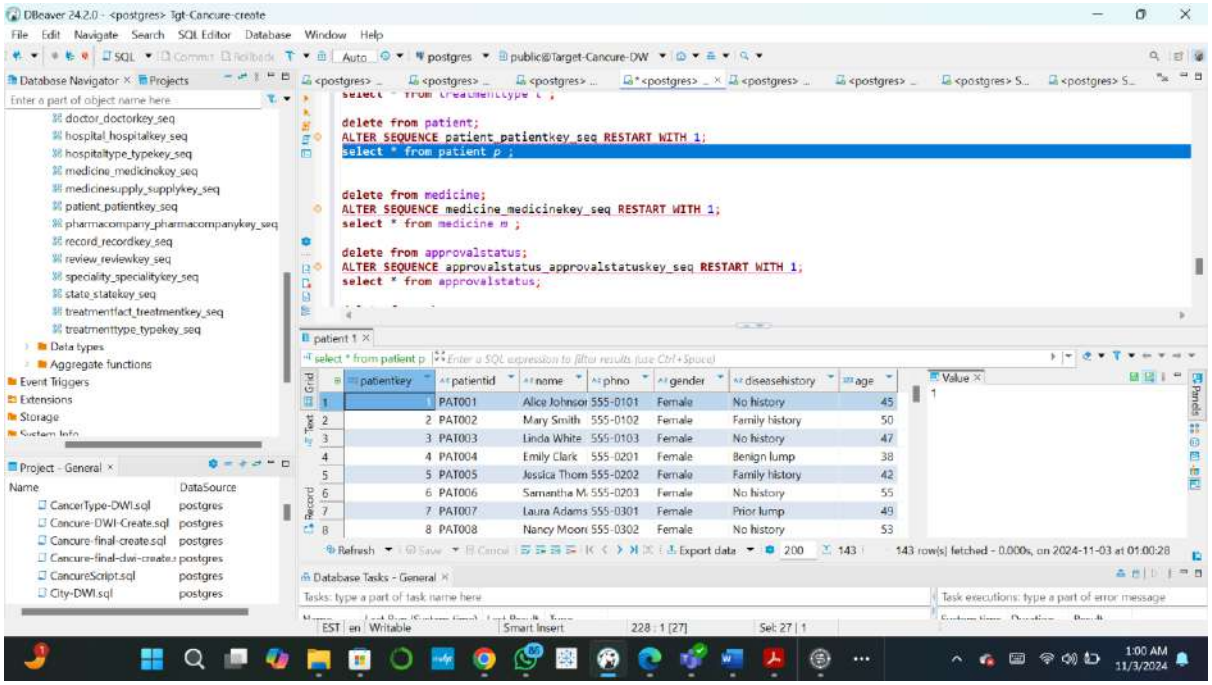


8. Medicine:

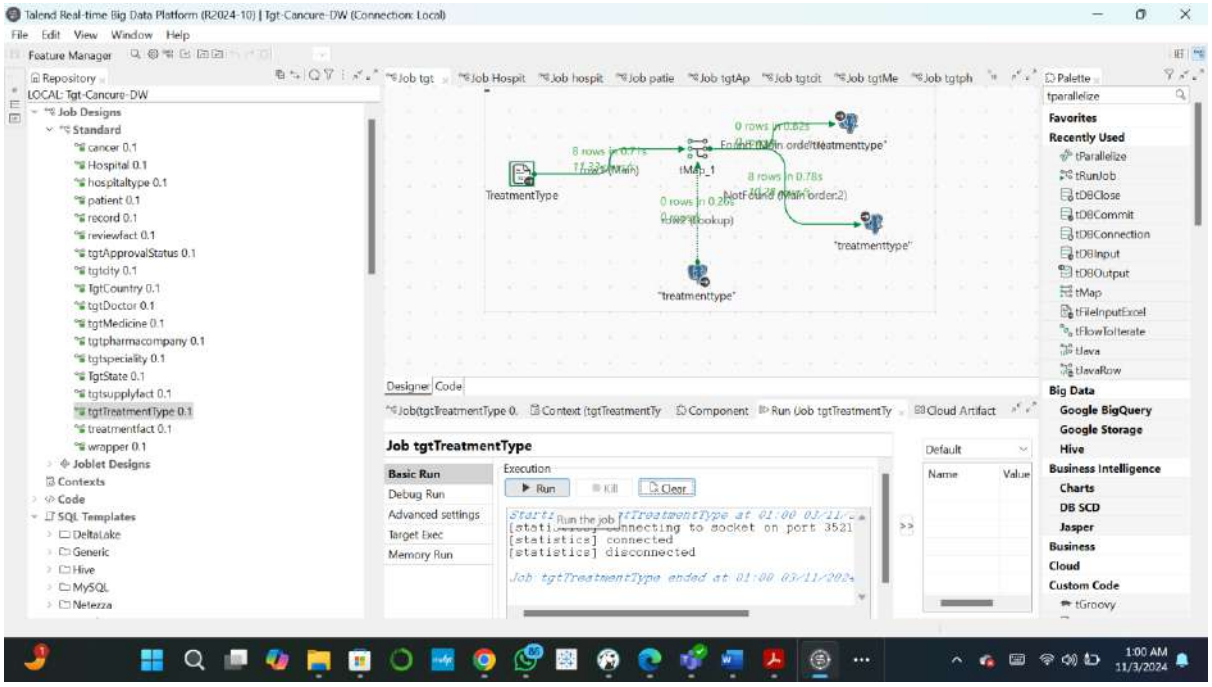

Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
Job tgtMedicine
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from medicine;
ALTER SEQUENCE medicine_medicinekey_seq RESTART WITH 1;
select * from medicine m ;
delete from pharmacompany;
ALTER SEQUENCE pharmacompany_pharmacompanykey_seq RESTART WITH 1;
select * from pharmacompany;

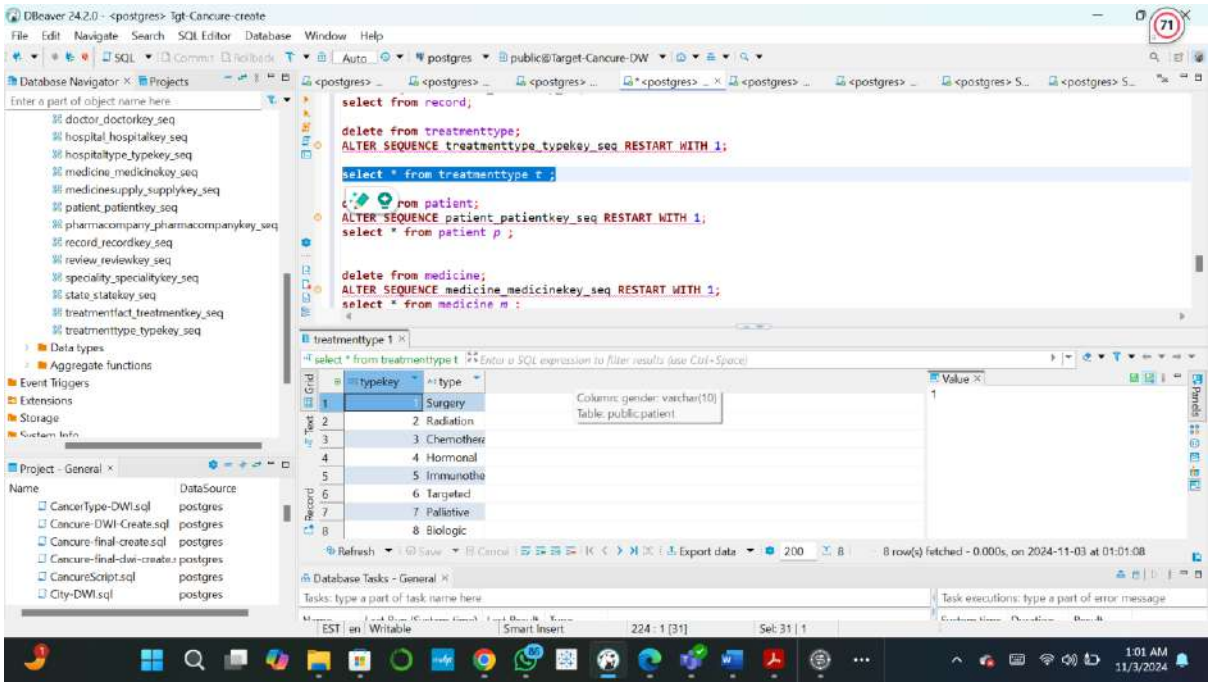
9. Patient:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
Job patient
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Kill
Clear
Starting job patient at 01:00 03/11/2024.
[statistics] connecting to socket on port 3867
[statistics] connected
[statistics] disconnected
Job patient ended at 01:00 03/11/2024.
Palette
Favorites
Recently Used
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
1:00 AM
11/3/2024

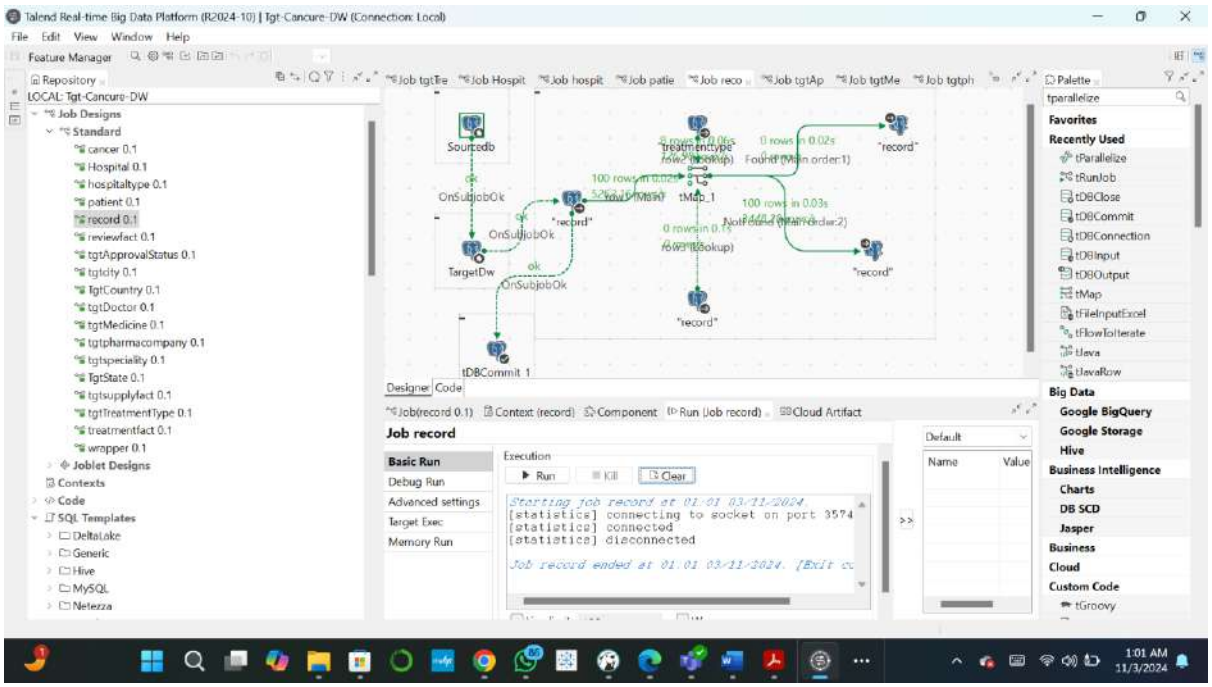

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from patient;
ALTER SEQUENCE patient_patientkey_seq RESTART WITH 1;
select * from patient p ;
delete from medicine;
ALTER SEQUENCE medicine_medicinekey_seq RESTART WITH 1;
select * from medicine m ;
delete from approvalstatus;
ALTER SEQUENCE approvalstatus_approvalstatuskey_seq RESTART WITH 1;
select * from approvalstatus;
patient 1
patientkey | patientid | name | phno | gender | diseasehistory | age
1 | PAT001 | Alice Johnson | 555-0101 | Female | No history | 45
2 | PAT002 | Mary Smith | 555-0102 | Female | Family history | 50
3 | PAT003 | Linda White | 555-0103 | Female | No history | 47
4 | PAT004 | Emily Clark | 555-0201 | Female | Benign lump | 38
5 | PAT005 | Jessica Thom | 555-0202 | Female | Family history | 42
6 | PAT006 | Samantha M | 555-0203 | Female | No history | 55
7 | PAT007 | Laura Adams | 555-0301 | Female | Prior lump | 49
8 | PAT008 | Nancy Moore | 555-0302 | Female | No history | 53
143 row(s) fetched - 0.000s, on 2024-11-03 at 01:00:28
1:00 AM
11/3/2024

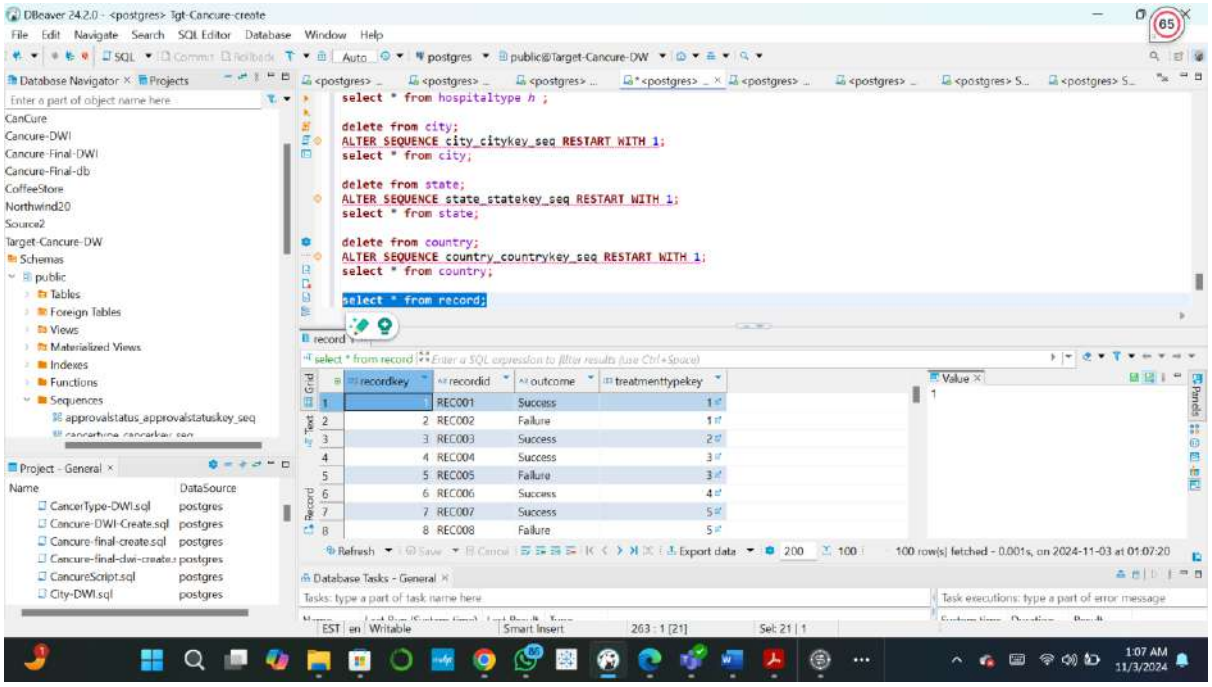
10.TreatmentType:



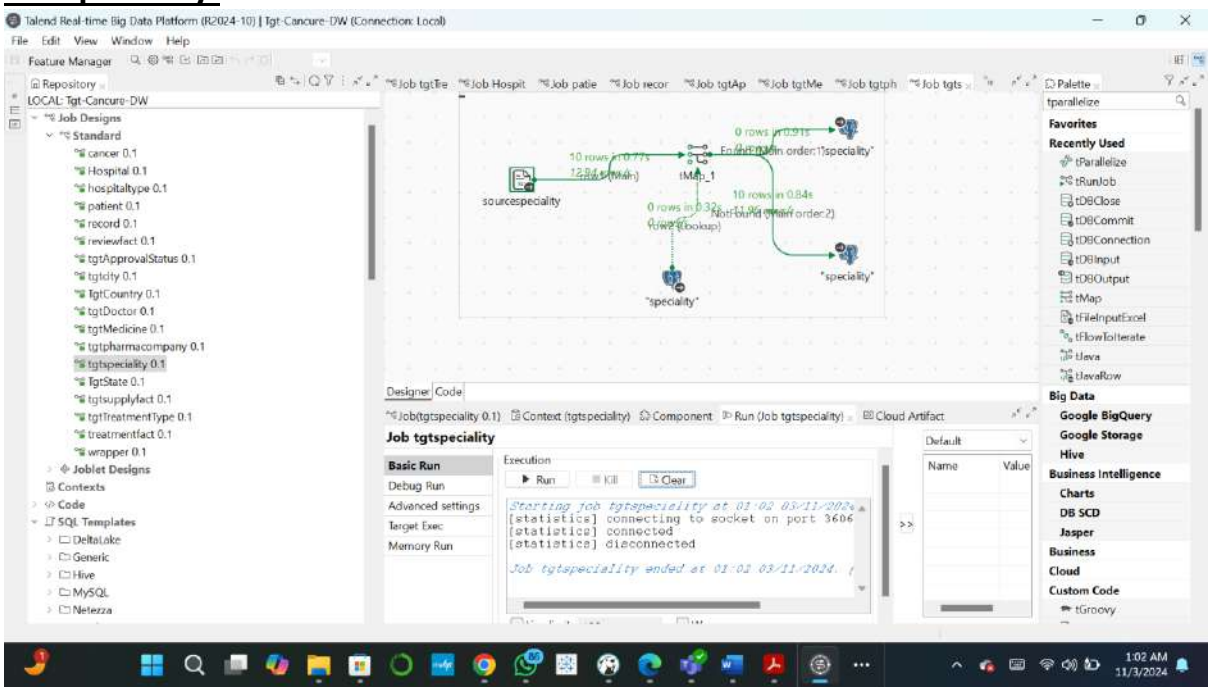


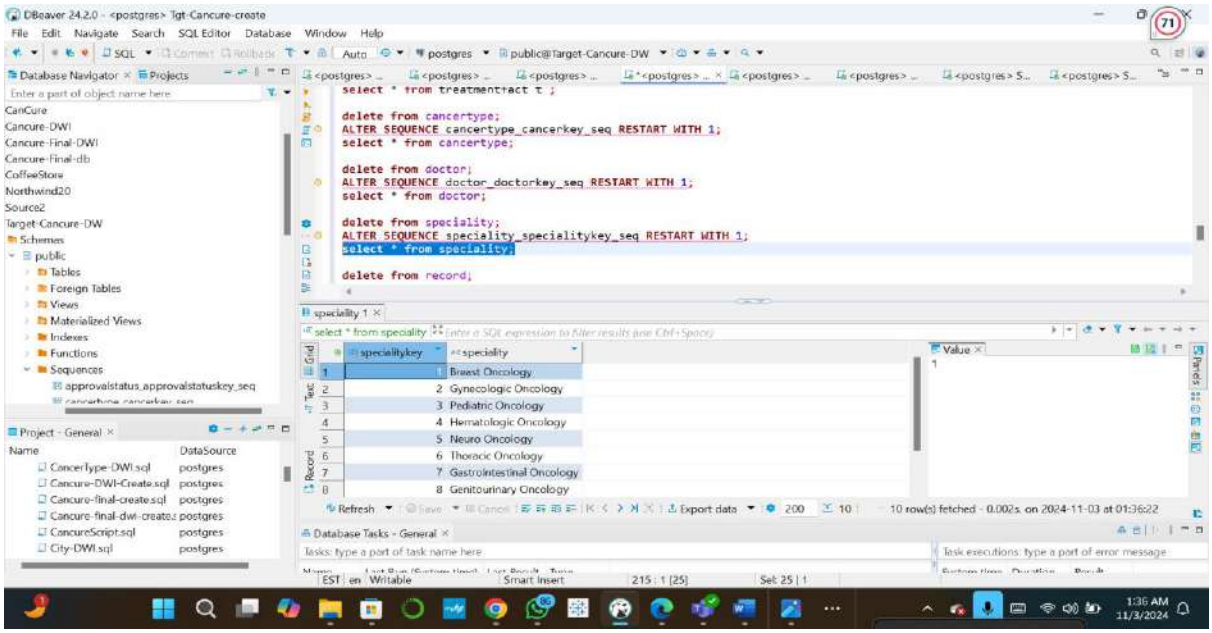
11. Record:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
File Edit View Window Help
Feature Manager
Repository
LOCAL: Tgt-Cancure-DW
Job Designs
Standard
cancer 0.1
Hospital 0.1
hospitaltype 0.1
patient 0.1
record 0.1
reviewfact 0.1
tgtApprovalStatus 0.1
tgtcity 0.1
TgtCountry 0.1
tgtDoctor 0.1
tgtMedicine 0.1
tgtpharmacompany 0.1
tgtspeciality 0.1
TgtState 0.1
tgtsupplyfact 0.1
tgtTreatmentType 0.1
treatmentfact 0.1
wrapper 0.1
Joblet Designs
Contexts
Code
SQL Templates
DeltaLake
Generic
Hive
MySQL
Netezza
Job tgtTre
Job Hospit
Job hospit
Job patie
Job reco
Job tgtAp
Job tgtMe
Job tgtph
SourceDb
TargetDw
tDBCommit_1
OnSubjobOk
ok
"record"
tMap_1
Designer
Code
Job(record 0.1)
Context (record)
Component
Run (Job record)
Cloud Artifact
Job record
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Kill
Clear
Starting job record at 01:01 03/11/2024.
[statistics] connecting to socket on port 3574
[statistics] connected
[statistics] disconnected
Job record ended at 01:01 03/11/2024.
Default
Name
Value
Palette
tparallelize
Favorites
Recently Used
tParallelize
tRunJob
tDBClose
tDBCommit
tDBConnection
tDBInput
tDBOutput
tMap
tFileInputExcel
tFlowToIterate
tJava
tJavaRow
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
tGroovy
1:01 AM
11/3/2024

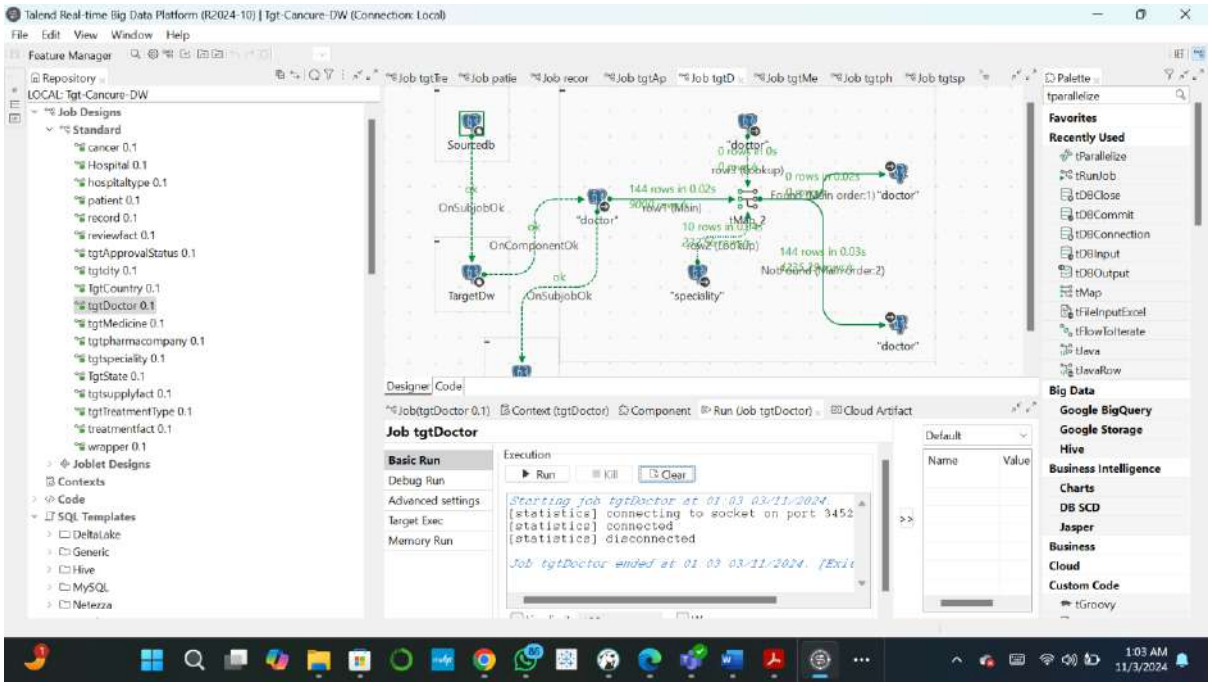


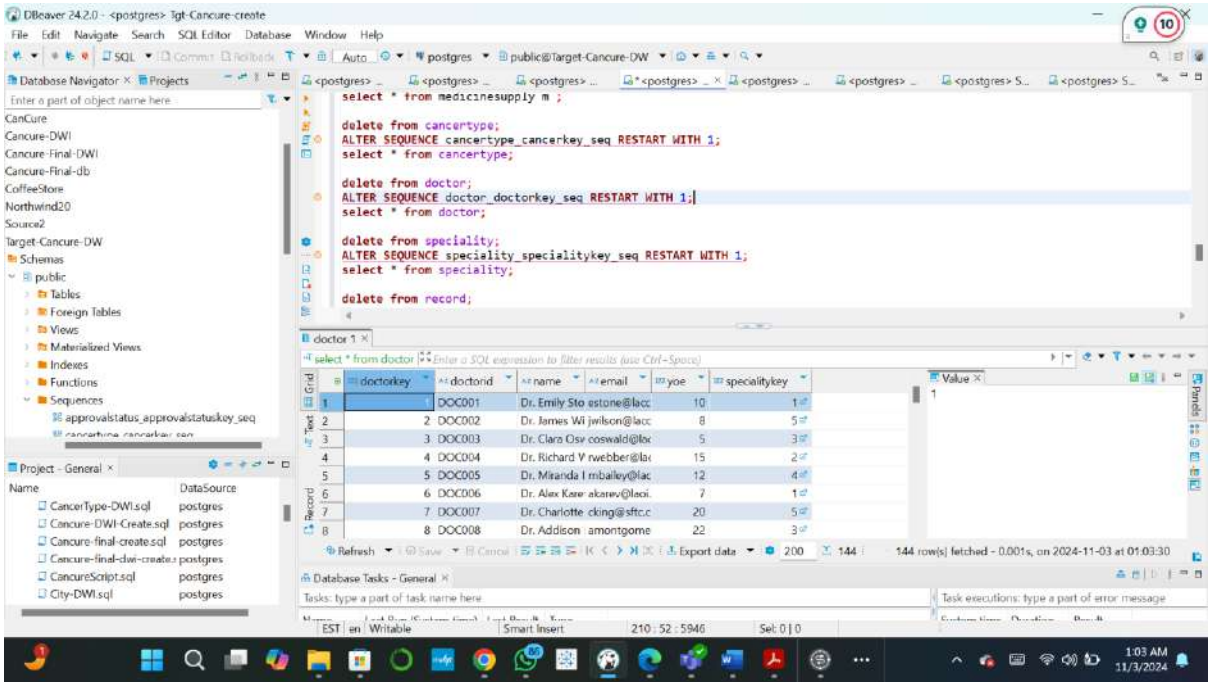
12. Speciality:



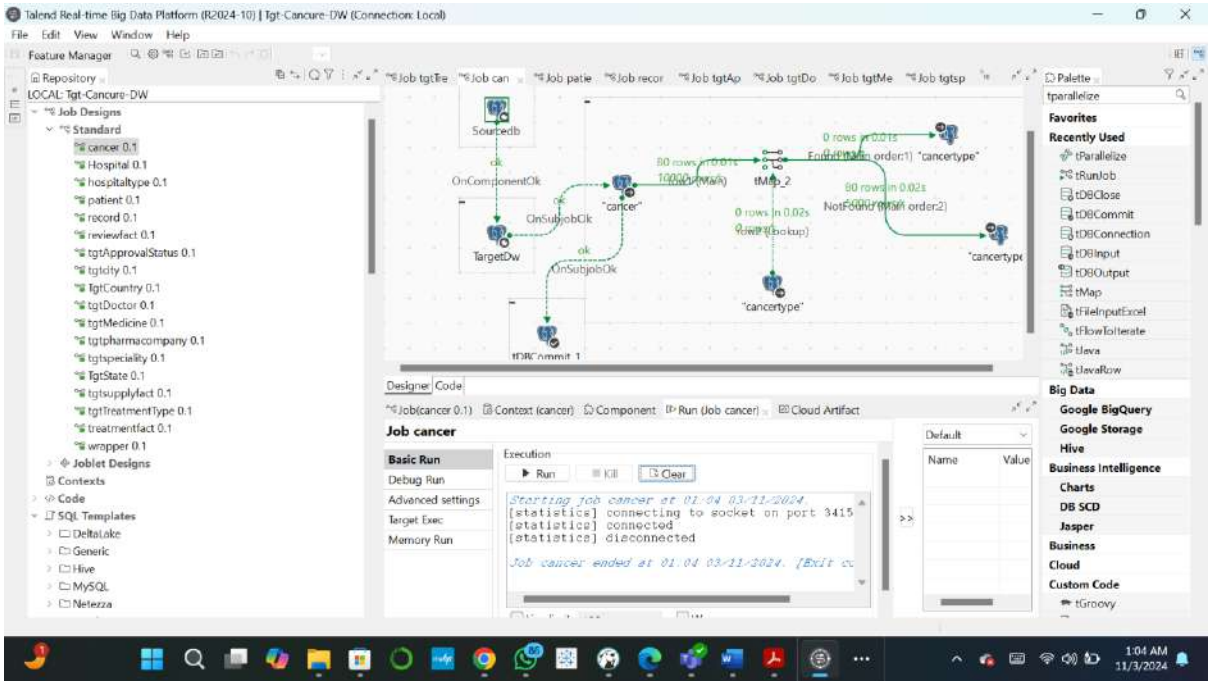

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
select * from treatmentfact t ;
delete from cancertype;
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
select * from cancertype;
delete from doctor;
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
delete from speciality;
ALTER SEQUENCE speciality_specialitykey_seq RESTART WITH 1;
select * from speciality;
delete from record;
speciality 1
specialitykey
speciality
1 Breast Oncology
2 Gynecologic Oncology
3 Pediatric Oncology
4 Hematologic Oncology
5 Neuro Oncology
6 Thoracic Oncology
7 Gastrointestinal Oncology
8 Genitourinary Oncology
10 row(s) fetched - 0.002s, on 2024-11-03 at 01:36:22
1:36 AM
11/3/2024

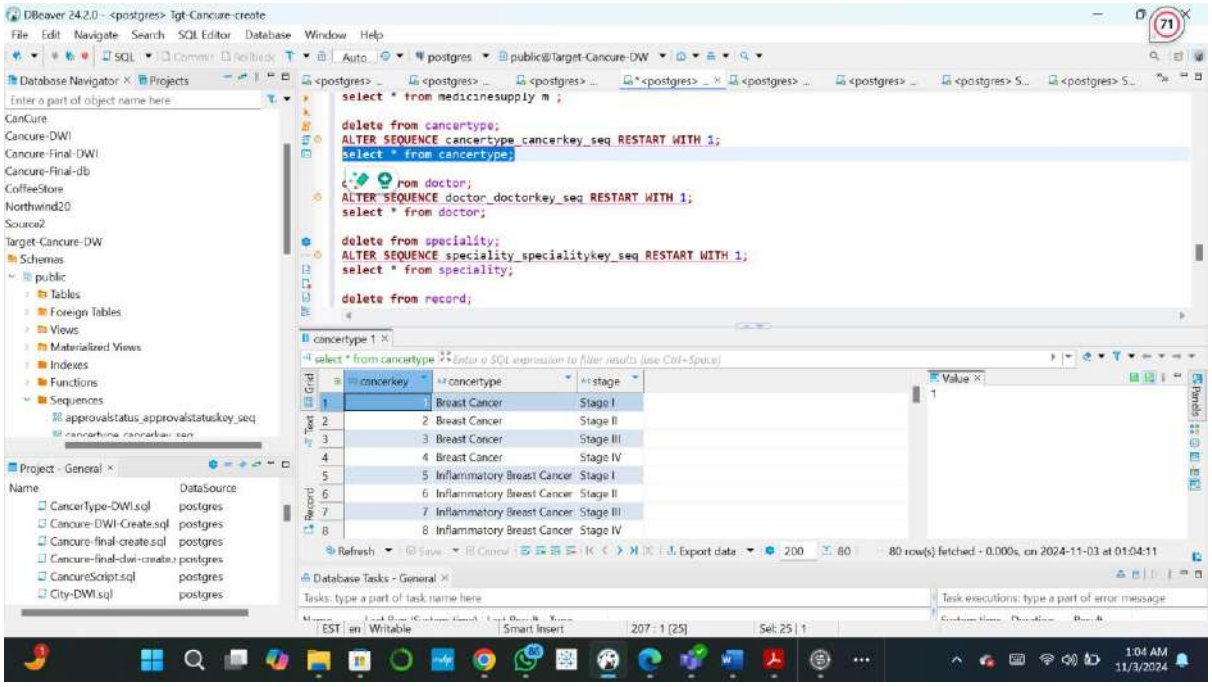
13. Doctor:


Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
LOCAL: Tgt-Cancure-DW
Job Designs
Standard
tgtDoctor 0.1
Sourcedb
TargetDw
"doctor"
"speciality"
144 rows in 0.02s
144 rows in 0.03s
10 rows in 0.01s
OnSubjobOk
OnComponentOk
Job tgtDoctor
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Starting job tgtDoctor at 01:03 03/11/2024.
[statistics] connecting to socket on port 3452
[statistics] connected
[statistics] disconnected
Job tgtDoctor ended at 01:03 03/11/2024.
Palette
tparallelize
Favorites
Recently Used
Big Data
Google BigQuery
Google Storage
Hive
Business Intelligence
Charts
DB SCD
Jasper
Business
Cloud
Custom Code
1:03 AM
11/3/2024


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
select * from medicinesupply m ;
delete from cancertype;
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
select * from cancertype;
delete from doctor;
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
delete from speciality;
ALTER SEQUENCE speciality_specialitykey_seq RESTART WITH 1;
select * from speciality;
delete from record;

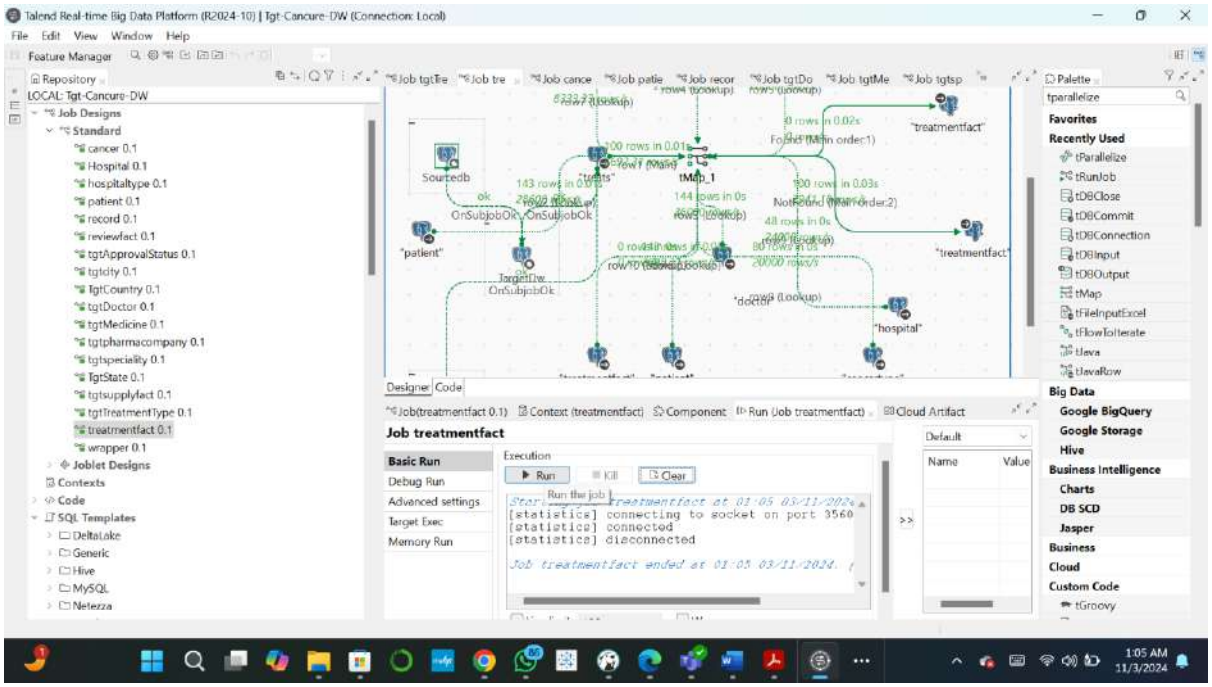
14.Cancer:

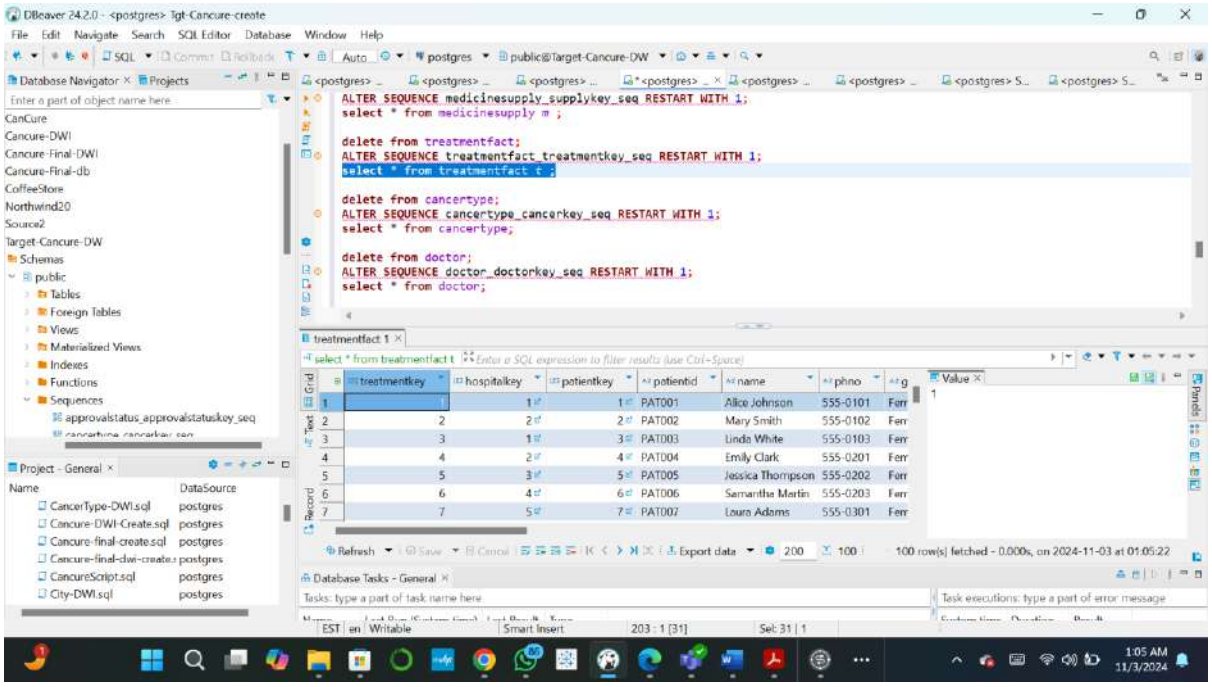

Talend Real-time Big Data Platform (R2024-10) | Tgt-Cancure-DW (Connection: Local)
Job cancer
Starting job cancer at 01:04 03/11/2024.
[statistics] connecting to socket on port 3415
[statistics] connected
[statistics] disconnected
Job cancer ended at 01:04 03/11/2024.



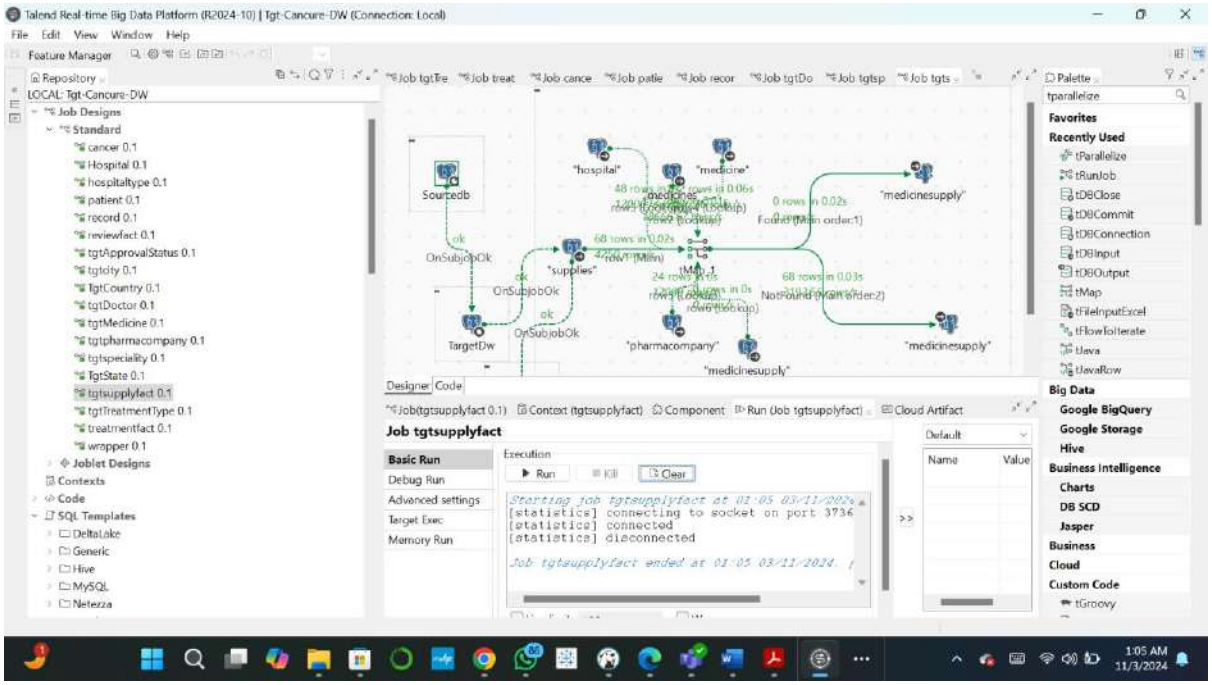
Fact Tables:

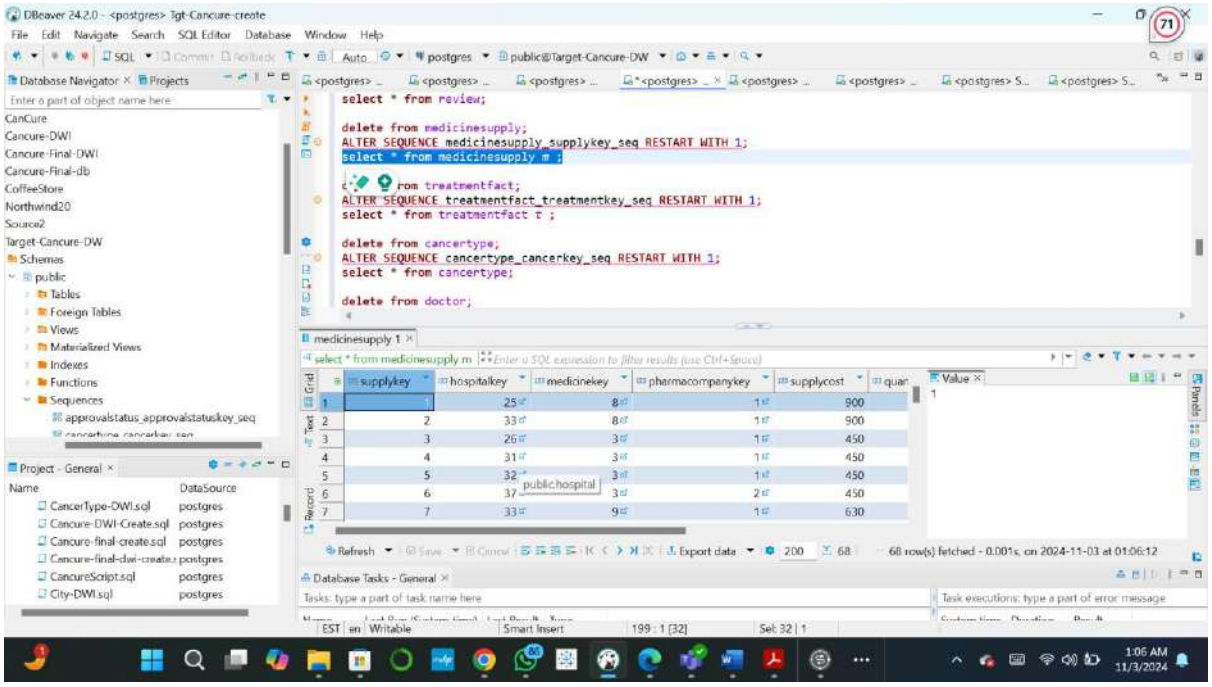
15.TreatmentFact:



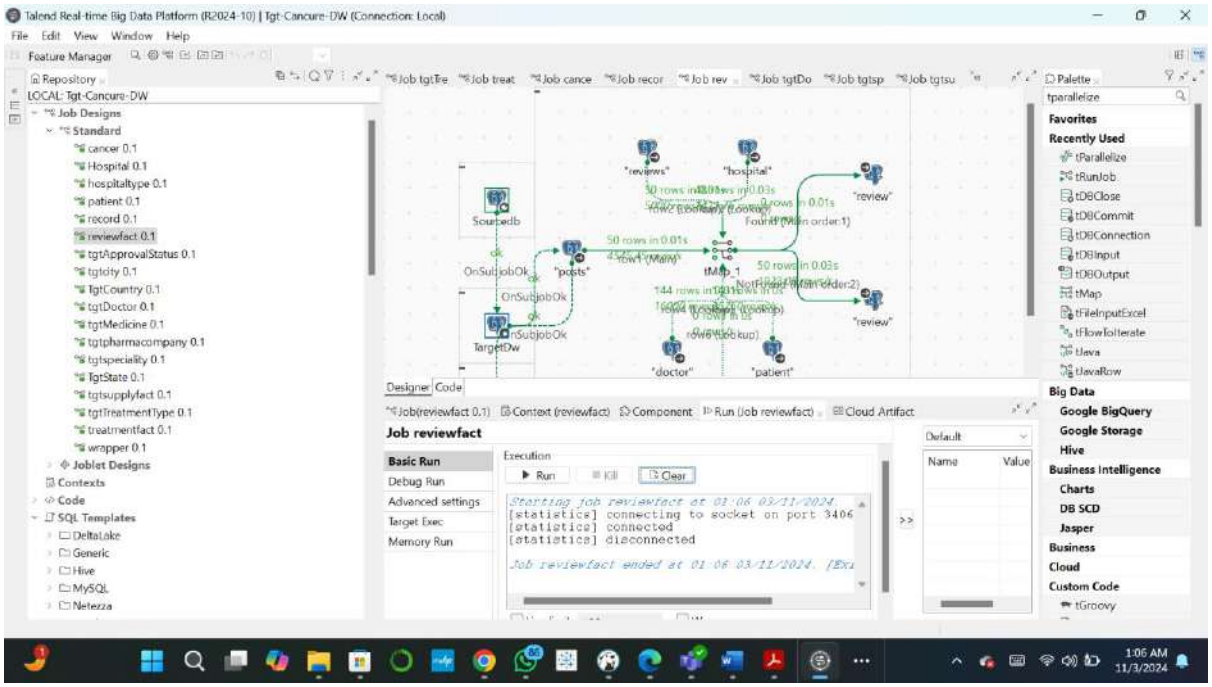


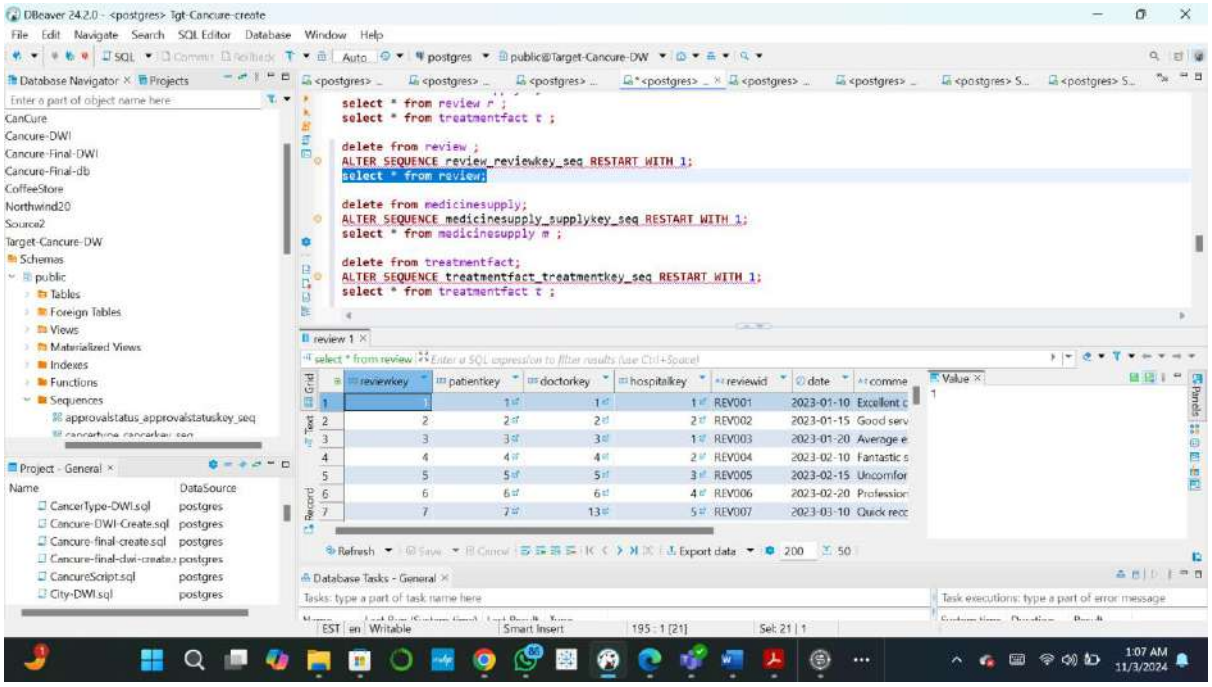
16.SupplyFact:





17.ReviewFact:



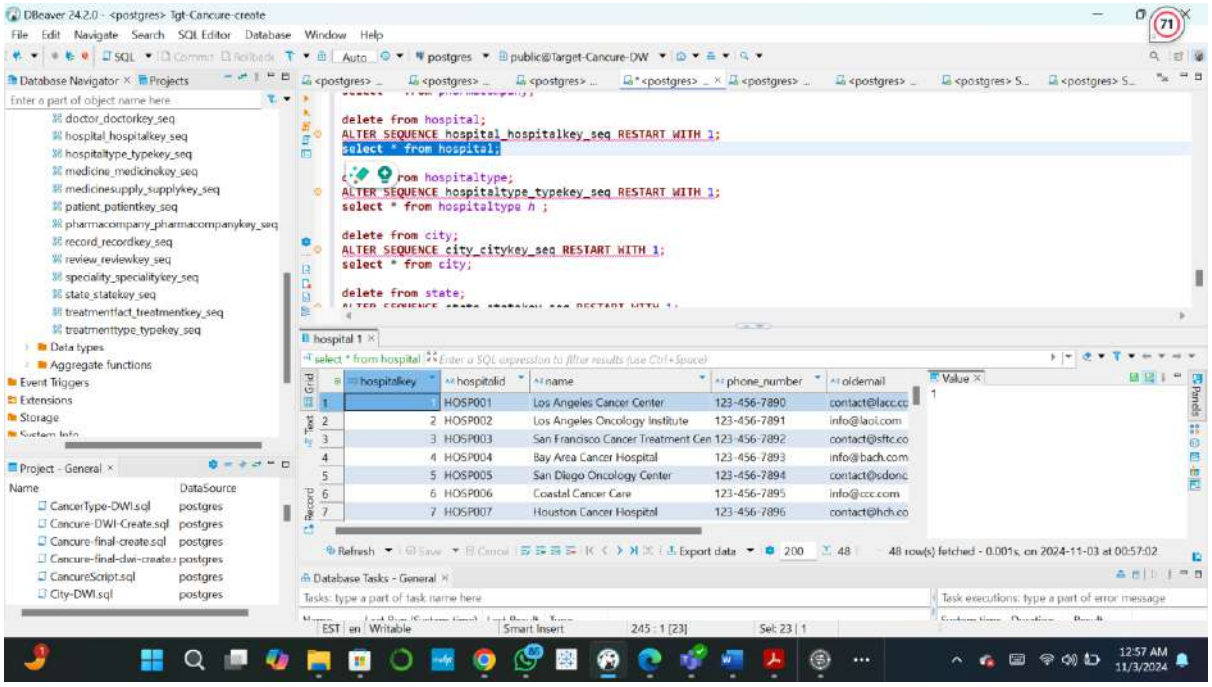


All the tables are successfully loaded into the Target DatawarehouseDB from the Relational Database through ETL.

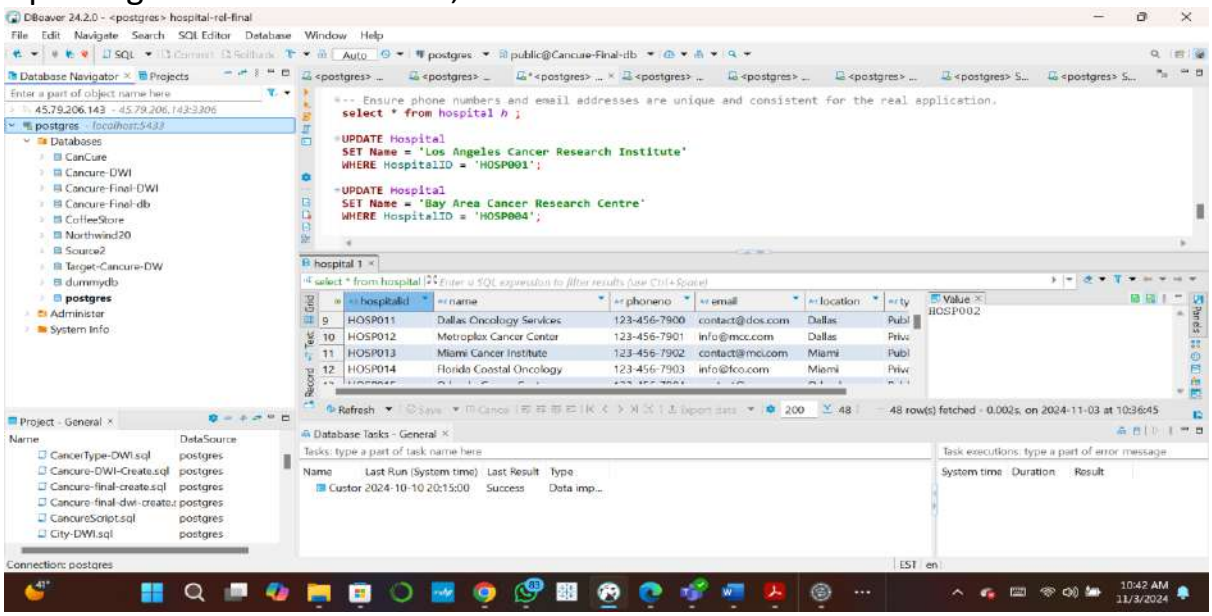
Update Flow:

Hospital:

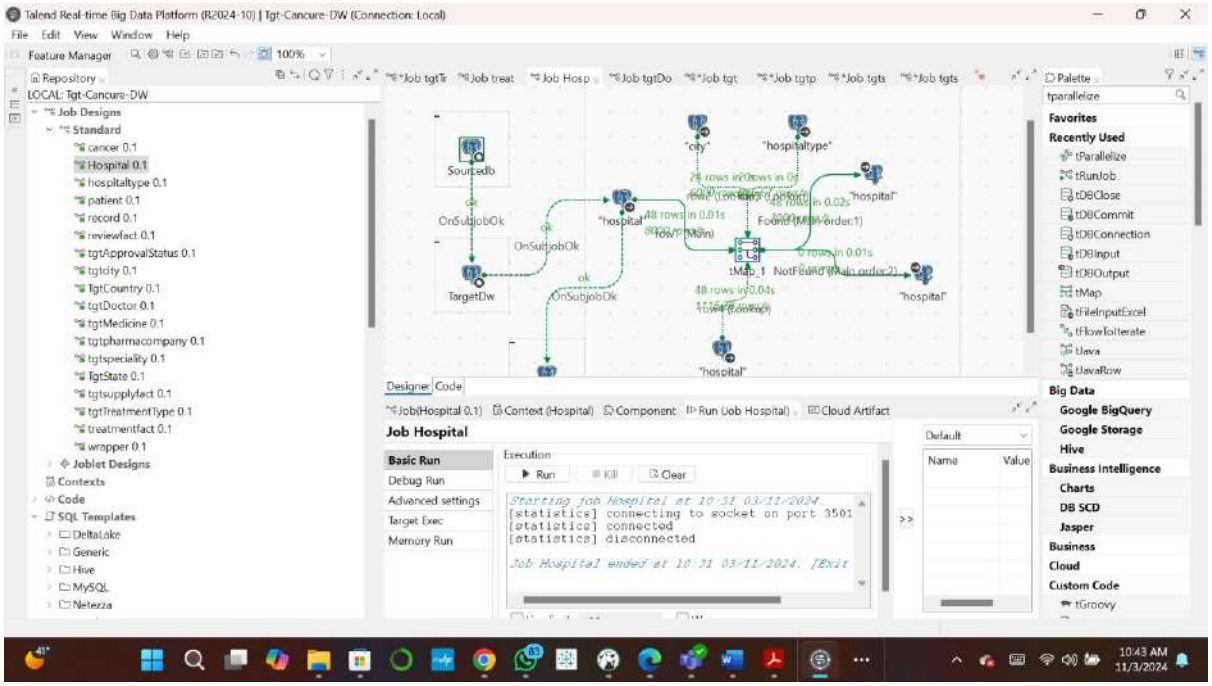
Before Update: Screenshot of the Hospital table in the data warehouse.



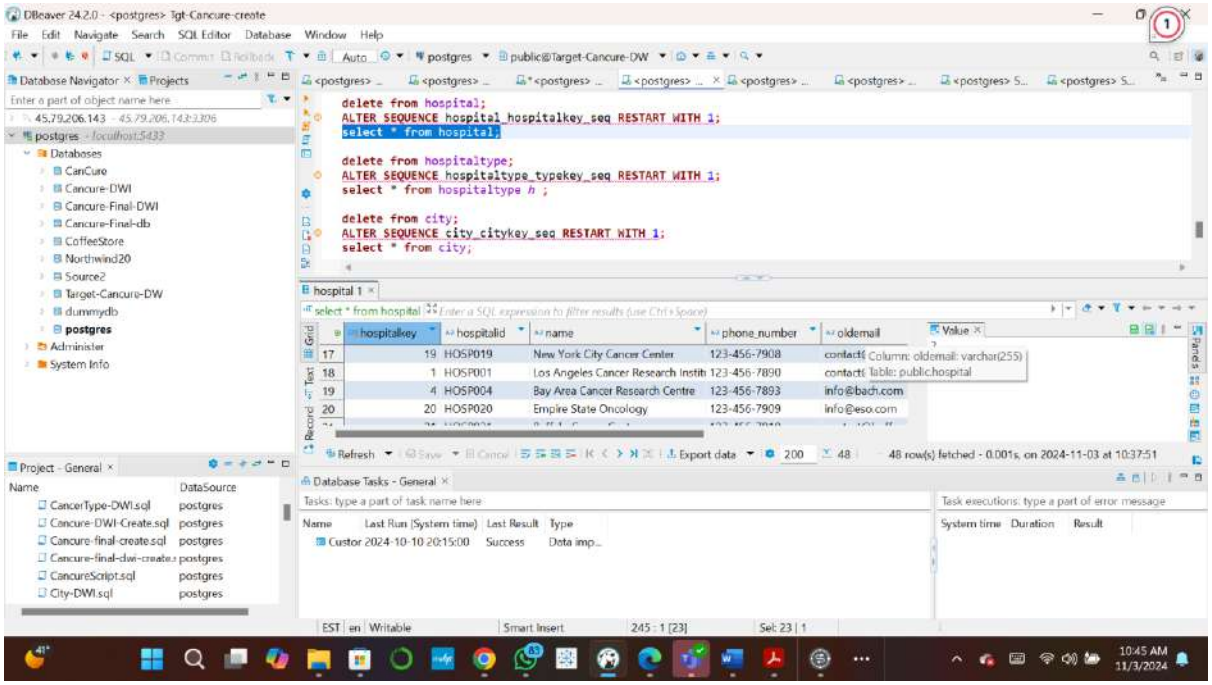
Updating names of 2 records, in relational DB:



Executing the update flow



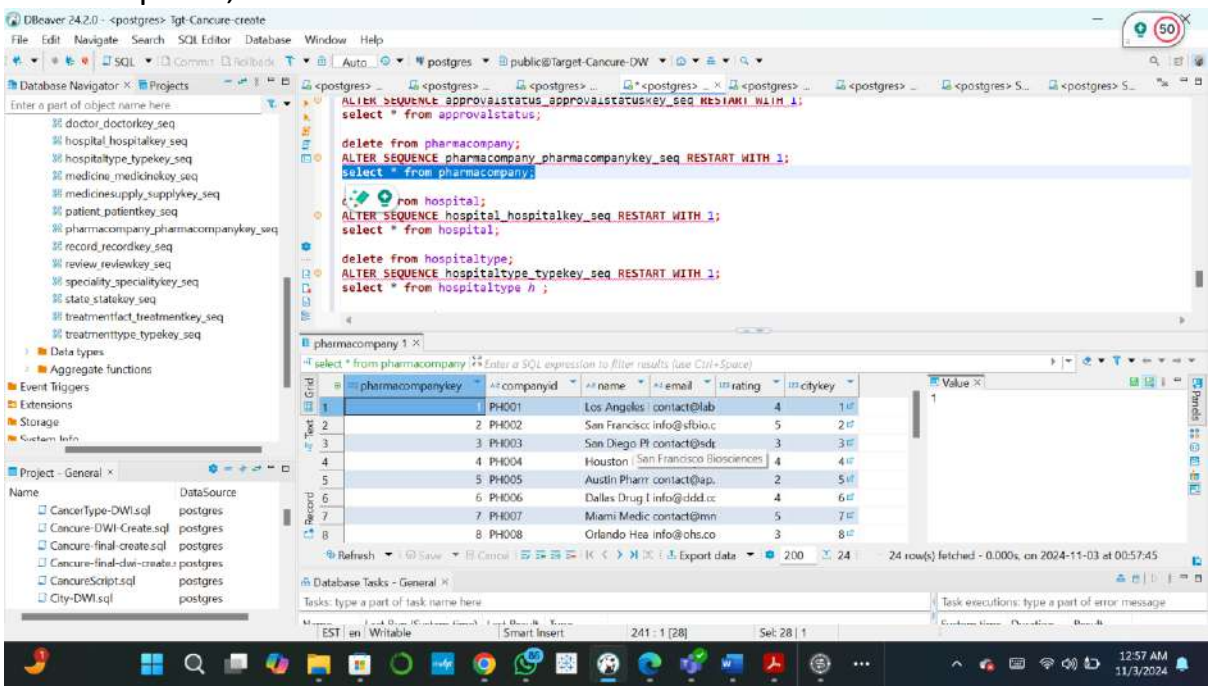
Running the **“Select * from Hospital”** in DatawarehouseDB to see if the update flow is successful,



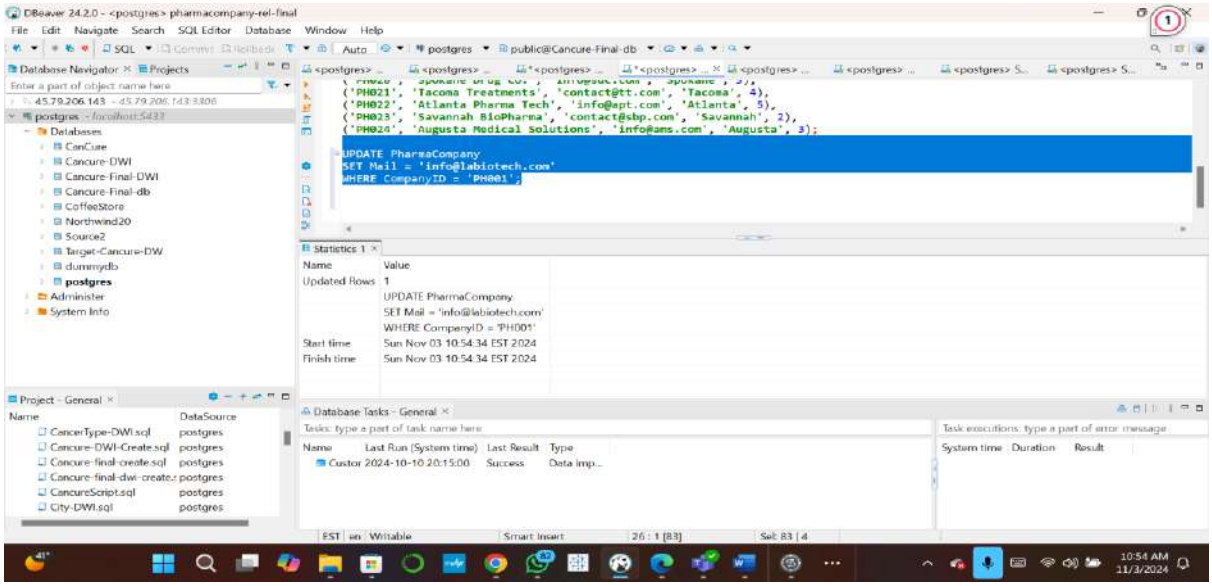
The change is successfully implemented, executed and reflected in the Data Warehouse DB. An **update of the Name** in hospitalID one and four is successfully reflecting

Pharma company:

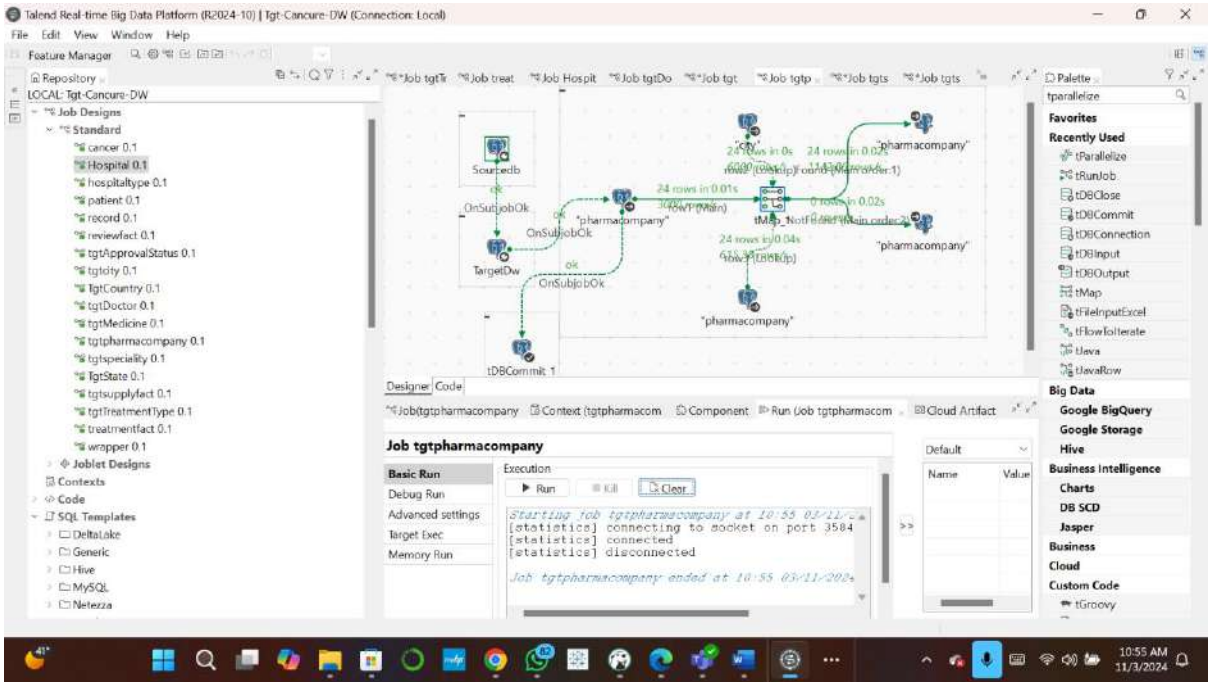
Before Update,



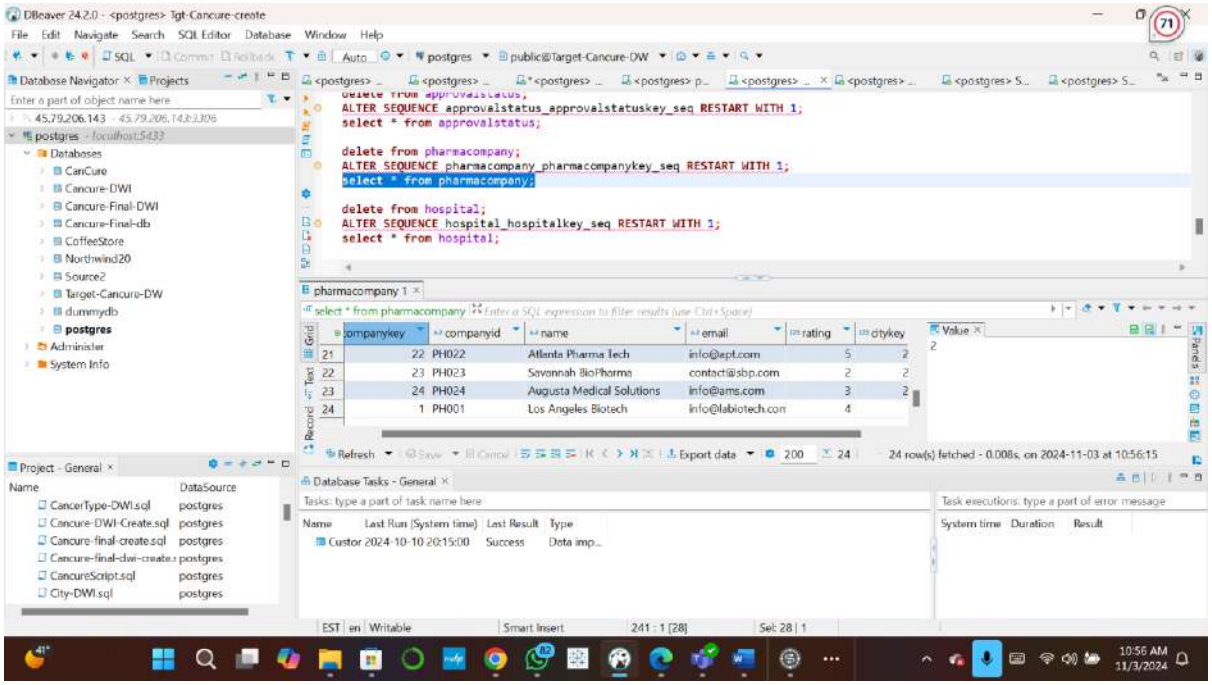
Updating the row1 email from 'contact@labiotech.com' to 'info@labiotech.com' in relational model,



Running the update flow in Talend,

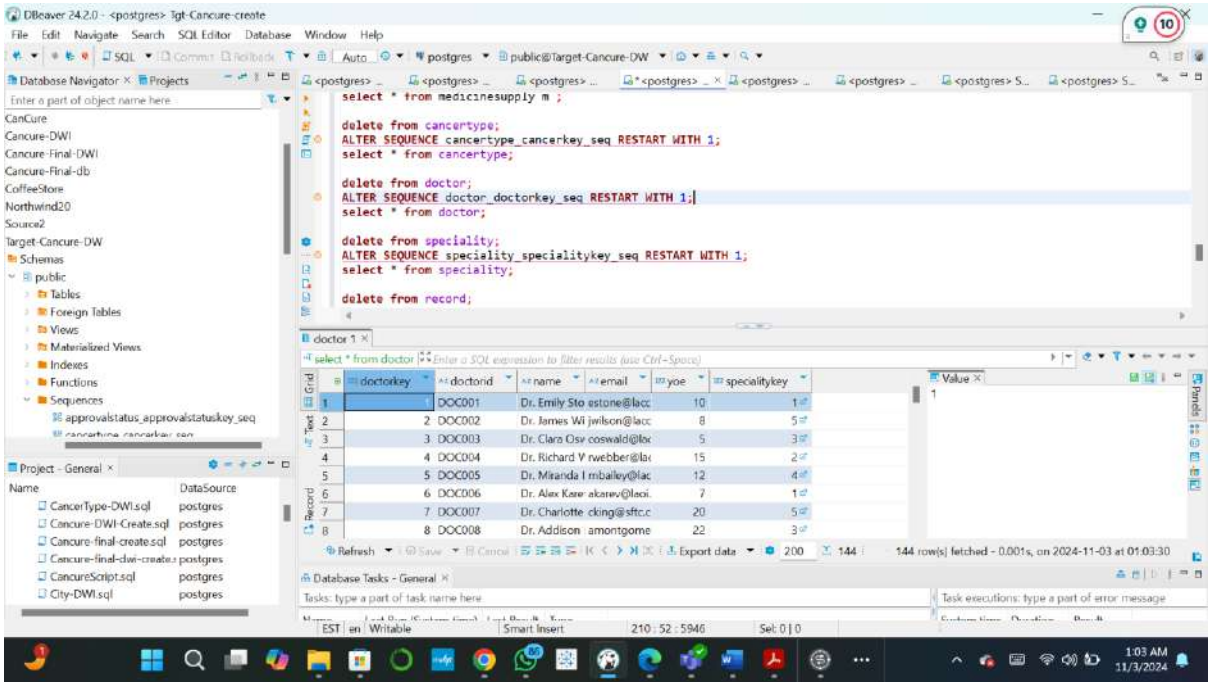


The update is reflecting in the pharma company dimension table,as we can see the updated email.

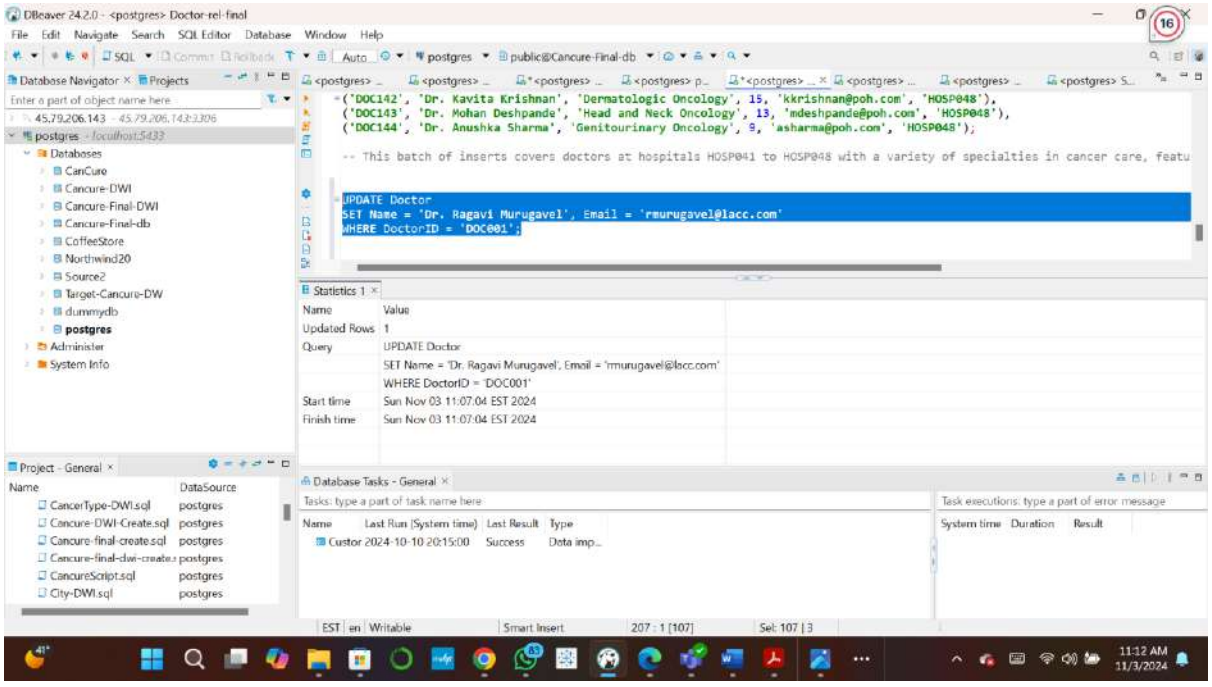


Doctor:

Before the Update, the loaded data in the data warehouse shows the following output

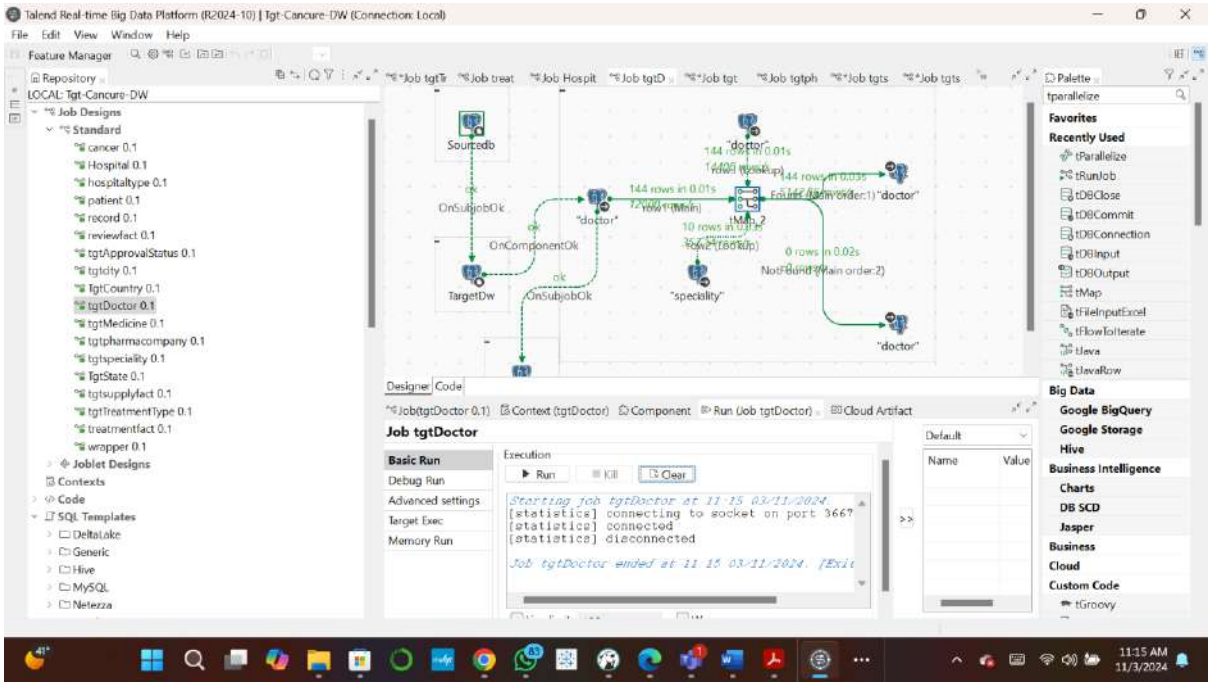


Now I am updating the the row in my relational database,

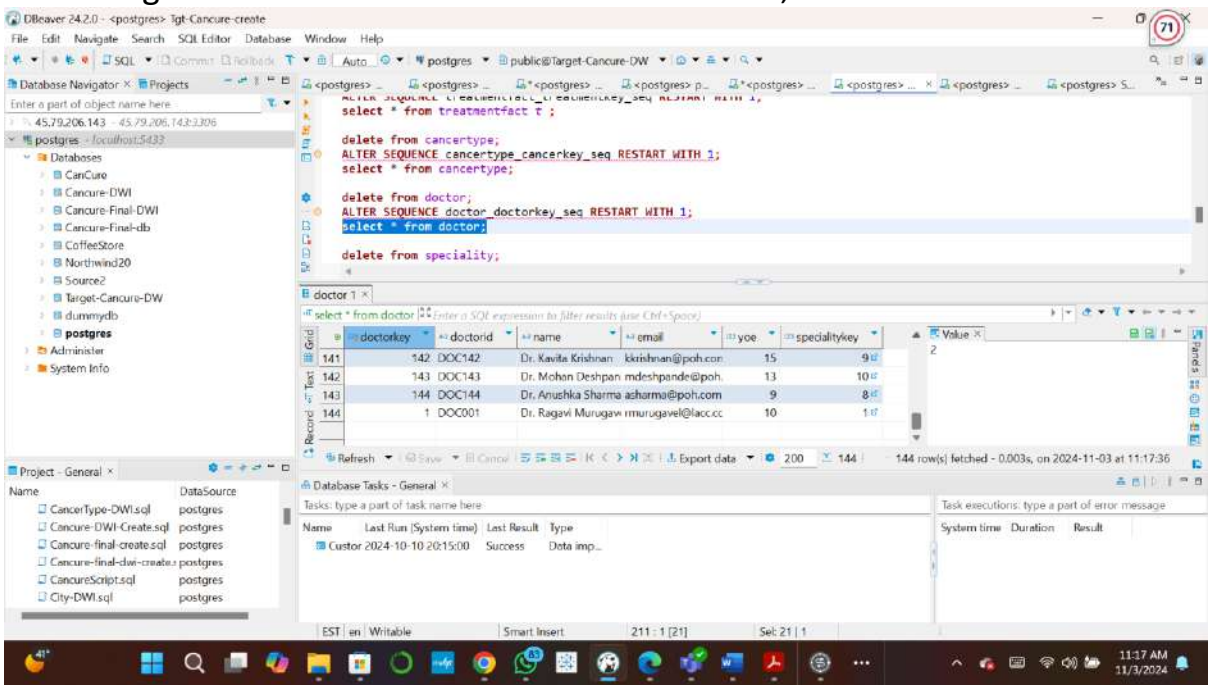


Changing the name and email.

Job is executed,

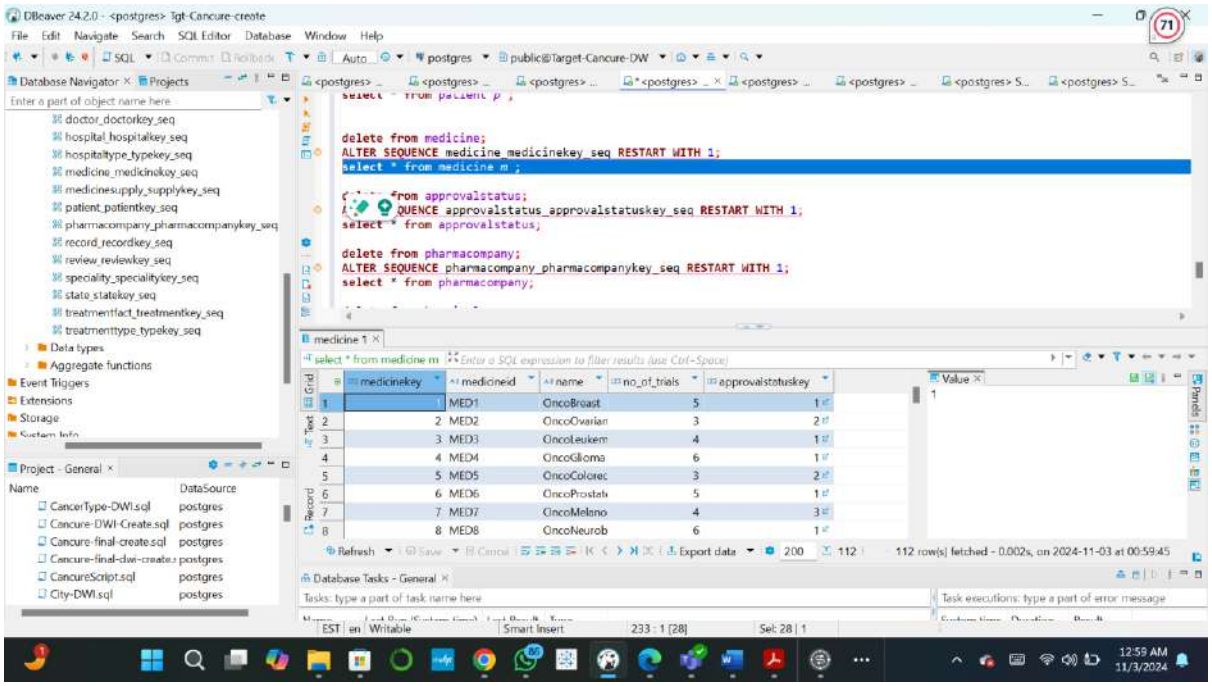


The change is reflected in the datawarehouse table,

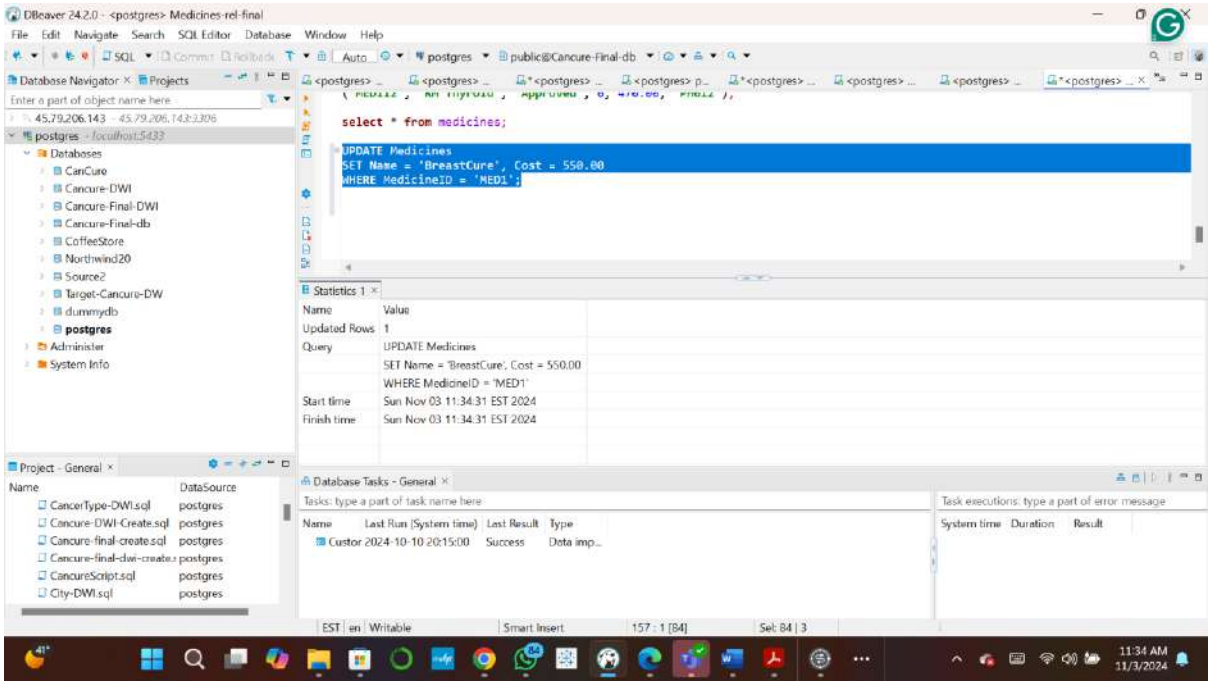


Medicines:

Before Update

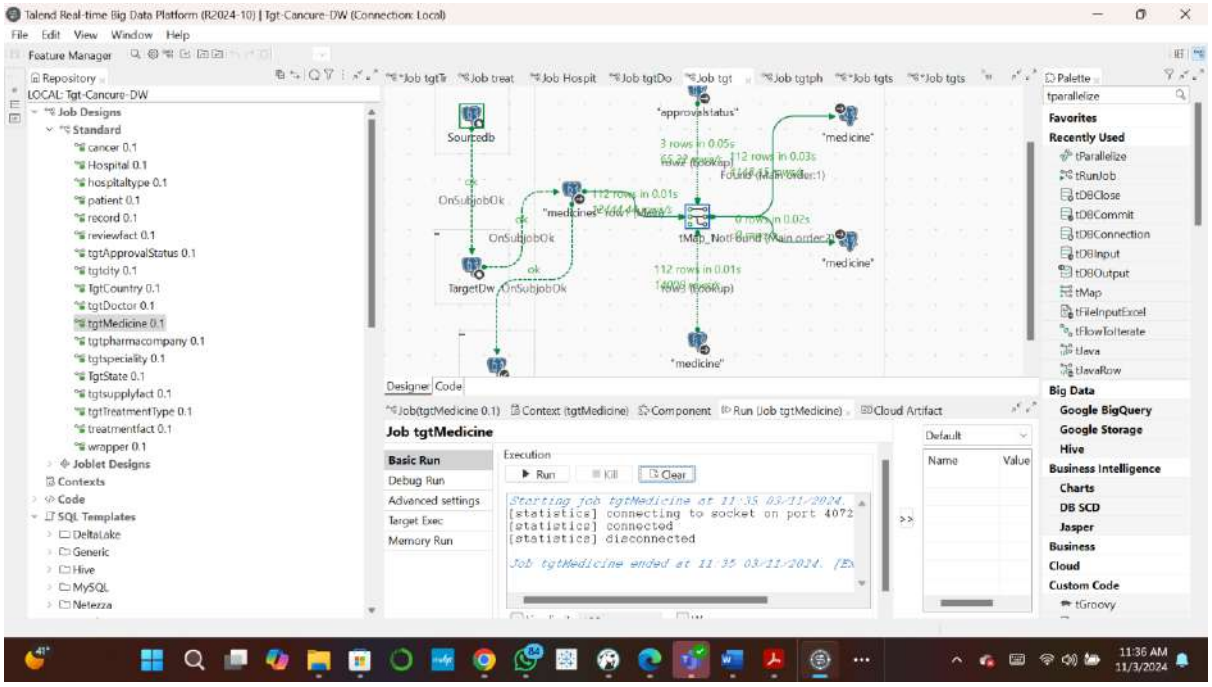


Now updating the rows in the relational model,

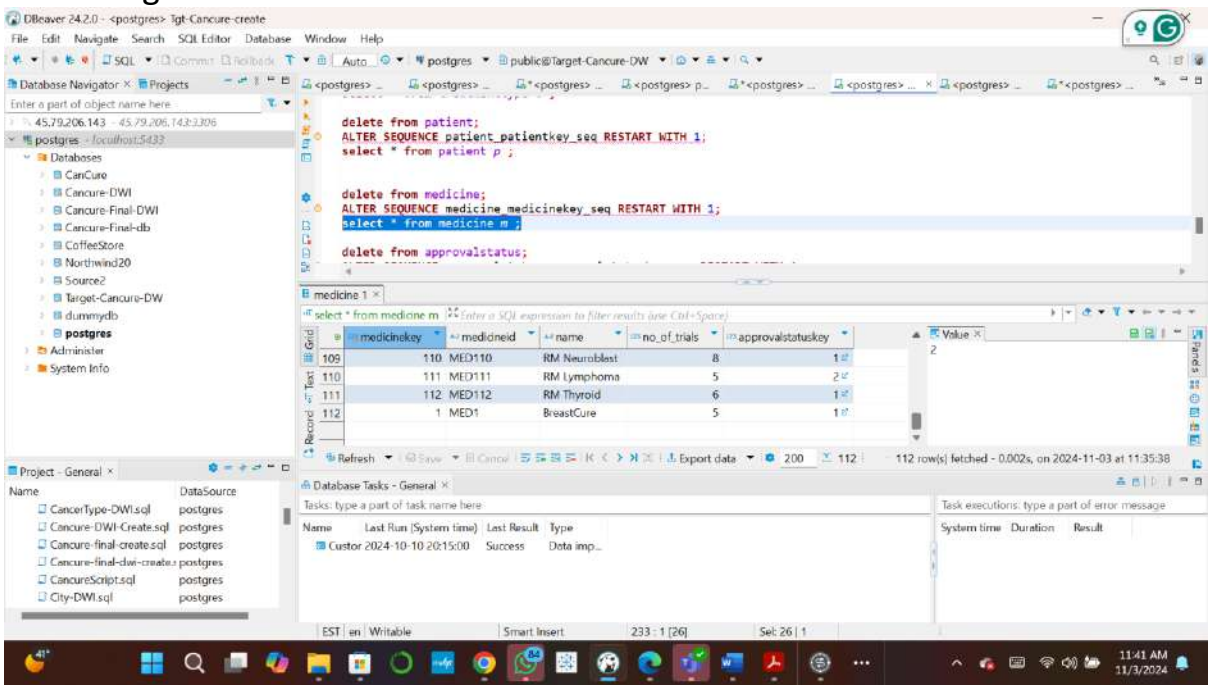


I am changing the name and cost for one record,

Running the medicine job

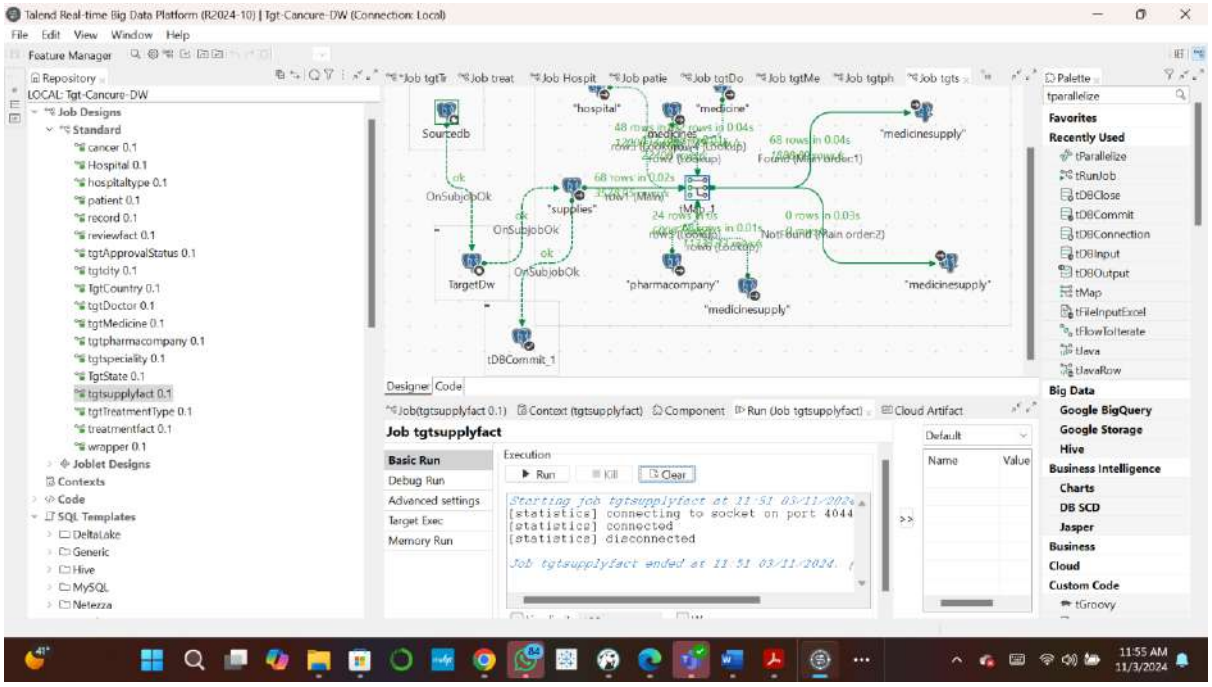


The changes are reflected in the medicine data warehouse table



The name change is reflected.

Now running the supply fact table,



The price changes is also reflected to 550 and correspondingly margin per unit becomes the selling price-supply cost(cost price)[1000-550]=450 which is the margin per cost for medicine key 1. The changes are reflected in the supplyfact table screenshot below.

```
delete from medicinesupply;
ALTER SEQUENCE medicinesupply_supplykey_seq RESTART WITH 1;
select * from medicinesupply m ;

delete from treatmentfact;
ALTER SEQUENCE treatmentfact_treatmentkey_seq RESTART WITH 1;
select * from treatmentfact t ;

delete from cancertype;
```

medicinesupply 1

select * from medicinesupply m | Enter a SQL expression to filter results (use Ctrl+Space)

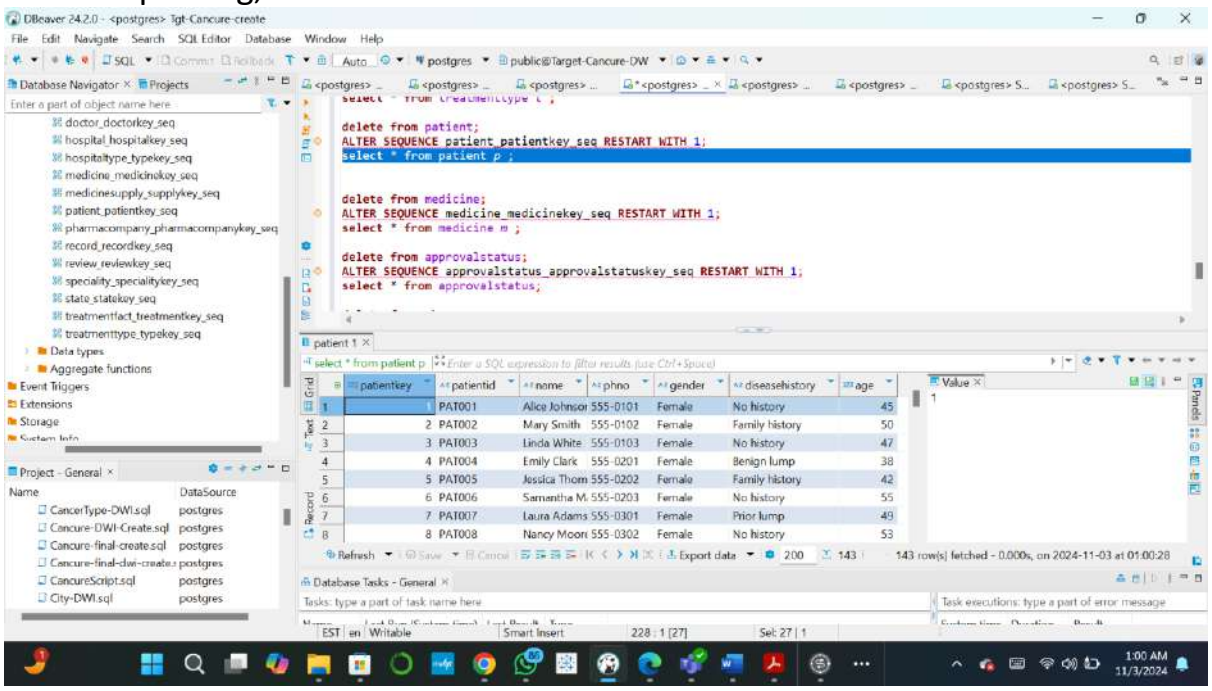
	medicinekey	pharmacompanykey	supplycost	quantity	sellingprice	marginperunit	leadtim
19	1	1	550	500	1,000	450	
20	2	1	610	400	1,000	390	
21	3	1	450	300	1,000	550	
22	4	1	720	500	1,000	280	

Refresh | Save | Cancel | Export data | 200 | 68

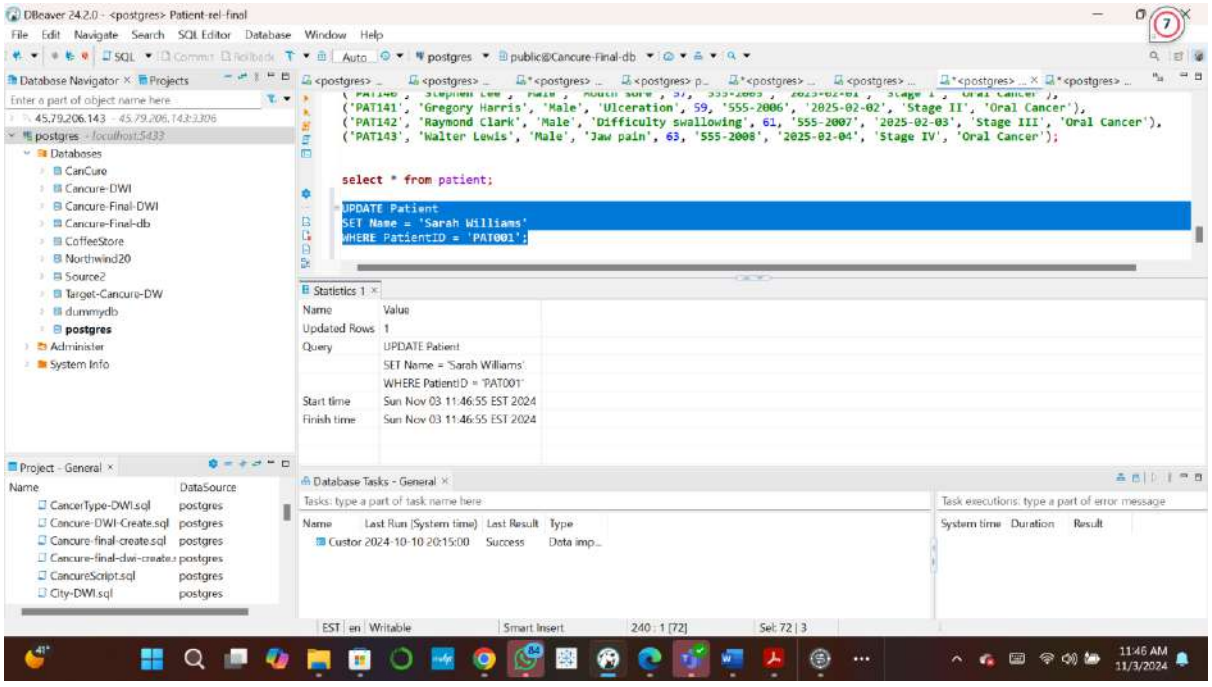
Database Tasks - General

Patient:

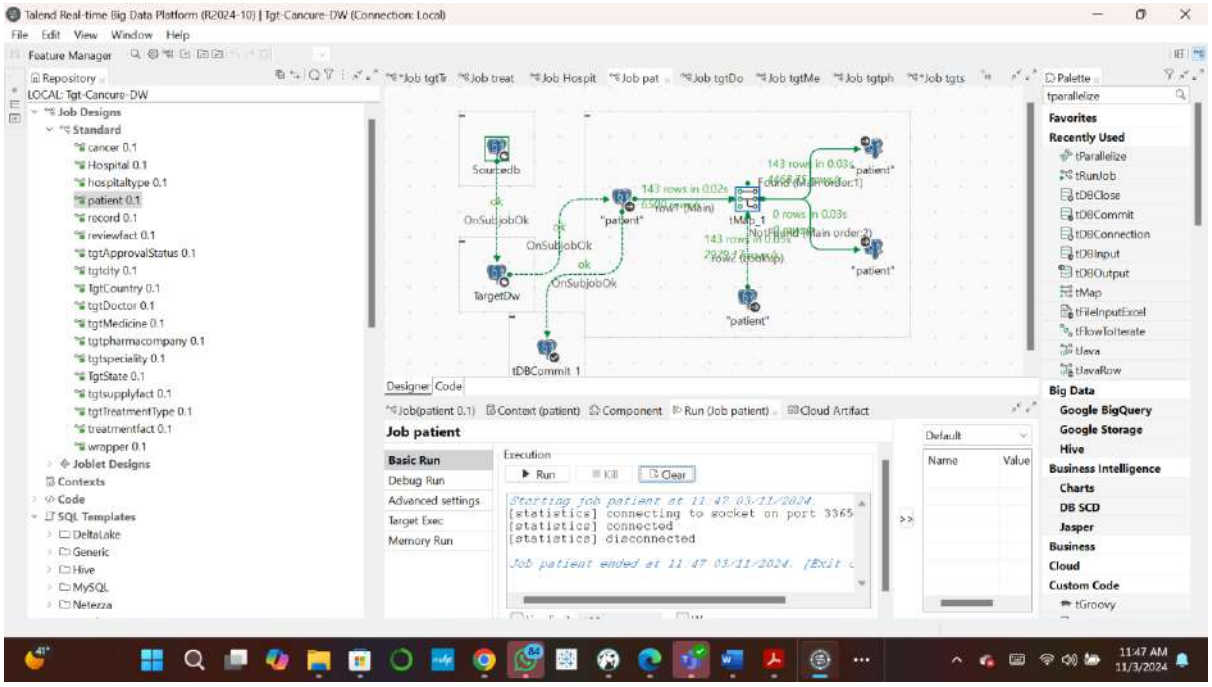
Before Updating, the Datawarehouse table



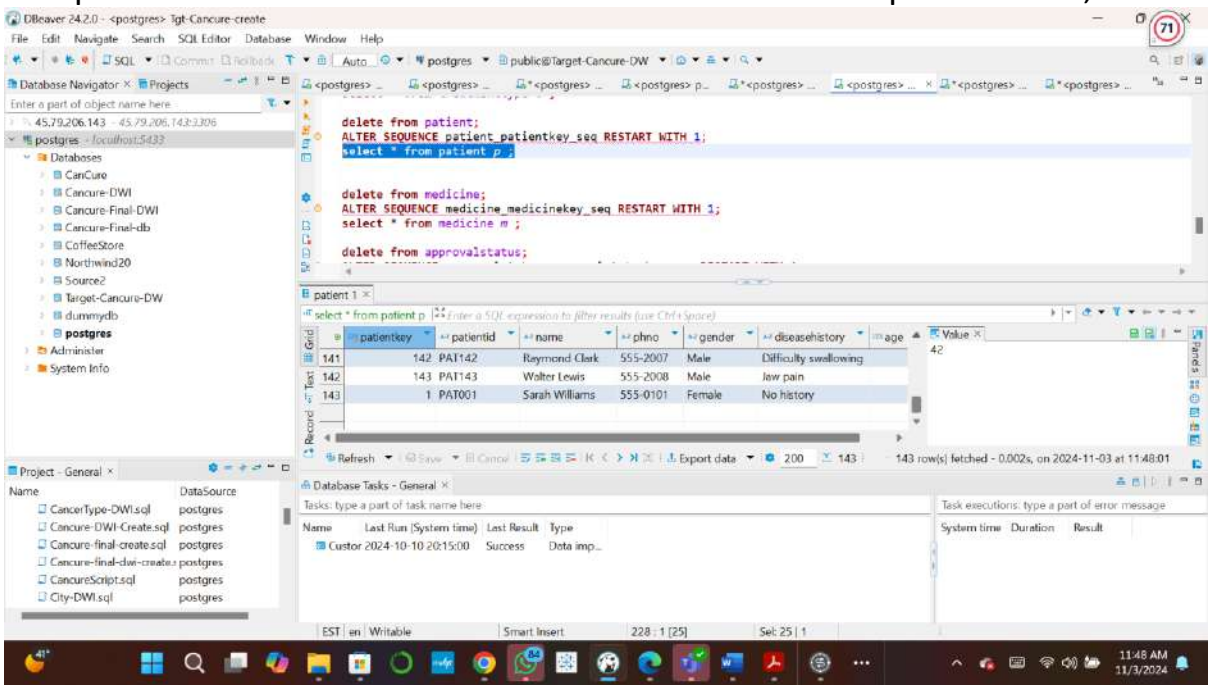
Updating in a relational database, I am updating the name of the patient



Run the update job,



An update of the name is reflected in the data warehouse patient table,



Thus the Update flows are working properly.

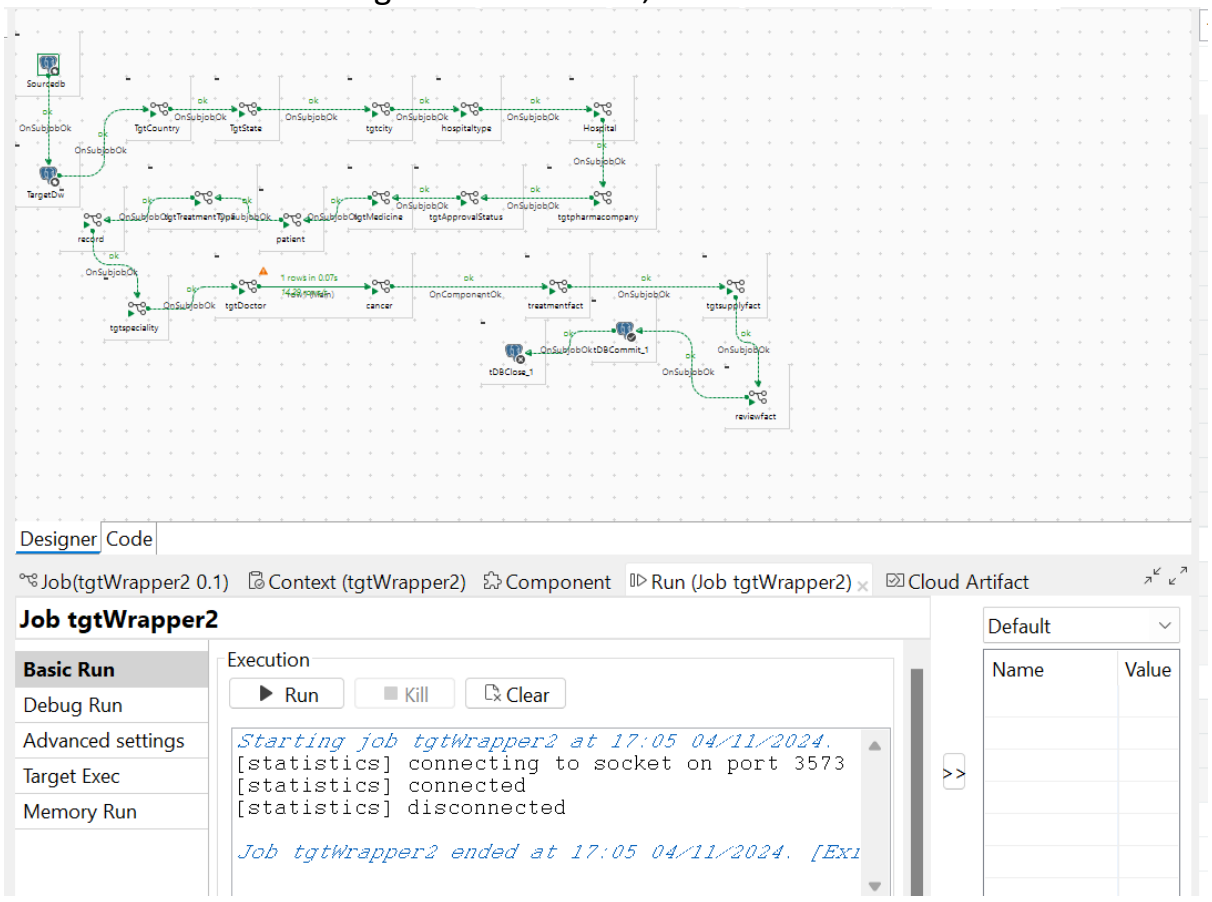
Control Flow:

Now we are again deleting all the data from our data warehouse and confirming if its deleted using the following sql queries,

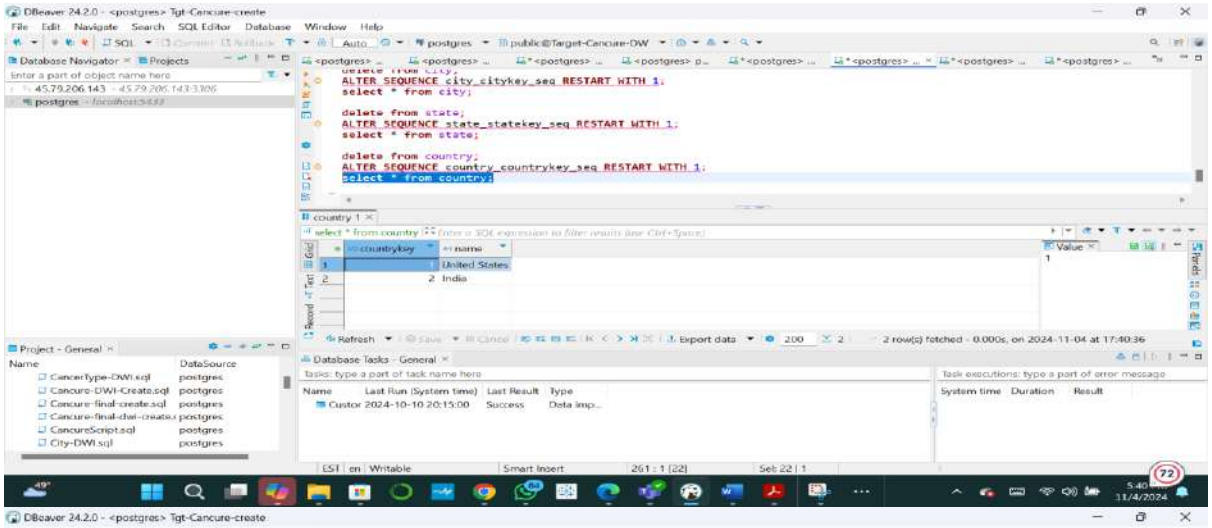
```
delete from review ;
ALTER SEQUENCE review_reviewkey_seq RESTART WITH 1;
select * from review;
delete from medicinesupply;
ALTER SEQUENCE medicinesupply_supplykey_seq RESTART WITH 1;
select * from medicinesupply m ;
delete from treatmentfact;
ALTER SEQUENCE treatmentfact_treatmentkey_seq RESTART WITH 1;
select * from treatmentfact t ;
delete from cancertype;
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
select * from cancertype;
delete from doctor;
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
delete from speciality;
ALTER SEQUENCE speciality_specialitykey_seq RESTART WITH 1;
select * from speciality;
delete from record;
ALTER SEQUENCE record_recordkey_seq RESTART WITH 1;

select * from record r ;
delete from treatmenttype;
ALTER SEQUENCE treatmenttype_typekey_seq RESTART WITH 1;
select * from treatmenttype t ;
delete from patient;
ALTER SEQUENCE patient_patientkey_seq RESTART WITH 1;
select * from patient p ;
delete from medicine;
ALTER SEQUENCE medicine_medicinekey_seq RESTART WITH 1;
select * from medicine m ;
delete from approvalstatus;
ALTER SEQUENCE approvalstatus_approvalstatuskey_seq RESTART WITH 1;
select * from approvalstatus;
delete from pharmacompany;
ALTER SEQUENCE pharmacompany_pharmacompanykey_seq RESTART WITH 1;
```

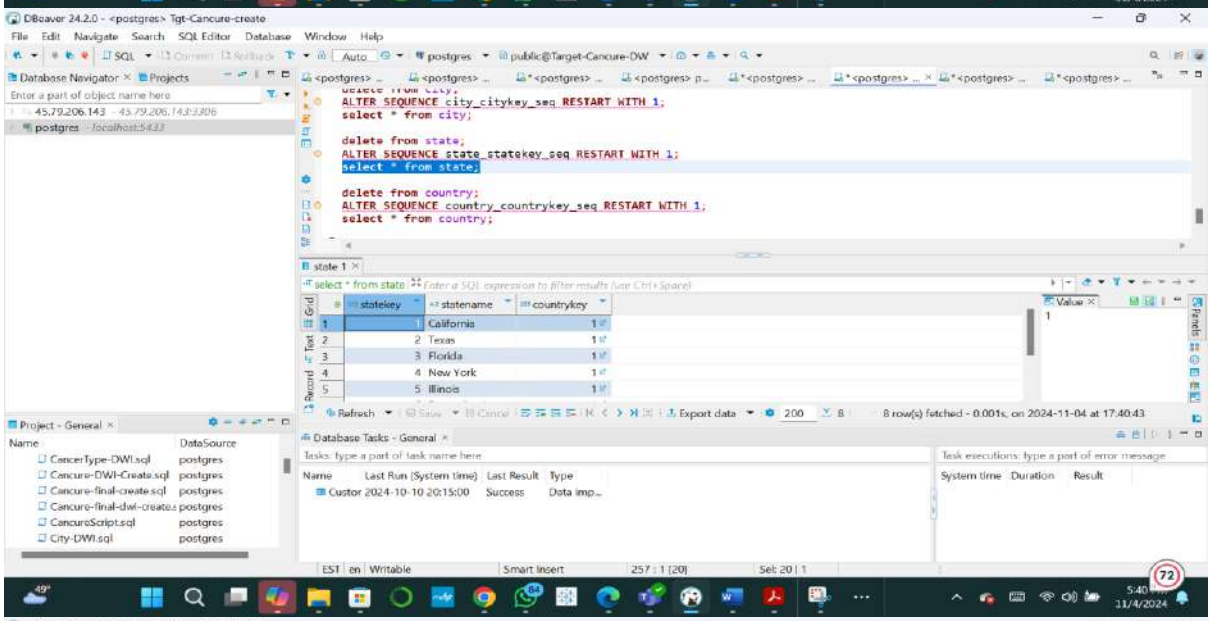
```
select * from pharmacompany;
delete from hospital;
ALTER SEQUENCE hospital_hospitalkey_seq RESTART WITH 1;
select * from hospital;
delete from hospitaltype;
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype h ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
delete from country;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
```

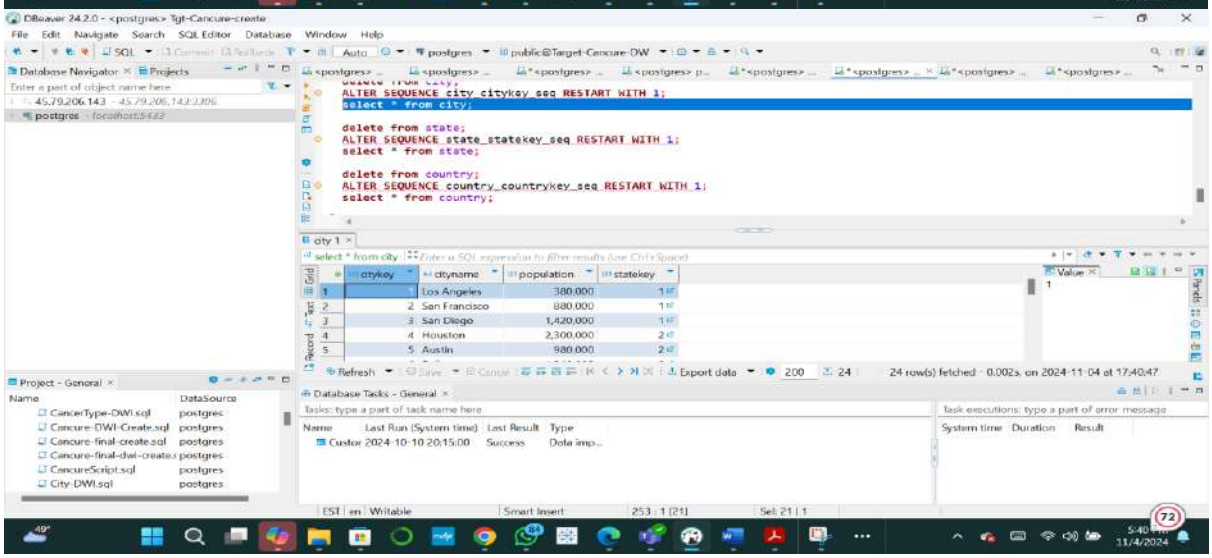
After this we are running the control flow,

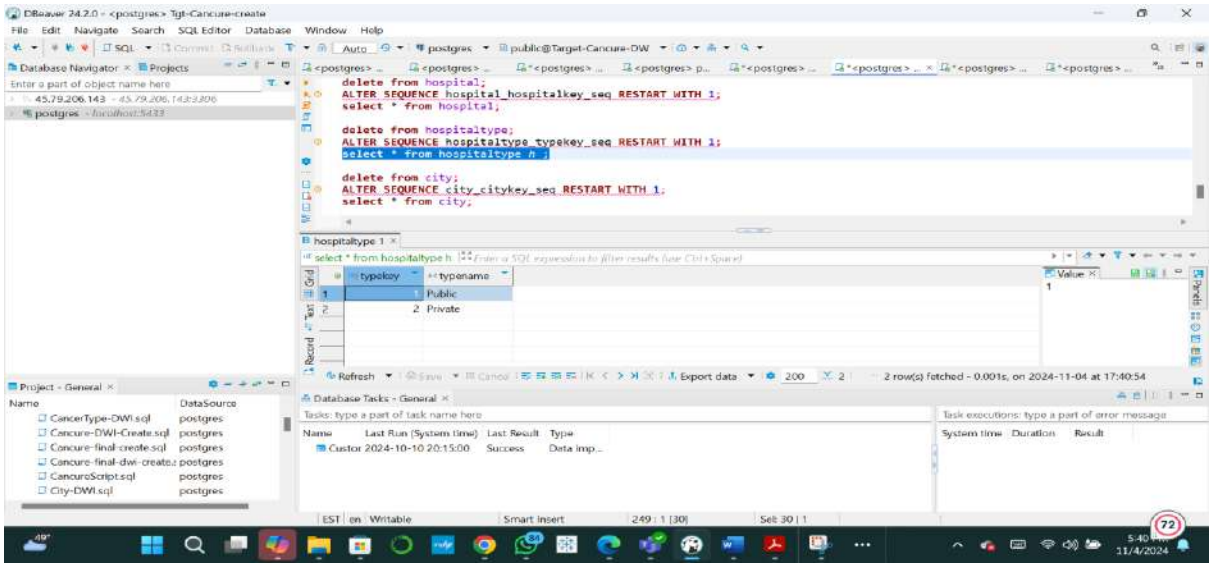


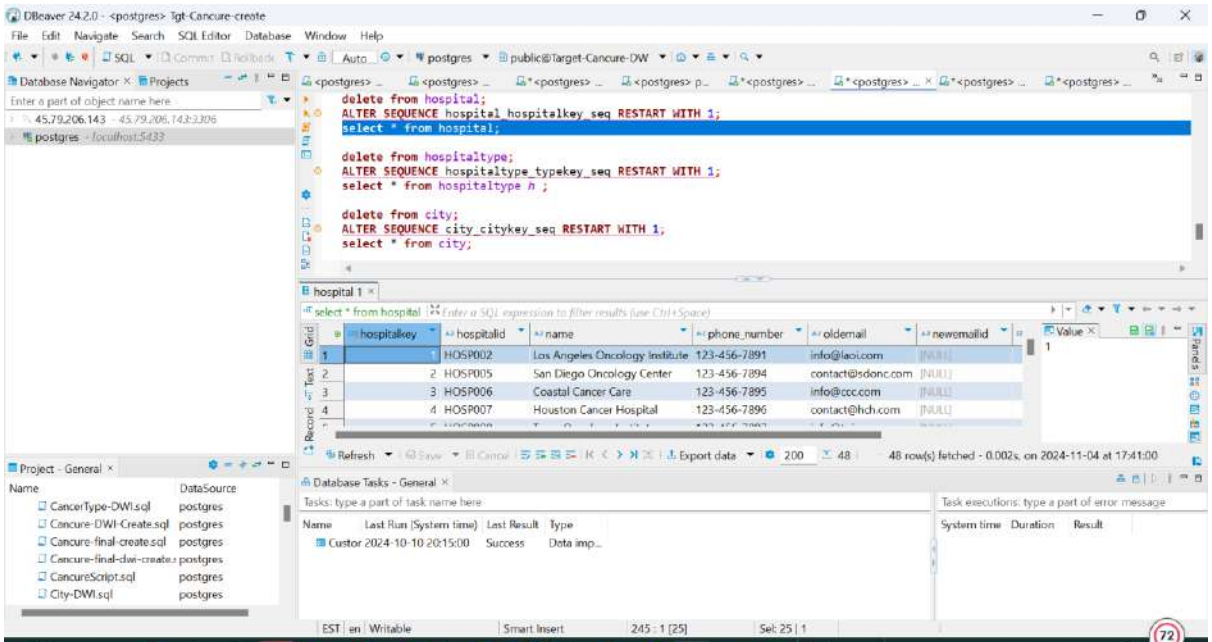
After running it successfully all the data is populated in all the tables again,the proof is attached below,

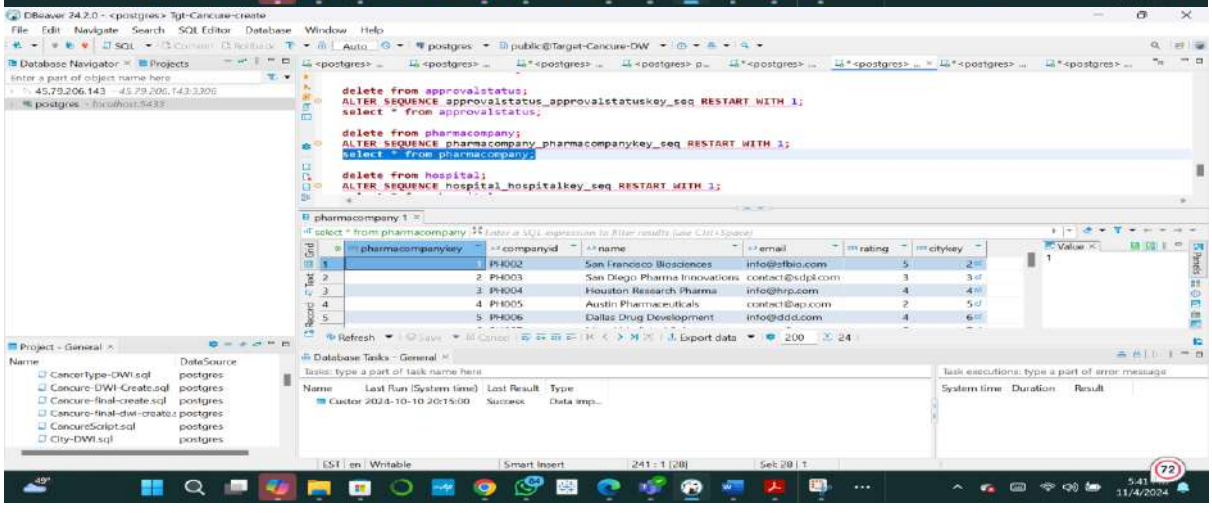

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
delete from country;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
country 1
countrykey
name
1 United States
2 India
2 row(s) fetched - 0.000s, on 2024-11-04 at 17:40:36

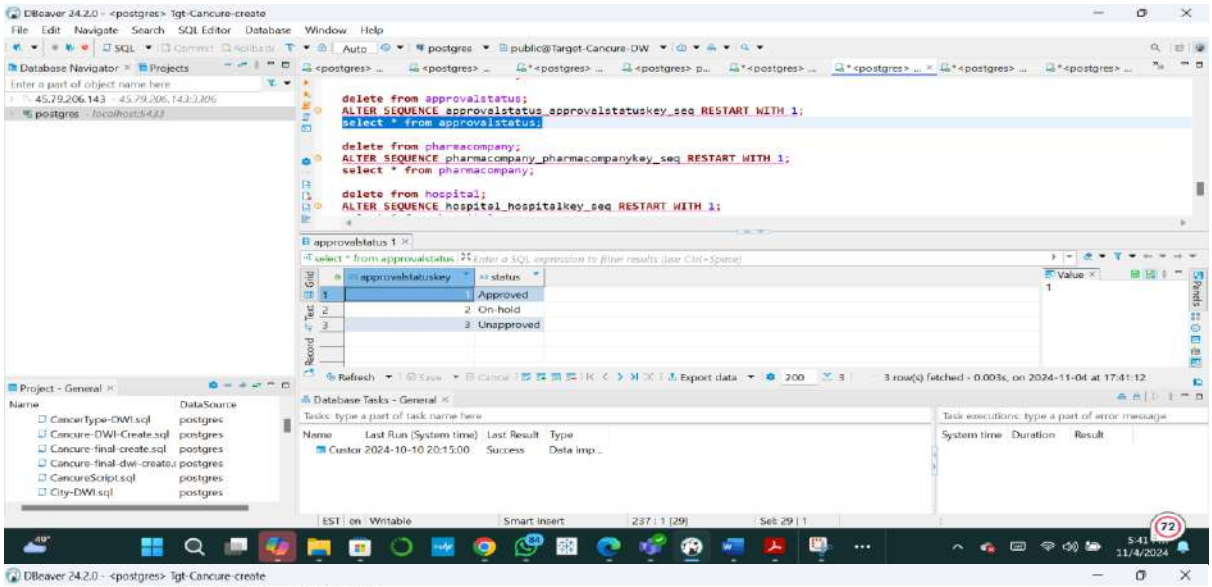

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
delete from country;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
state 1
statekey
statename
countrykey
1 California 1
2 Texas 1
3 Florida 1
4 New York 1
5 Illinois 1
8 row(s) fetched - 0.001s, on 2024-11-04 at 17:40:43

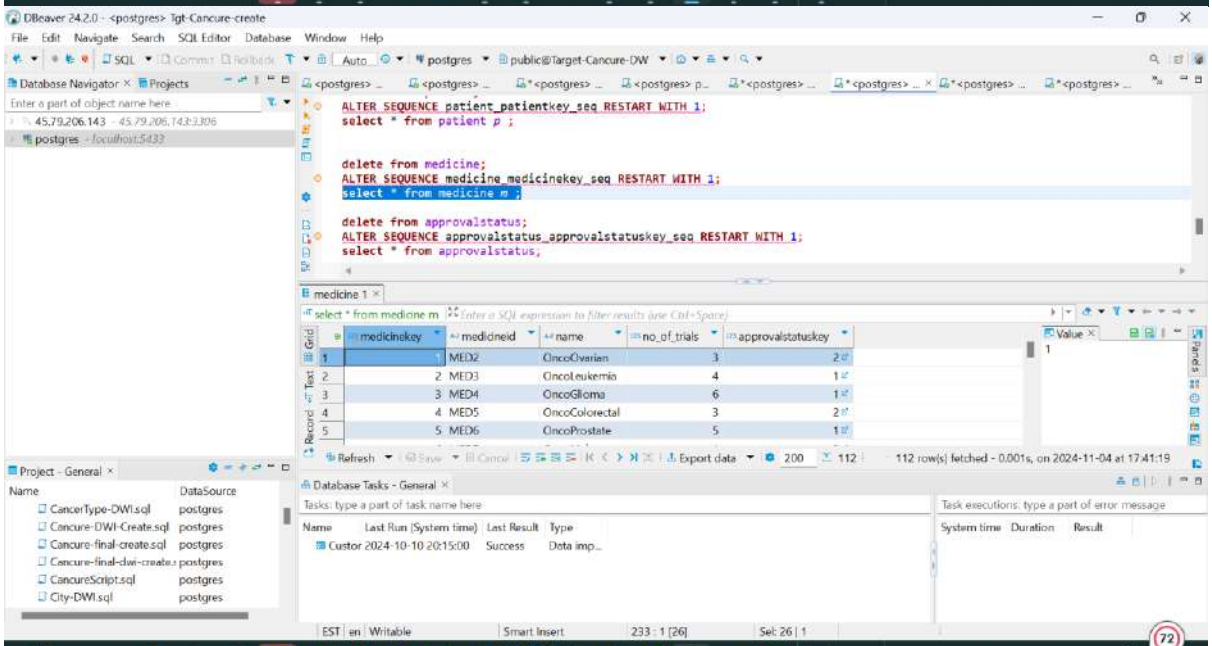

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
delete from state;
ALTER SEQUENCE state_statekey_seq RESTART WITH 1;
select * from state;
delete from country;
ALTER SEQUENCE country_countrykey_seq RESTART WITH 1;
select * from country;
city 1
citykey
cityname
population
statekey
1 Los Angeles 380,000 1
2 San Francisco 880,000 1
3 San Diego 1,420,000 1
4 Houston 2,300,000 2
5 Austin 980,000 2
24 row(s) fetched - 0.002s, on 2024-11-04 at 17:40:47

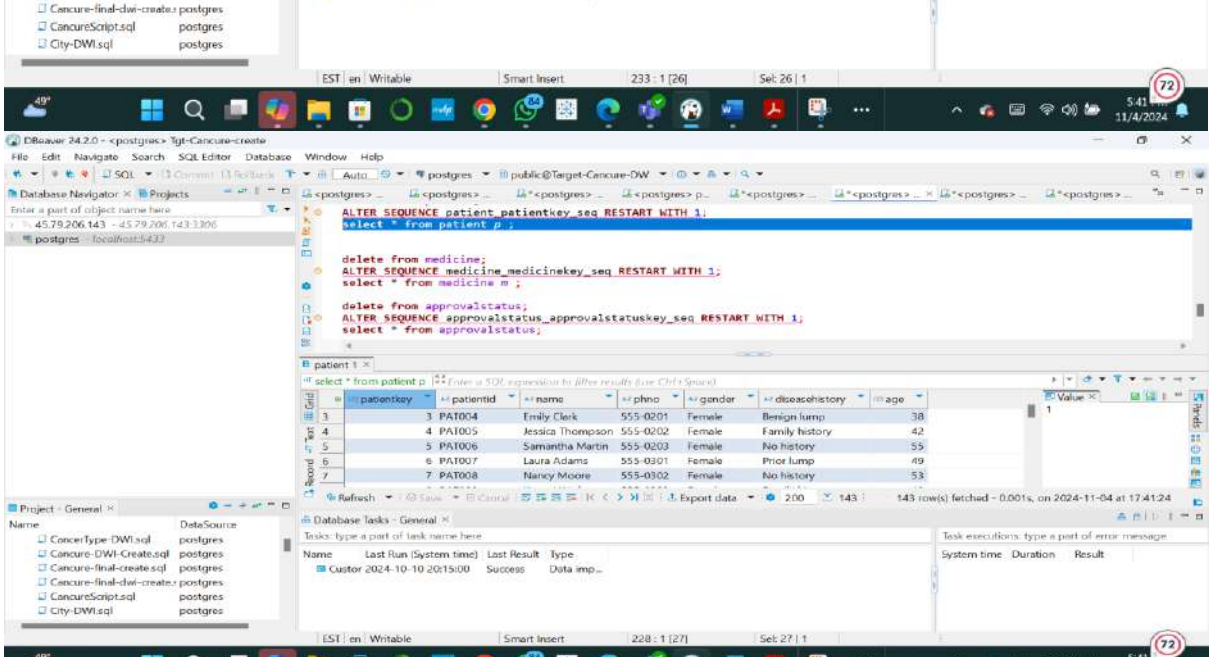

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator Projects
postgres - localhost:5433
delete from hospital;
ALTER SEQUENCE hospital_hospitalkey_seq RESTART WITH 1;
select * from hospital;
delete from hospitaltype;
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype h ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
hospitaltype 1
typekey
typename
1 Public
2 Private
2 row(s) fetched - 0.001s, on 2024-11-04 at 17:40:54
Project - General
Name
DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
Cancure-final-dwi-create.s postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Name
Last Run (System time)
Last Result
Type
Custor 2024-10-10 20:15:00
Success
Data imp...
Task executions: type a part of error message
System time
Duration
Result
EST | en | Writable | Smart Insert | 249 : 1 [30] | Sel: 30 | 1
11/4/2024

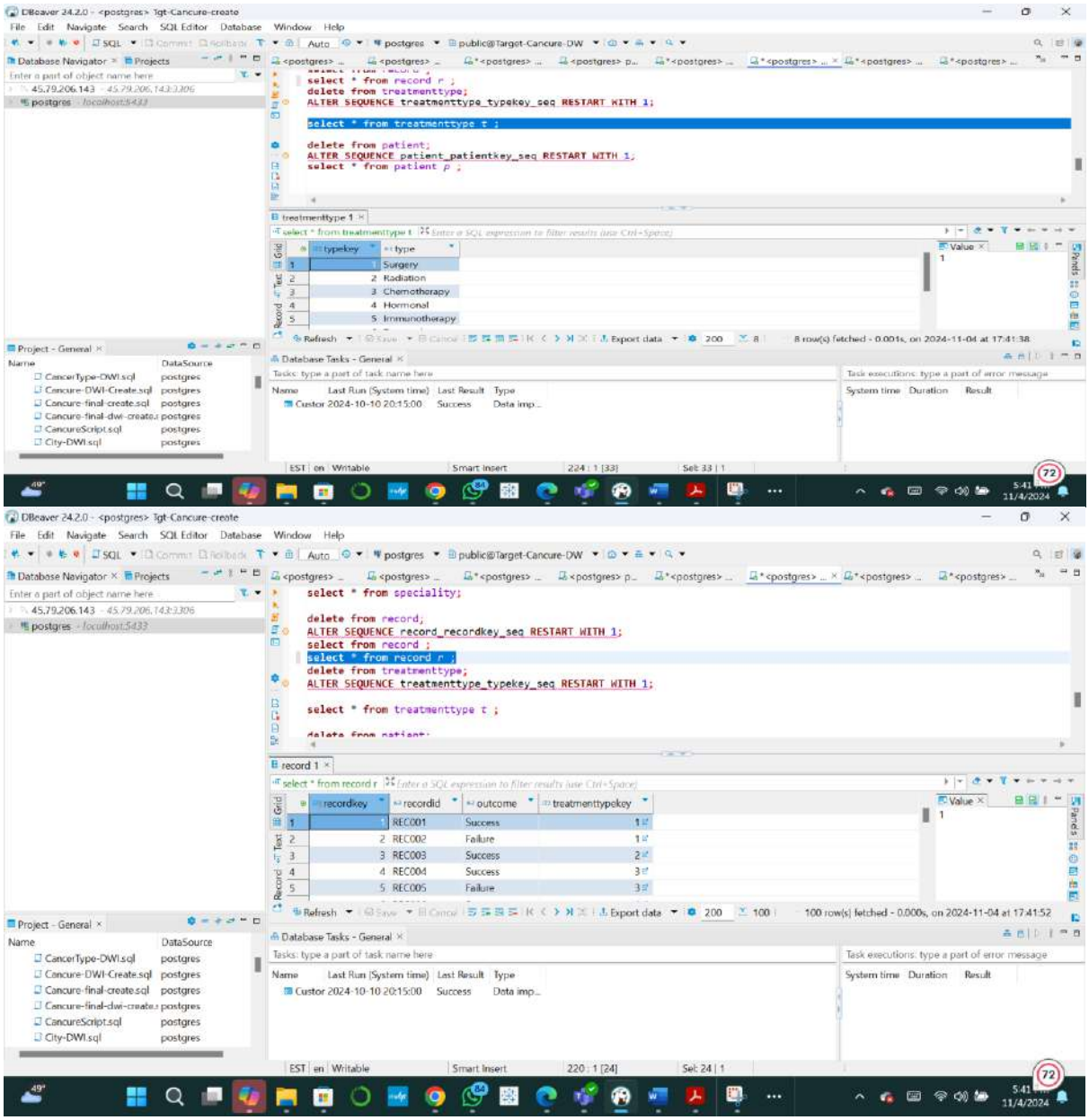

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator Projects
postgres - localhost:5433
delete from hospital;
ALTER SEQUENCE hospital_hospitalkey_seq RESTART WITH 1;
select * from hospital;
delete from hospitaltype;
ALTER SEQUENCE hospitaltype_typekey_seq RESTART WITH 1;
select * from hospitaltype h ;
delete from city;
ALTER SEQUENCE city_citykey_seq RESTART WITH 1;
select * from city;
hospital 1
hospitalkey
hospitalid
name
phone_number
oldemail
newemailid
1 HOSP002 Los Angeles Oncology Institute 123-456-7891 info@laoi.com
2 HOSP005 San Diego Oncology Center 123-456-7894 contact@sdonc.com
3 HOSP006 Coastal Cancer Care 123-456-7895 info@ccc.com
4 HOSP007 Houston Cancer Hospital 123-456-7896 contact@hch.com
48 row(s) fetched - 0.002s, on 2024-11-04 at 17:41:00
Project - General
Name
DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
Cancure-final-dwi-create.s postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Name
Last Run (System time)
Last Result
Type
Custor 2024-10-10 20:15:00
Success
Data imp...
Task executions: type a part of error message
System time
Duration
Result
EST | en | Writable | Smart Insert | 245 : 1 [25] | Sel: 25 | 1
11/4/2024


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator Projects
postgres - localhost:5433
delete from approvalstatus;
ALTER SEQUENCE approvalstatus_approvalstatuskey_seq RESTART WITH 1;
select * from approvalstatus;
delete from pharmacompany;
ALTER SEQUENCE pharmacompany_pharmacompanykey_seq RESTART WITH 1;
select * from pharmacompany;
delete from hospital;
ALTER SEQUENCE hospital_hospitalkey_seq RESTART WITH 1;
pharmacompany 1
pharmacompanykey
companyid
name
email
rating
citykey
1 PH002 San Francisco Biosciences info@sfbio.com 5 2
2 PH003 San Diego Pharma Innovations contact@sdpi.com 3 3
3 PH004 Houston Research Pharma info@hrp.com 4 4
4 PH005 Austin Pharmaceuticals contact@ap.com 2 5
5 PH006 Dallas Drug Development info@ddd.com 4 6
24 row(s) fetched - 0.001s, on 2024-11-04 at 17:41:31
Project - General
Name
DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
Cancure-final-dwi-create.s postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Name
Last Run (System time)
Last Result
Type
Custor 2024-10-10 20:15:00
Success
Data imp...
Task executions: type a part of error message
System time
Duration
Result
EST | en | Writable | Smart Insert | 241 : 1 [28] | Sel: 28 | 1
11/4/2024

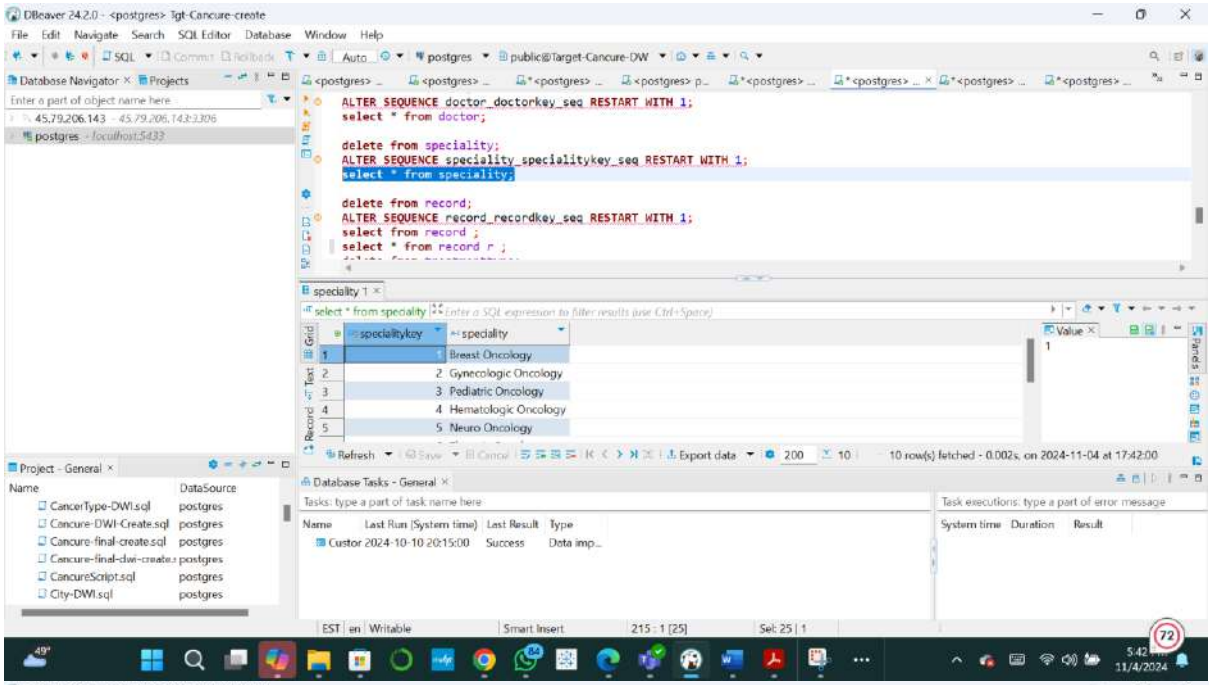

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from approvalstatus;
ALTER SEQUENCE approvalstatus_approvalstatuskey_seq RESTART WITH 1;
select * from approvalstatus;
delete from pharmacompany;
ALTER SEQUENCE pharmacompany_pharmacompanykey_seq RESTART WITH 1;
select * from pharmacompany;
delete from hospital;
ALTER SEQUENCE hospital_hospitalkey_seq RESTART WITH 1;
approvalstatus 1
approvalstatuskey
status
1 Approved
2 On-hold
3 Unapproved
3 row(s) fetched - 0.003s, on 2024-11-04 at 17:41:12
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...

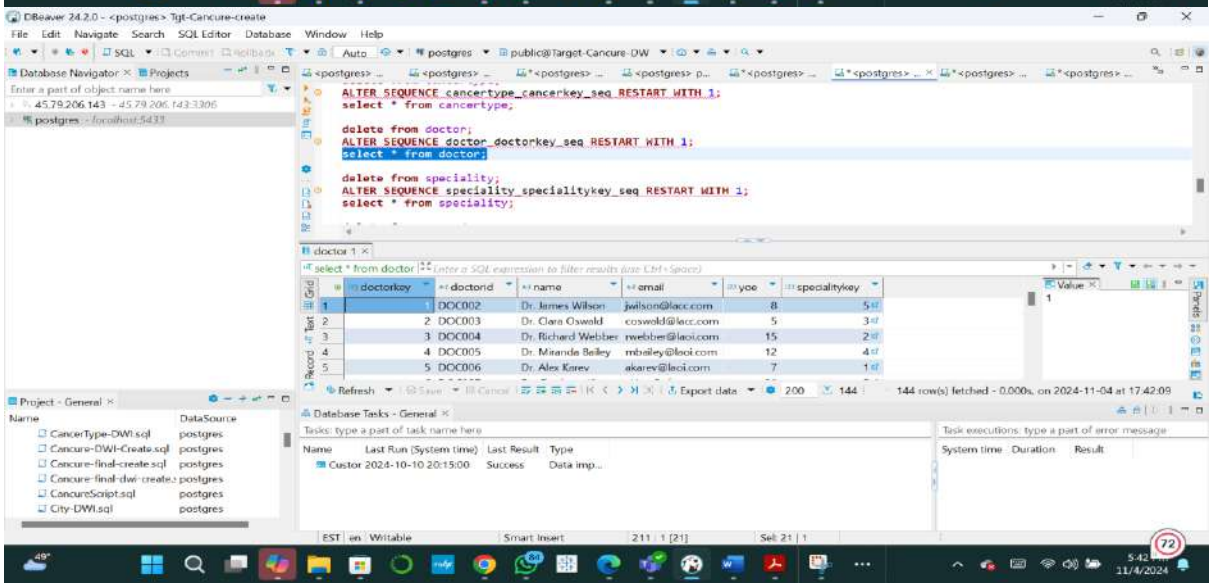

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
ALTER SEQUENCE patient_patientkey_seq RESTART WITH 1;
select * from patient p ;
delete from medicine;
ALTER SEQUENCE medicine_medicinekey_seq RESTART WITH 1;
select * from medicine m ;
delete from approvalstatus;
ALTER SEQUENCE approvalstatus_approvalstatuskey_seq RESTART WITH 1;
select * from approvalstatus;
medicine 1
medicinekey
medicineid
name
no_of_trials
approvalstatuskey
1 MED2 OncoOvarian 3 2
2 MED3 OncoLeukemia 4 1
3 MED4 OncoGlioma 6 1
4 MED5 OncoColorectal 3 2
5 MED6 OncoProstate 5 1
112 row(s) fetched - 0.001s, on 2024-11-04 at 17:41:19
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...

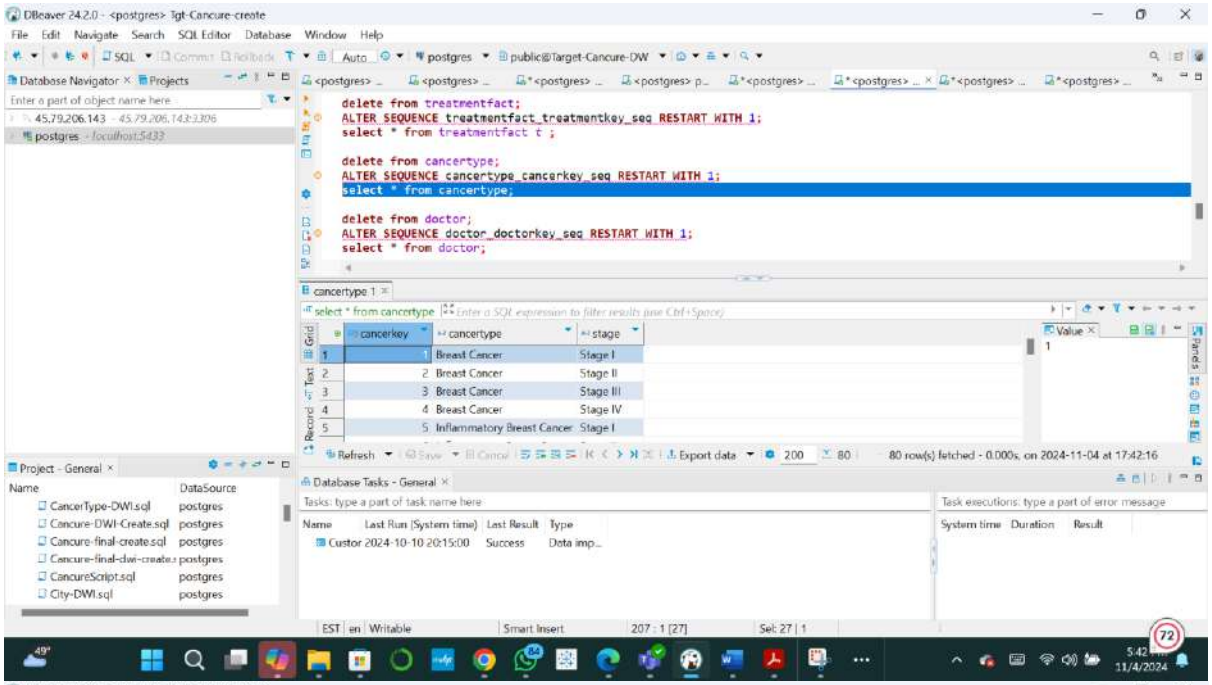

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
ALTER SEQUENCE patient_patientkey_seq RESTART WITH 1;
select * from patient p ;
delete from medicine;
ALTER SEQUENCE medicine_medicinekey_seq RESTART WITH 1;
select * from medicine m ;
delete from approvalstatus;
ALTER SEQUENCE approvalstatus_approvalstatuskey_seq RESTART WITH 1;
select * from approvalstatus;
patient 1
patientkey
patientid
name
phno
gender
diseasehistory
age
3 PAT004 Emily Clark 555-0201 Female Benign lump 38
4 PAT005 Jessica Thompson 555-0202 Female Family history 42
5 PAT006 Samantha Martin 555-0203 Female No history 55
6 PAT007 Laura Adams 555-0301 Female Prior lump 49
7 PAT008 Nancy Moore 555-0302 Female No history 53
143 row(s) fetched - 0.001s, on 2024-11-04 at 17:41:24
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...

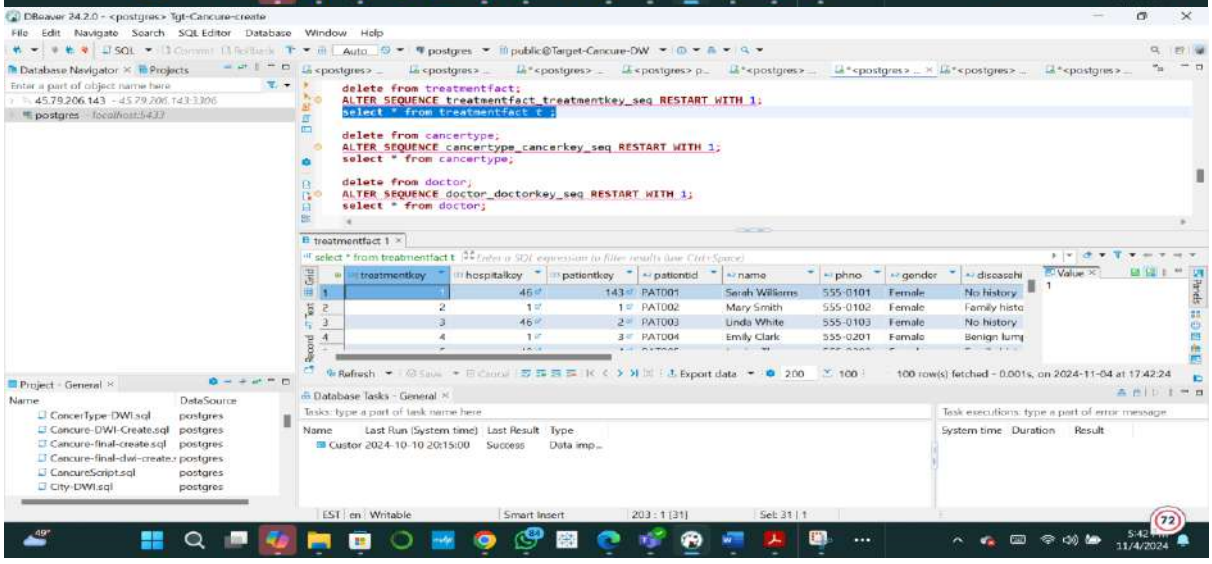

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator Projects
45.79.206.143 - 45.79.206.143:3306
postgres - localhost:5433
select * from record r ;
delete from treatmenttype;
ALTER SEQUENCE treatmenttype_typekey_seq RESTART WITH 1;
select * from treatmenttype t ;
delete from patient;
ALTER SEQUENCE patient_patientkey_seq RESTART WITH 1;
select * from patient p ;
treatmenttype 1
typekey type
1 Surgery
2 Radiation
3 Chemotherapy
4 Hormonal
5 Immunotherapy
8 row(s) fetched - 0.001s, on 2024-11-04 at 17:41:38
Project - General
Name DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
Cancure-final-dwi-create.s postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Name Last Run (System time) Last Result Type
Custor 2024-10-10 20:15:00 Success Data imp...
EST en Writable Smart Insert 224 : 1 [33] Sel: 33 | 1
11/4/2024

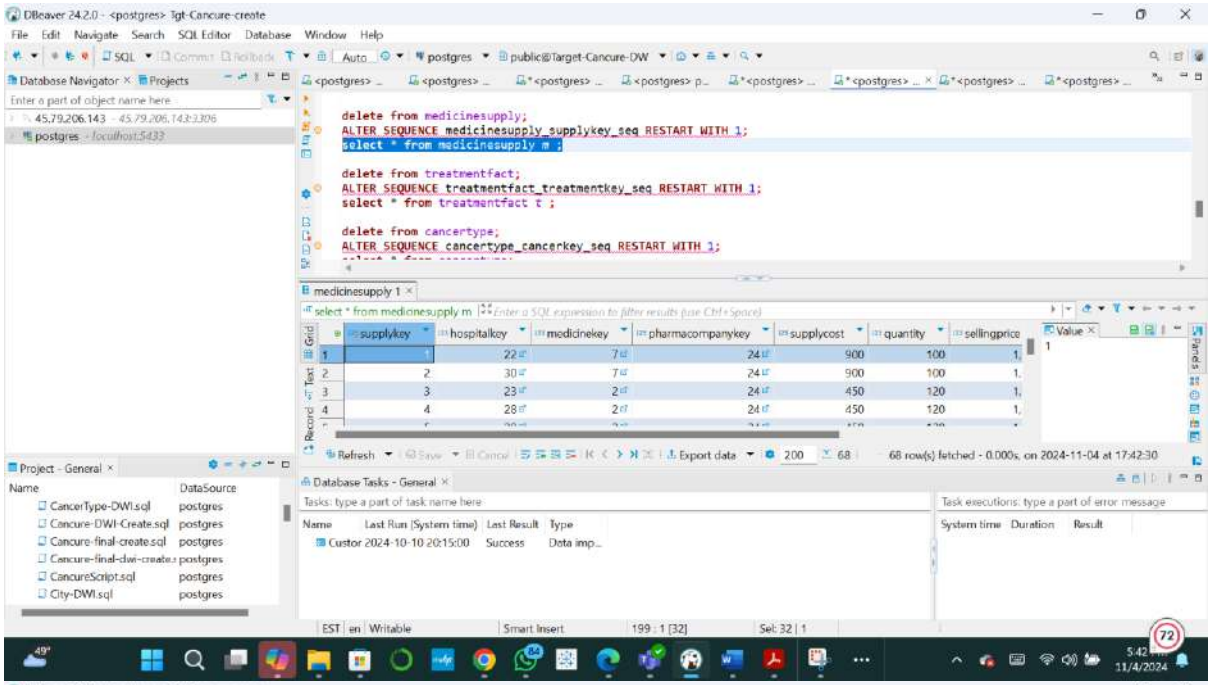
DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator Projects
45.79.206.143 - 45.79.206.143:3306
postgres - localhost:5433
select * from speciality;
delete from record;
ALTER SEQUENCE record_recordkey_seq RESTART WITH 1;
select from record ;
select * from record r ;
delete from treatmenttype;
ALTER SEQUENCE treatmenttype_typekey_seq RESTART WITH 1;
select * from treatmenttype t ;
record 1
recordkey recordid outcome treatmenttypekey
1 REC001 Success 1
2 REC002 Failure 1
3 REC003 Success 2
4 REC004 Success 3
5 REC005 Failure 3
100 row(s) fetched - 0.000s, on 2024-11-04 at 17:41:52
Project - General
Name DataSource
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
Cancure-final-dwi-create.s postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Name Last Run (System time) Last Result Type
Custor 2024-10-10 20:15:00 Success Data imp...
EST en Writable Smart Insert 220 : 1 [24] Sel: 24 | 1
11/4/2024

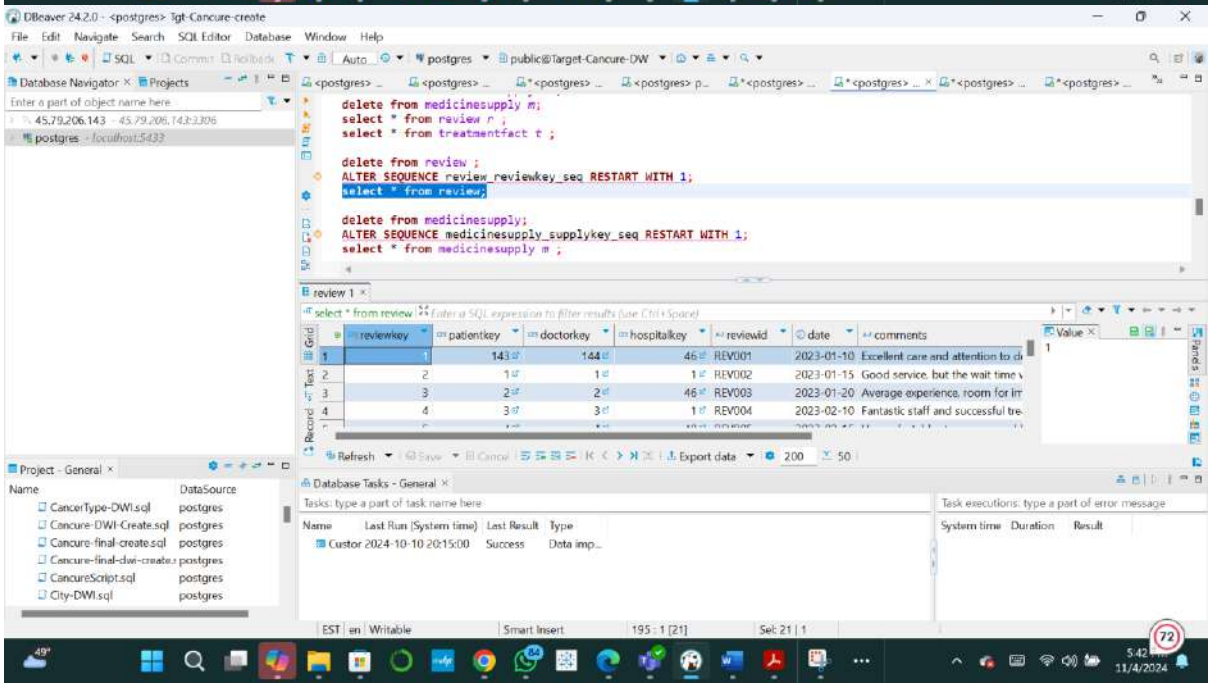

DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
delete from speciality;
ALTER SEQUENCE speciality_specialitykey_seq RESTART WITH 1;
select * from speciality;
delete from record;
ALTER SEQUENCE record_recordkey_seq RESTART WITH 1;
select from record ;
select * from record r ;
speciality 1
select * from speciality
specialitykey
speciality
1 Breast Oncology
2 Gynecologic Oncology
3 Pediatric Oncology
4 Hematologic Oncology
5 Neuro Oncology
10 row(s) fetched - 0.002s, on 2024-11-04 at 17:42:00
Database Tasks - General
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
select * from cancertype;
delete from doctor;
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
delete from speciality;
ALTER SEQUENCE speciality_specialitykey_seq RESTART WITH 1;
select * from speciality;
doctor 1
select * from doctor
doctorkey doctorid name email yoe specialitykey
1 DOC002 Dr. James Wilson jwilson@lacc.com 8 5
2 DOC003 Dr. Clara Oswald coswald@lacc.com 5 3
3 DOC004 Dr. Richard Webber rwebber@laoi.com 15 2
4 DOC005 Dr. Miranda Bailey mbailey@laoi.com 12 4
5 DOC006 Dr. Alex Karev akarev@laoi.com 7 1
144 row(s) fetched - 0.000s, on 2024-11-04 at 17:42:09
Database Tasks - General
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
delete from treatmentfact;
ALTER SEQUENCE treatmentfact_treatmentkey_seq RESTART WITH 1;
select * from treatmentfact t ;
delete from cancertype;
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
select * from cancertype;
delete from doctor;
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
cancertype 1
cancerkey cancertype stage
1 Breast Cancer Stage I
2 Breast Cancer Stage II
3 Breast Cancer Stage III
4 Breast Cancer Stage IV
5 Inflammatory Breast Cancer Stage I
80 row(s) fetched - 0.000s, on 2024-11-04 at 17:42:16
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
delete from treatmentfact;
ALTER SEQUENCE treatmentfact_treatmentkey_seq RESTART WITH 1;
select * from treatmentfact t ;
delete from cancertype;
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
select * from cancertype;
delete from doctor;
ALTER SEQUENCE doctor_doctorkey_seq RESTART WITH 1;
select * from doctor;
treatmentfact 1
treatmentkey hospitalkey patientkey patientid name phno gender diseasshi
1 46 143 PAT001 Sarah Williams 555-0101 Female No history
2 1 1 PAT002 Mary Smith 555-0102 Female Family histo
3 46 2 PAT003 Linda White 555-0103 Female No history
4 1 3 PAT004 Emily Clark 555-0201 Female Benign lump
100 row(s) fetched - 0.001s, on 2024-11-04 at 17:42:24
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
delete from medicinesupply;
ALTER SEQUENCE medicinesupply_supplykey_seq RESTART WITH 1;
select * from medicinesupply m ;
delete from treatmentfact;
ALTER SEQUENCE treatmentfact_treatmentkey_seq RESTART WITH 1;
select * from treatmentfact t ;
delete from cancertype;
ALTER SEQUENCE cancertype_cancerkey_seq RESTART WITH 1;
medicinesupply 1
supplykey hospitalkey medicinekey pharmacompanykey supplycost quantity sellingprice
1 22 7 24 900 100 1.
2 30 7 24 900 100 1.
3 23 2 24 450 120 1.
4 28 2 24 450 120 1.
68 row(s) fetched - 0.000s, on 2024-11-04 at 17:42:30
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...
EST | en | Writable | Smart Insert | 199 : 1 [32] | Sel: 32 | 1
11/4/2024


DBeaver 24.2.0 - <postgres> Tgt-Cancure-create
File Edit Navigate Search SQL Editor Database Window Help
delete from medicinesupply m;
select * from review r ;
select * from treatmentfact t ;
delete from review ;
ALTER SEQUENCE review_reviewkey_seq RESTART WITH 1;
select * from review;
delete from medicinesupply;
ALTER SEQUENCE medicinesupply_supplykey_seq RESTART WITH 1;
select * from medicinesupply m ;
review 1
reviewkey patientkey doctorkey hospitalkey reviewid date comments
1 143 144 46 REV001 2023-01-10 Excellent care and attention to d
2 1 1 1 REV002 2023-01-15 Good service, but the wait time
3 2 2 46 REV003 2023-01-20 Average experience, room for im
4 3 3 1 REV004 2023-02-10 Fantastic staff and successful tre
50
Project - General
CancerType-DWI.sql postgres
Cancure-DWI-Create.sql postgres
Cancure-final-create.sql postgres
CancureScript.sql postgres
City-DWI.sql postgres
Database Tasks - General
Custor 2024-10-10 20:15:00 Success Data imp...
EST | en | Writable | Smart Insert | 195 : 1 [21] | Sel: 21 | 1
11/4/2024

Additional Features:

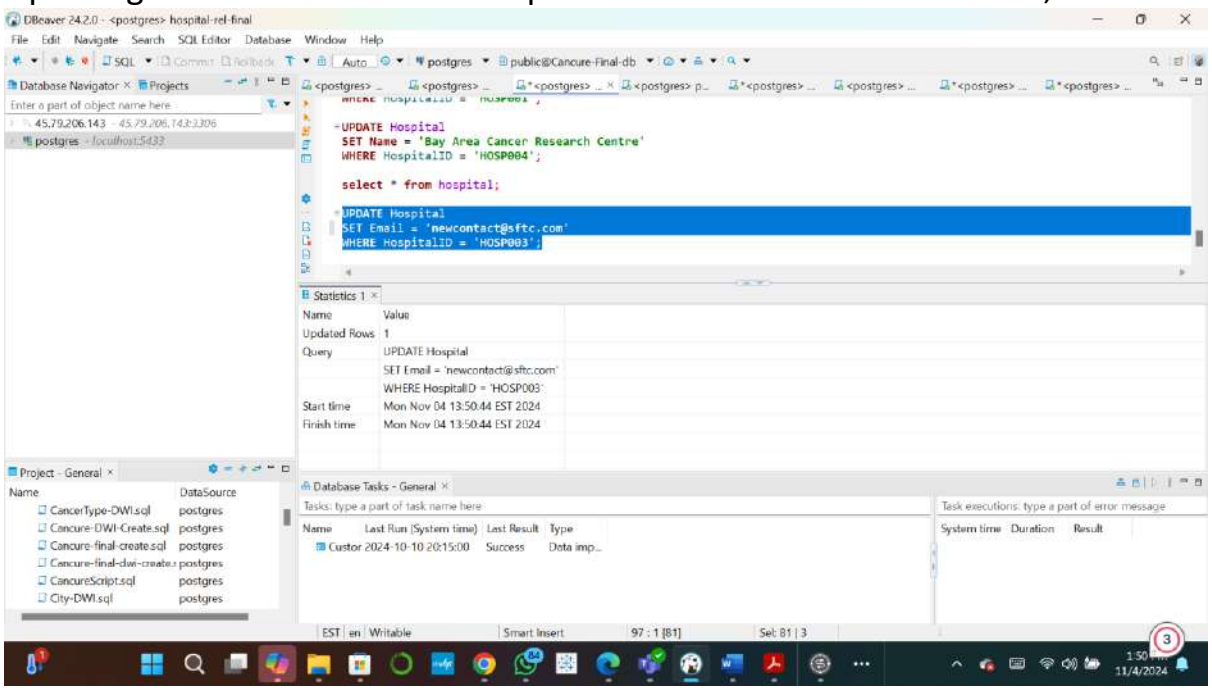
SCD: Type 1

The update flow is essentially the implementation of Type 1 SCDs because it updates the attributes of the dimensions as and when needed and removes the old data.

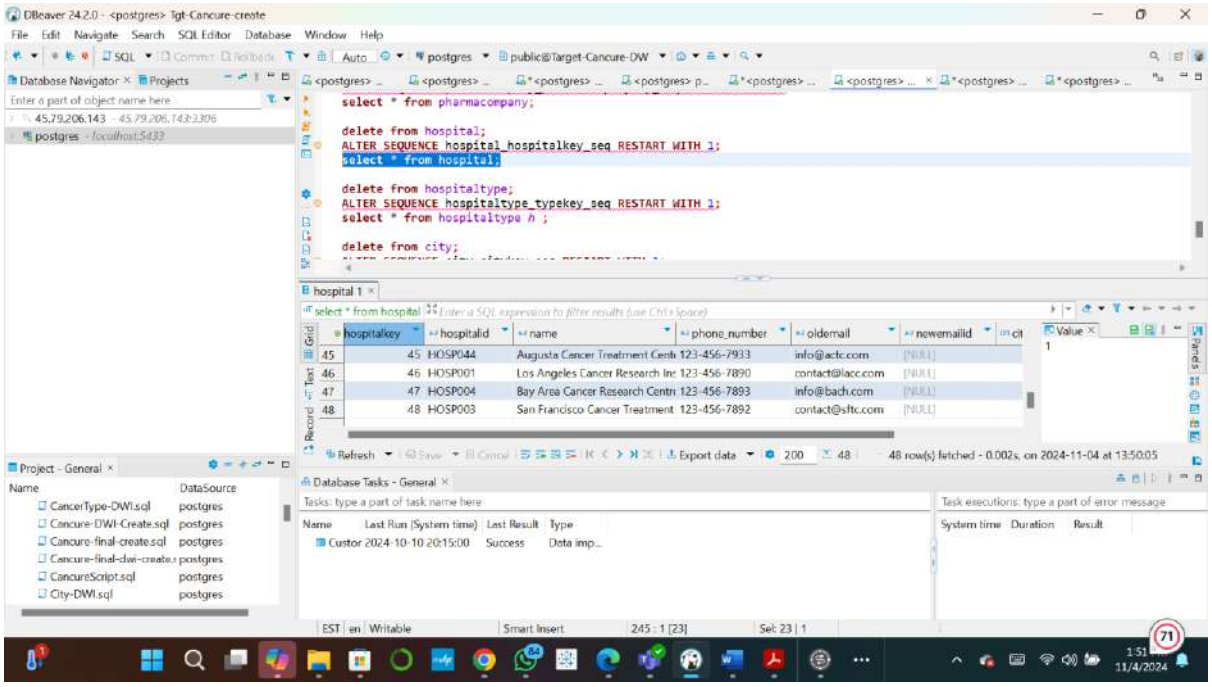
SCD: Type 3

Now, implementing Type 3 SCD, where only **the previous historical value is stored**, for example, in the hospital dimension,we store both the old and the new email of the dimension.

Updating the email address of Hospital 3 in relational database table,

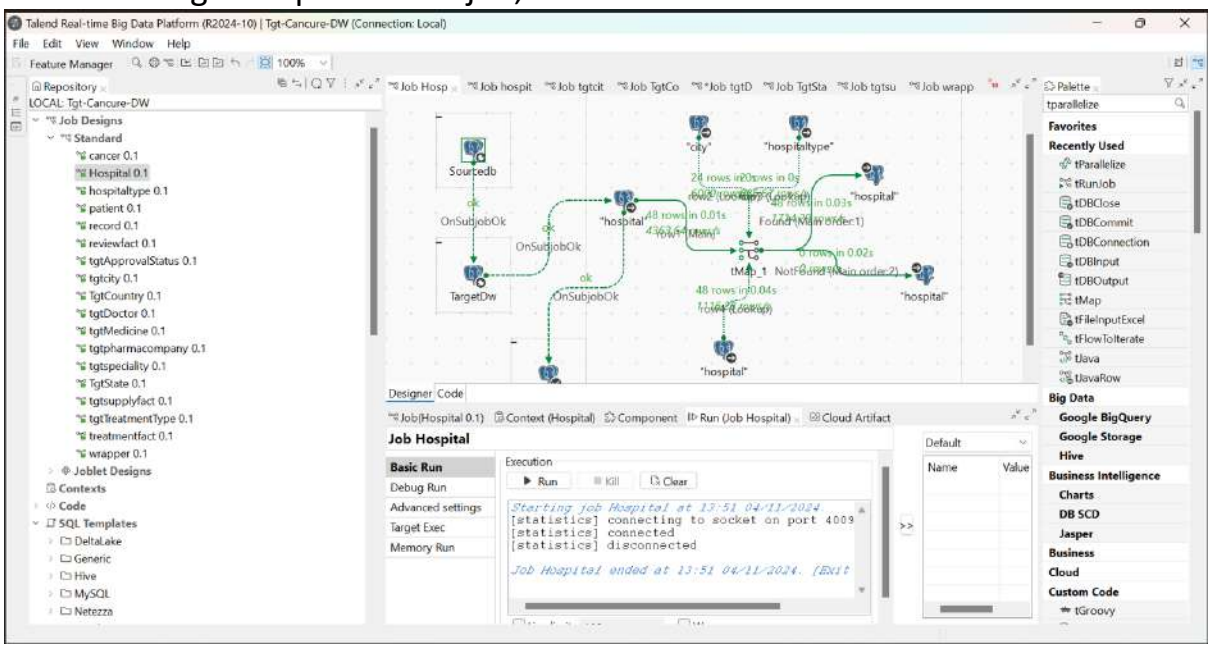


Before Updating Hospital data warehouse dimension table,

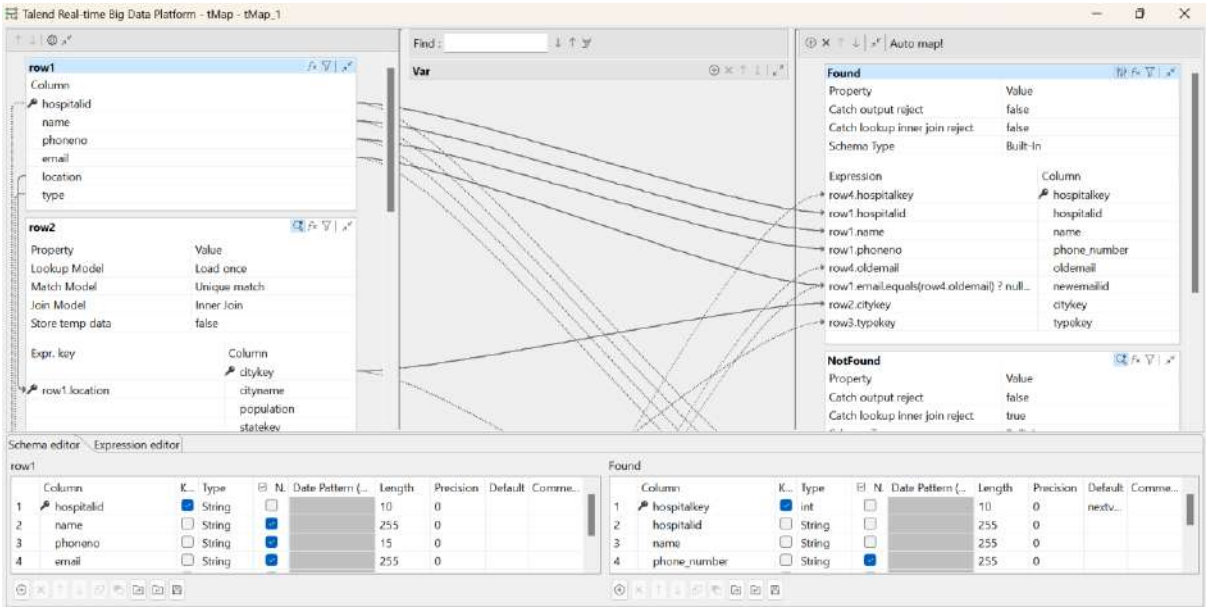


As we can see the newemailid column of the hospital 3 is null.

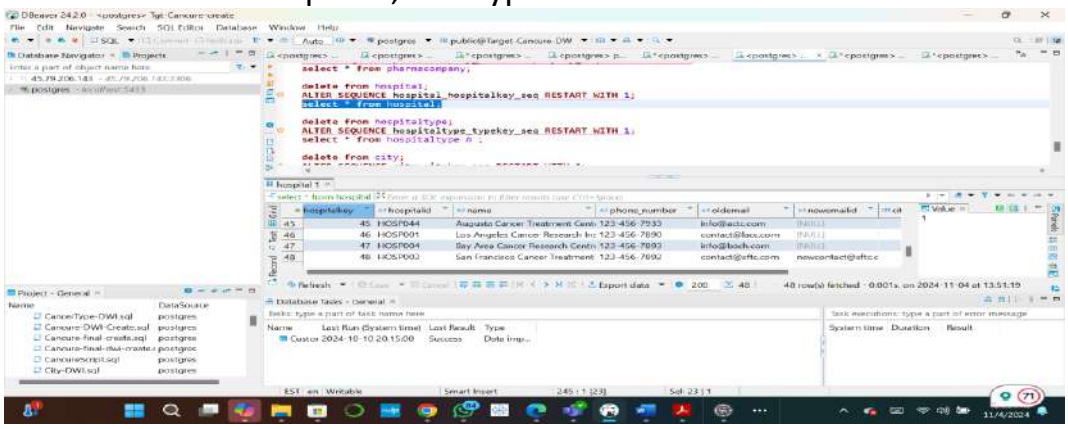
After running the update flow job,



T-map configuration:

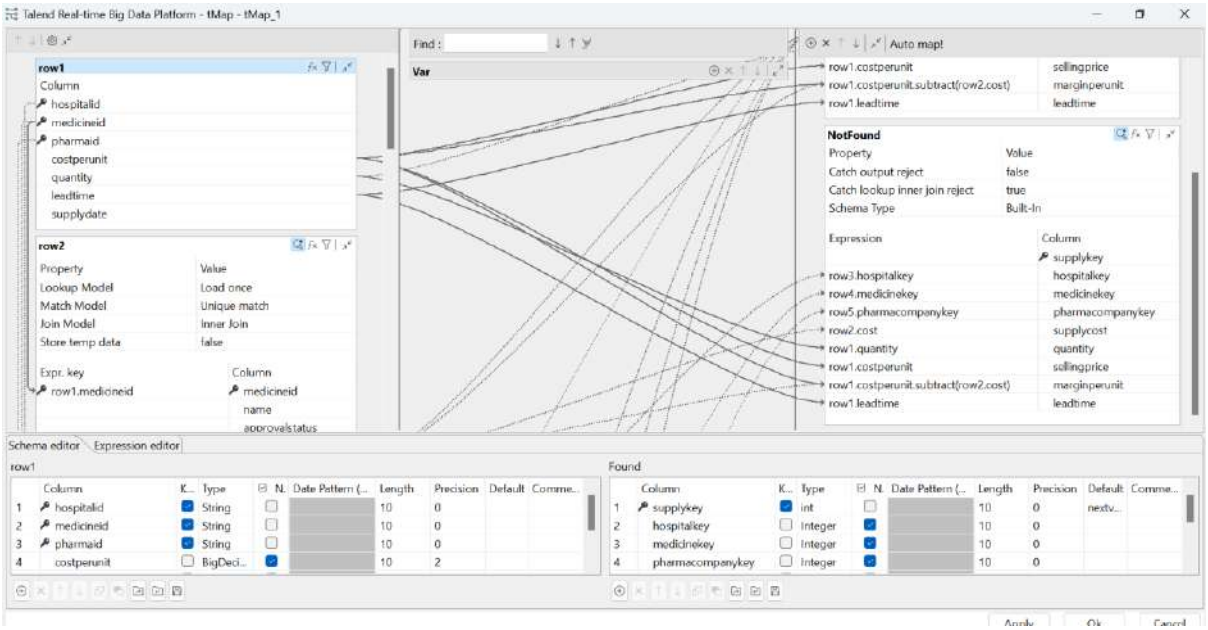


Updated data warehouse hospital dimension table **with both old and new email information** for hospital3,thus type 3 SCD is successful.

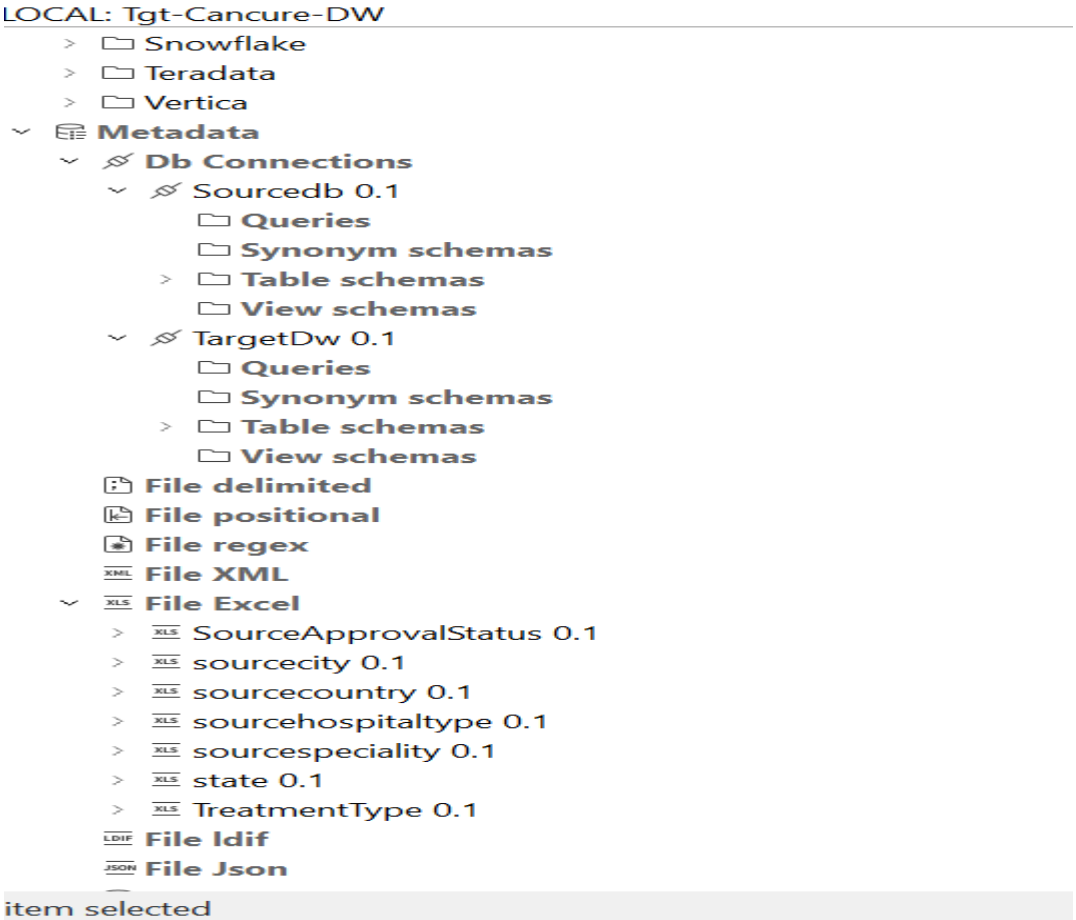


Calculated Measures:

We have implemented a calculated measure of our supplies fact table named MarginPerUnit which is the difference between the other two measures of cost price and selling price: The T-map shows the implementation of the calculated/derived measure Marginperunit



DataSources:



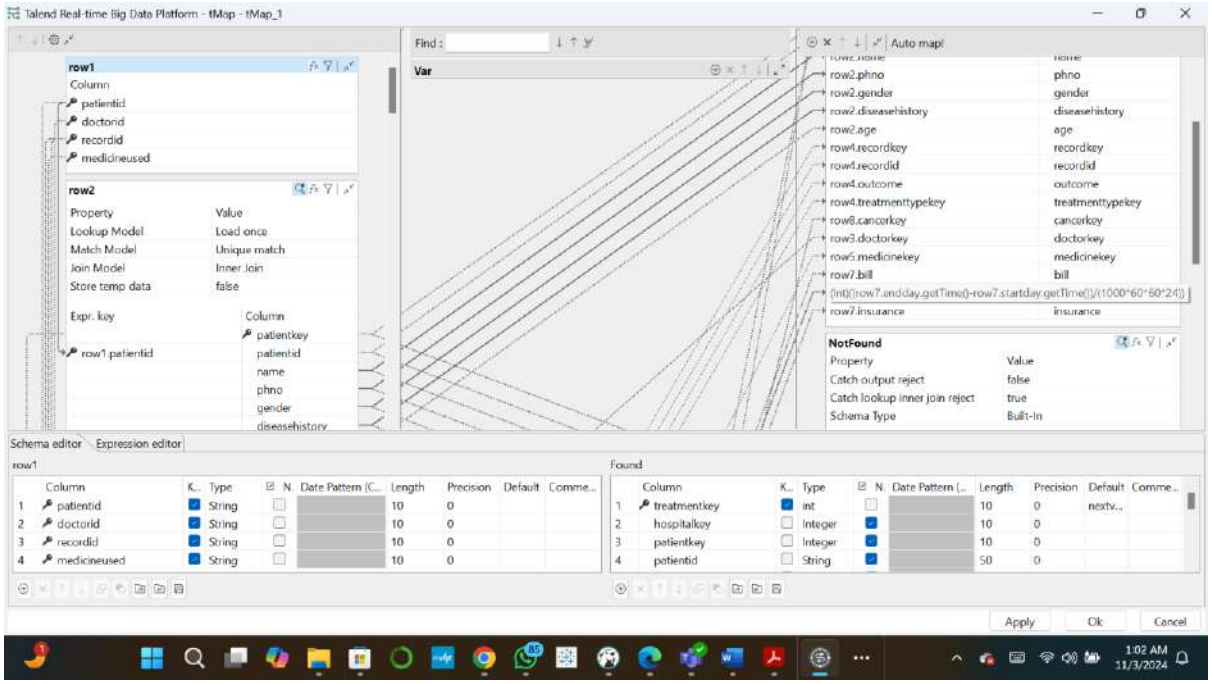
Excel and OLTP database are my two sources of data in this project to populate 17 tables of my datawarehouse of these 17 tables. 7 tables are populated from Excel sheets and the rest 10 tables are populated from the OLTP database. From the above screenshots we can see my sourcedb(OLTP Database) and all the 7 excel

sheets.

Transformations:

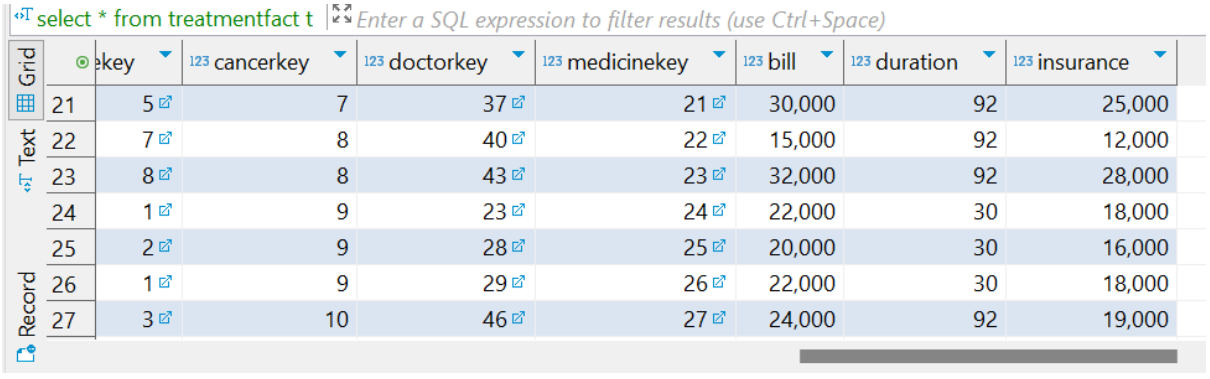
Some of the transformations done in the project are,

1. **Dates to int(duration):**



In the treatment fact table, the start and end days are converted to duration in int using the highlighted formula.

We can see the calculated duration in the below table

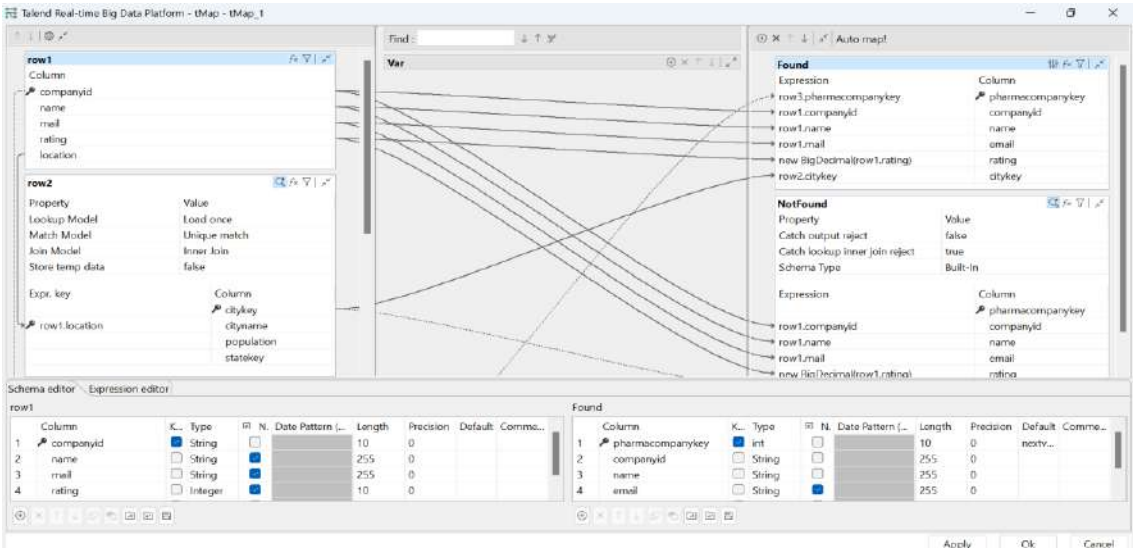


	ekey	cancerkey	doctorkey	medicinekey	bill	duration	insurance
21	5	7	37	21	30,000	92	25,000
22	7	8	40	22	15,000	92	12,000
23	8	8	43	23	32,000	92	28,000
24	1	9	23	24	22,000	30	18,000
25	2	9	28	25	20,000	30	16,000
26	1	9	29	26	22,000	30	18,000
27	3	10	46	27	24,000	92	19,000

2.Converting int to big decimal:

In the pharma company dimension table, **we converted the int data of the rating column in our source relational database to the big decimal datatype of rating in our datawarehouse database for a better precision**

The same can be seen in the T-map below,

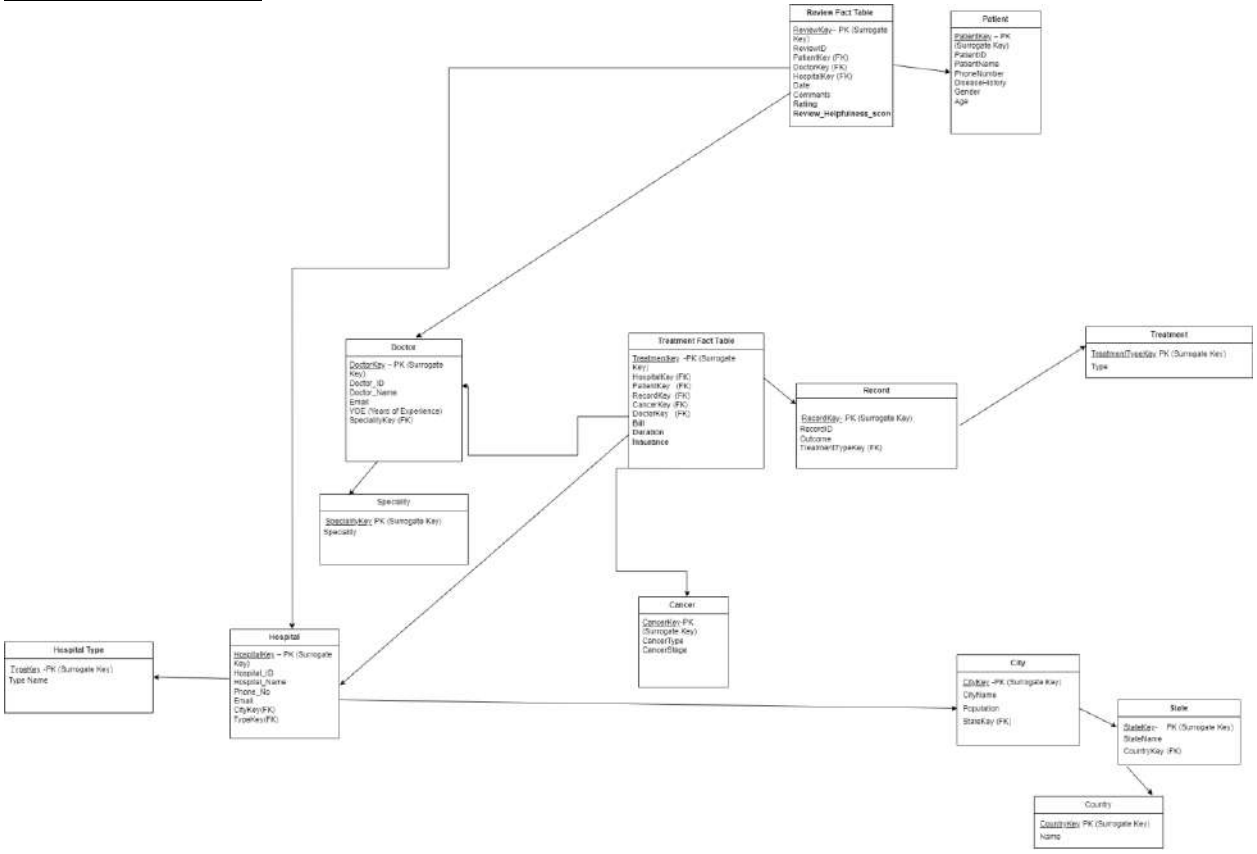


Project 2:

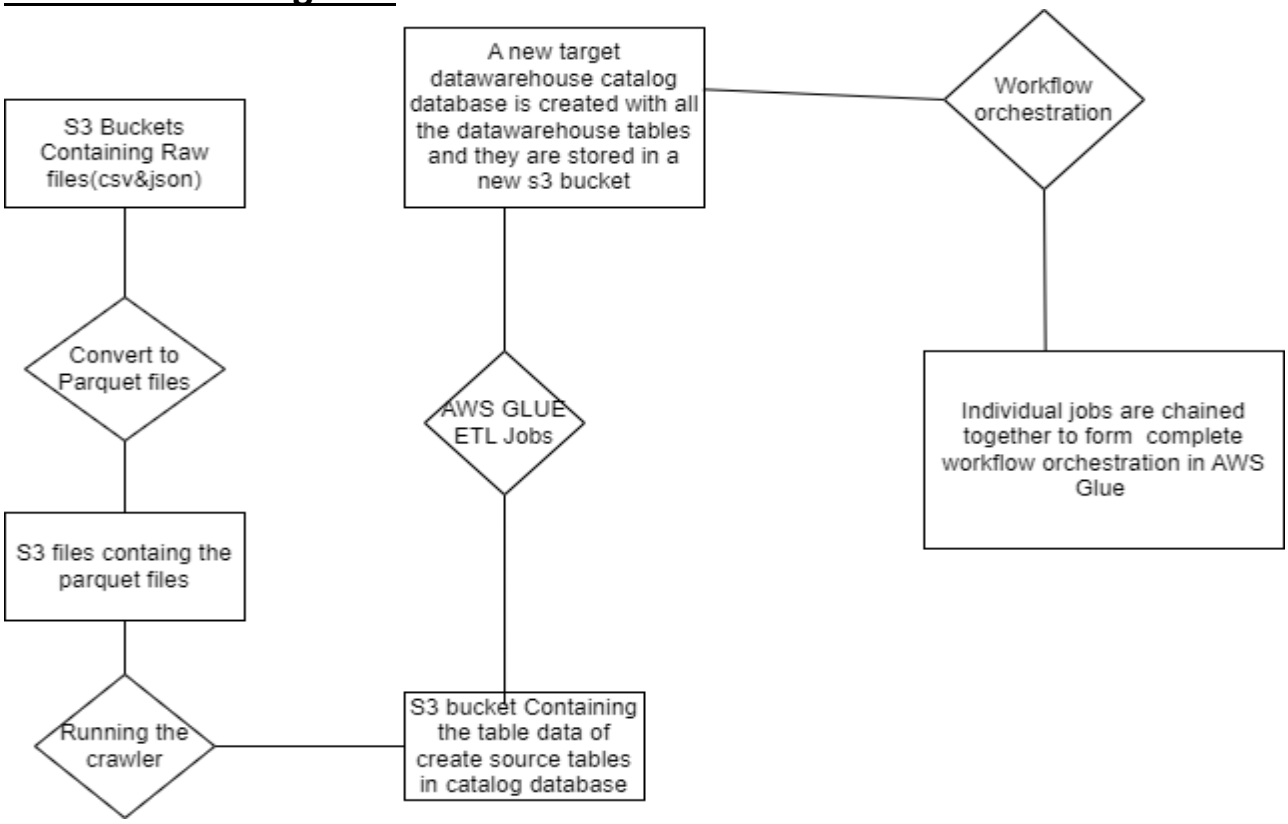
Cloud Implementation:

As mentioned in the previous milestone we have cut down on our on-prem project and excluded the supplies part hence the updated Logical Model:

Logical Model:



Architecture Diagram:



Implementation:

Data Sources:

First, we convert the data stored in our relational tables into CSV and then load it in S3 Bucket. In addition to that we have some files in JSON format, and all the necessary conversions are done using **Python.**

In [3]:
```
import pandas as pd


file_path = file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\COUNTRY.xlsx'

df = pd.read_excel(file_path)

json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\country.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\country.json
```

In [4]:
```
import pandas as pd

file_path = file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\State.xlsx'

df = pd.read_excel(file_path)

json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\state.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\state.json
```

In [5]:
```
import pandas as pd


file_path = file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\city.xlsx'

df = pd.read_excel(file_path)

json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\city.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\city.json
```

In [6]:
```
import pandas as pd



df = pd.read_excel(file_path)

json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\approvalstatus.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\approvalstatus.json
```

In [7]:
```
import pandas as pd

file_path = file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\HospitalType.xlsx'

df = pd.read_excel(file_path)

json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\hospitaltype.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\hospitaltype.json
```

In [8]:
```
import pandas as pd


file_path = file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\Speciality.xlsx'

df = pd.read_excel(file_path)


json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\speciality.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\speciality.json
```

In [10]:
```
import pandas as pd

file_path = file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\TreatmentType.xlsx'

df = pd.read_excel(file_path)

json_path = r'C:\Users\aravi\OneDrive\Documents\JSON_data_source\treatmenttype.json'
df.to_json(json_path, orient='records', lines=True)

print(f'File converted and saved as {json_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\JSON_data_source\treatmenttype.json
```

In [11]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\COUNTRY.xlsx'

df = pd.read_excel(file_path)


csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\country.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\country.csv
```

In [12]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\State.xlsx'


df = pd.read_excel(file_path)

csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\state.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\state.csv
```

In [13]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\city.xlsx'

df = pd.read_excel(file_path)

csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\city.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\city.csv
```

In [14]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\ApprovalStatus.xlsx'

df = pd.read_excel(file_path)

csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\approvalstatus.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\approvalstatus.csv
```

In [15]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\HospitalType.xlsx'

df = pd.read_excel(file_path)

csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\hospitaltype.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\hospitaltype.csv
```

In [16]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\Speciality.xlsx'

df = pd.read_excel(file_path)

csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\speciality.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

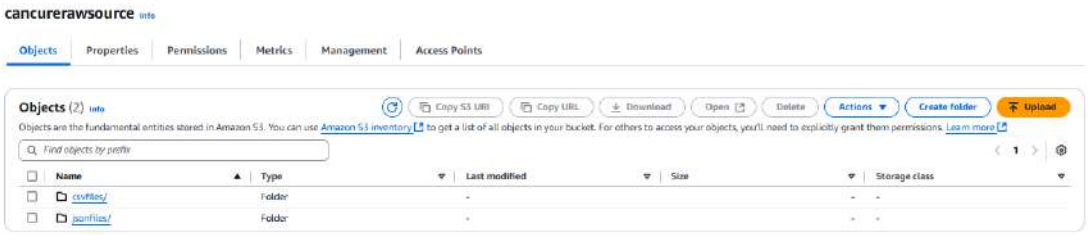
```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\speciality.csv
```

In [17]:
```
import pandas as pd

file_path = r'C:\Users\aravi\OneDrive\Desktop\DataSources\TreatmentType.xlsx'

df = pd.read_excel(file_path)

csv_path = r'C:\Users\aravi\OneDrive\Documents\CSV_data_source\treatmenttype.csv'

df.to_csv(csv_path, index=False)

print(f'File converted and saved as {csv_path}')
```

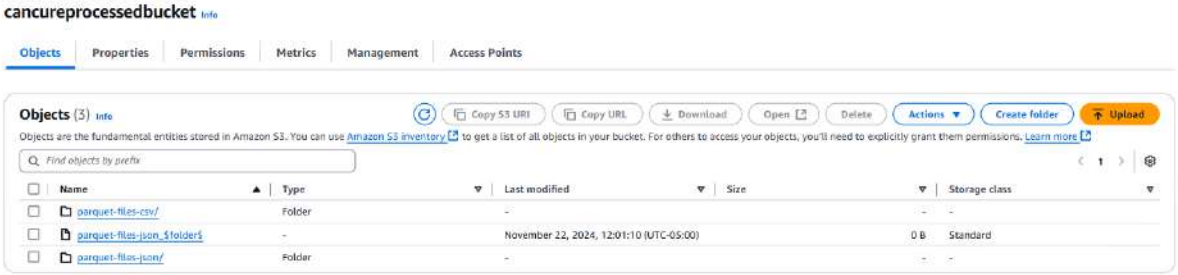
```
File converted and saved as C:\Users\aravi\OneDrive\Documents\CSV_data_source\treatmenttype.csv
```

In [ ]:

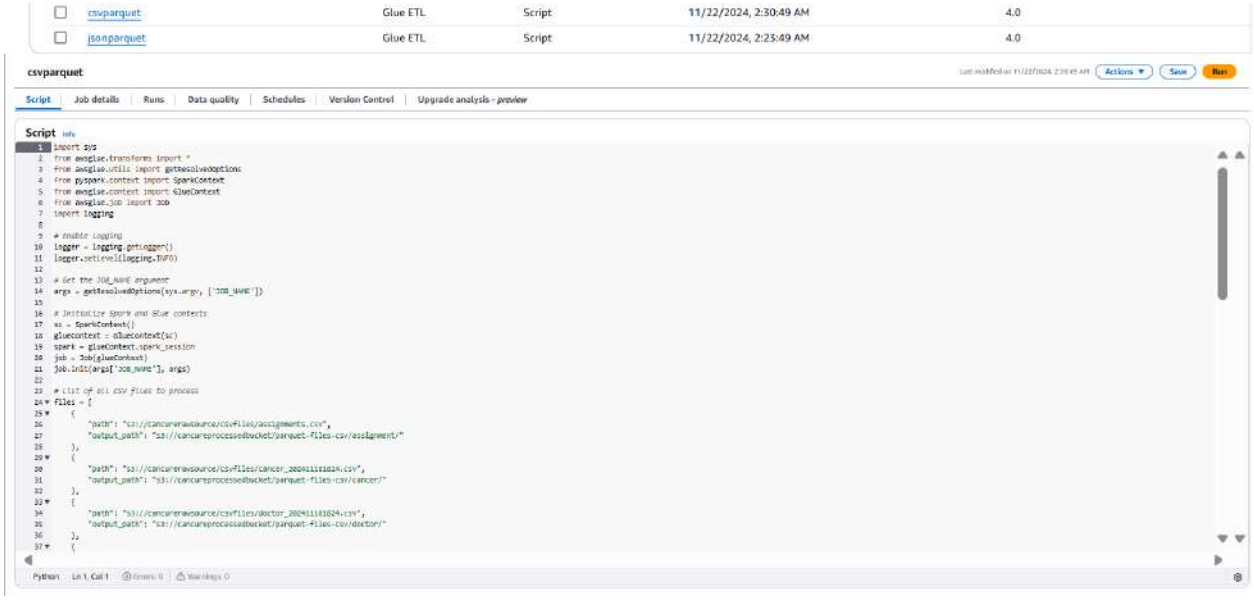
Now these CSV and JSON files are loaded into an s3 bucket named [cancurerawsource](). These files act as our source.

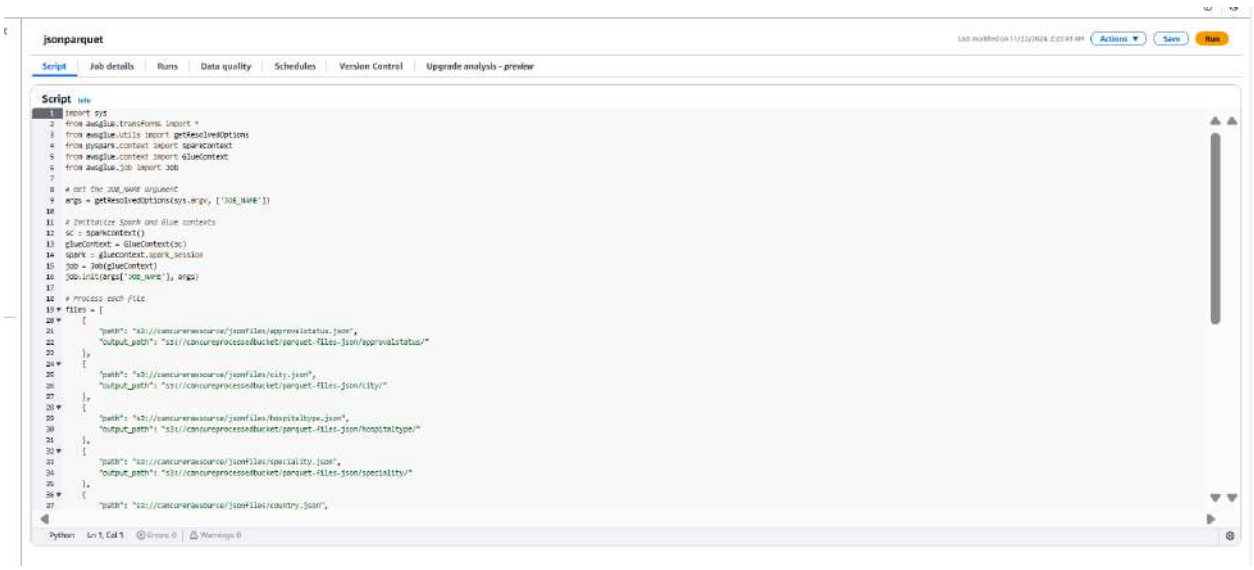


Now **these raw CSV files and JSON files are converted to parquet format using AWS Glue jobs** and is stored in another bucket named cancureprocessedbucket.



The AWS Glue jobs used to convert the raw files are

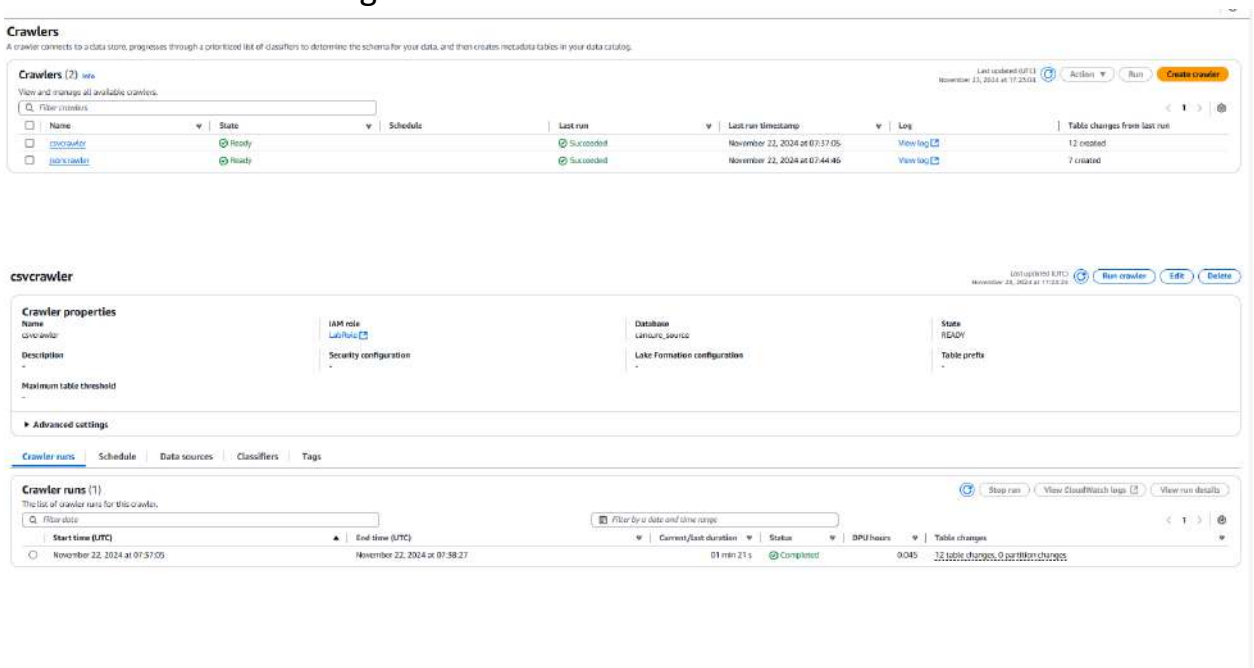


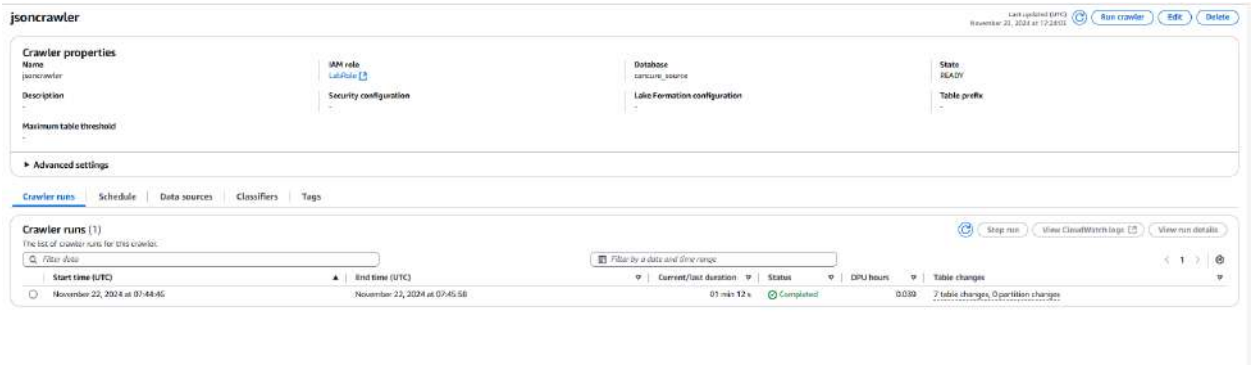


So these jobs are our first transformation and it is stored in another bucket as mentioned.

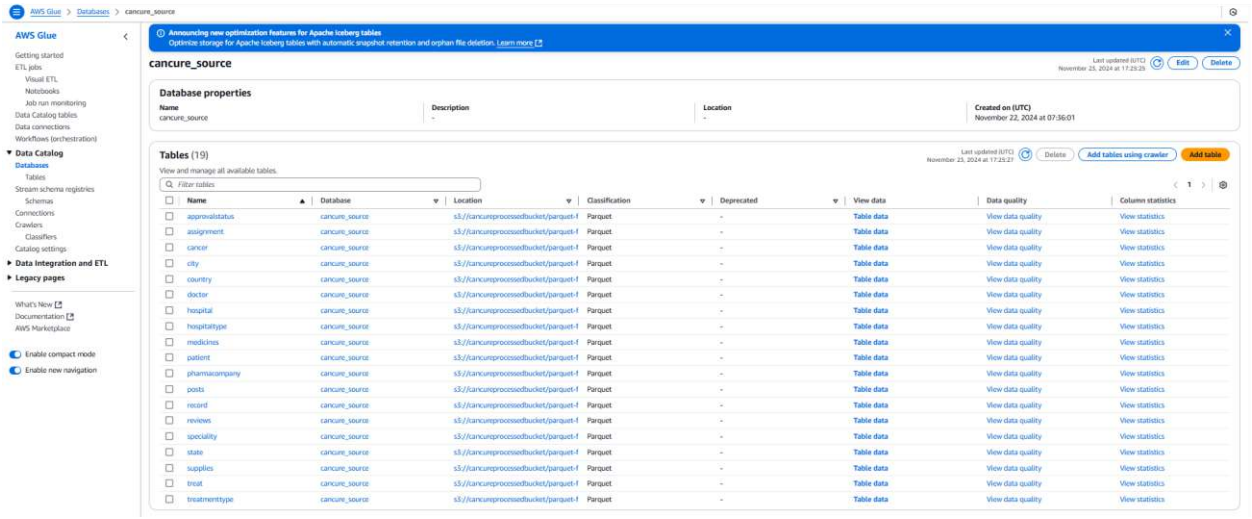
Crawlers:

Now we build crawlers to crawler over these two parquet folders to create our source relational catalog database.



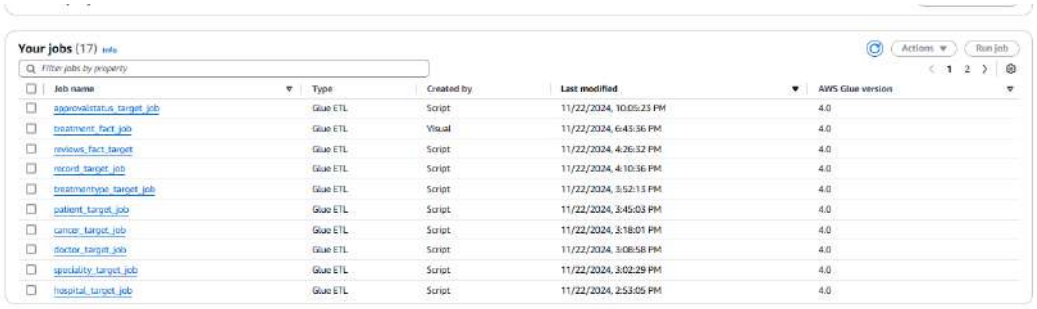


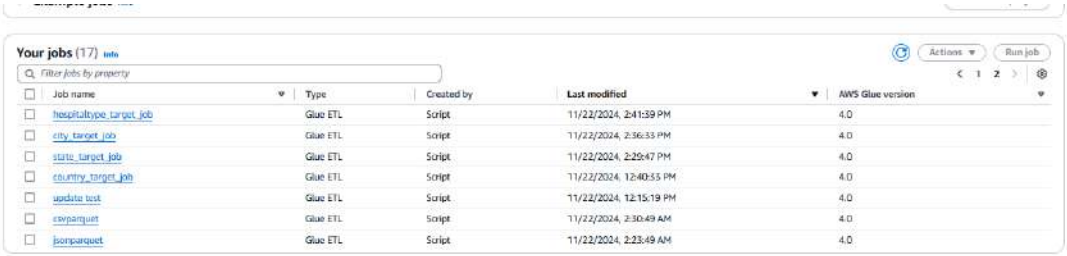
And once both the crawlers are run.
We have a source catalog database.
There were total of 12 CSV files and 7 JSON files,
Which are converted to corresponding tables in the source catalog database.



Now the data source is loaded and source catalog database is created, we now create ETL AWS Glue jobs to transform and load the data in the target data warehouse database,
There are a total 13 jobs including for all dimensions and fact tables.
We have followed the AWS Glue script editor to construct these jobs,

We have jobs for all the tables,





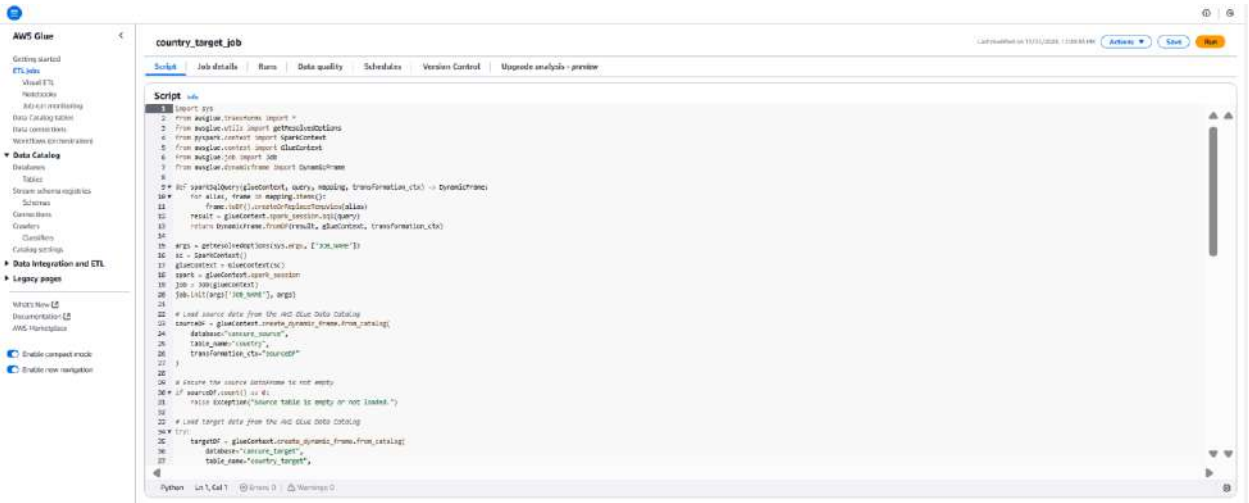
Pipeline Tools:

So in order to build and check this pipeline we have used various tools like S3 Bucket, AWS Glue jobs, Workflow Orchestration, Crawlers and Athena and AWS Catalog databases.

Now seeing the individual jobs,

Individual Jobs:

Country Job:



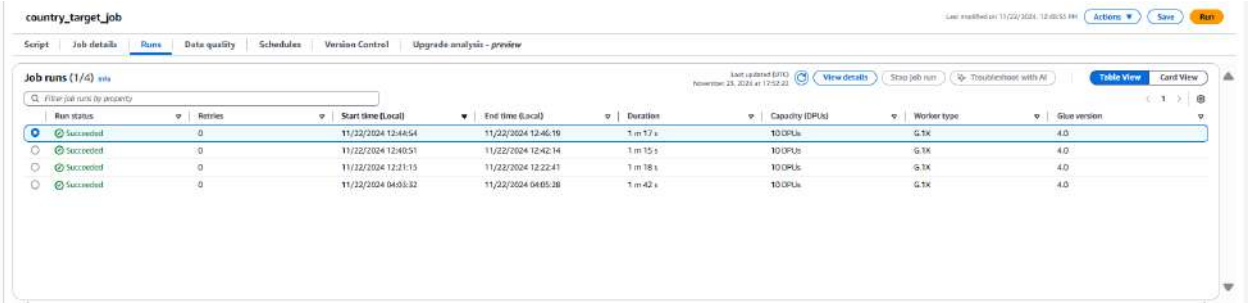
```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY Name ASC) AS CountryKey,
            Name
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.Name) = TRIM(tgt.Name) -- Match based on Name column
        WHERE src.Name IS NOT NULL AND TRIM(src.Name) != ''
    """
else:

    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY Name ASC) AS CountryKey,
            Name
        FROM source_table
        WHERE Name IS NOT NULL AND TRIM(Name) != ''
    """
```

This part of the code matches and sees if any duplicate data is there in the incoming data so that duplicate data is not inserted every time the job is run and we also add a surrogate key.

And it is stored in another target data warehouse S3 bucket named **cancuretargetdw.**

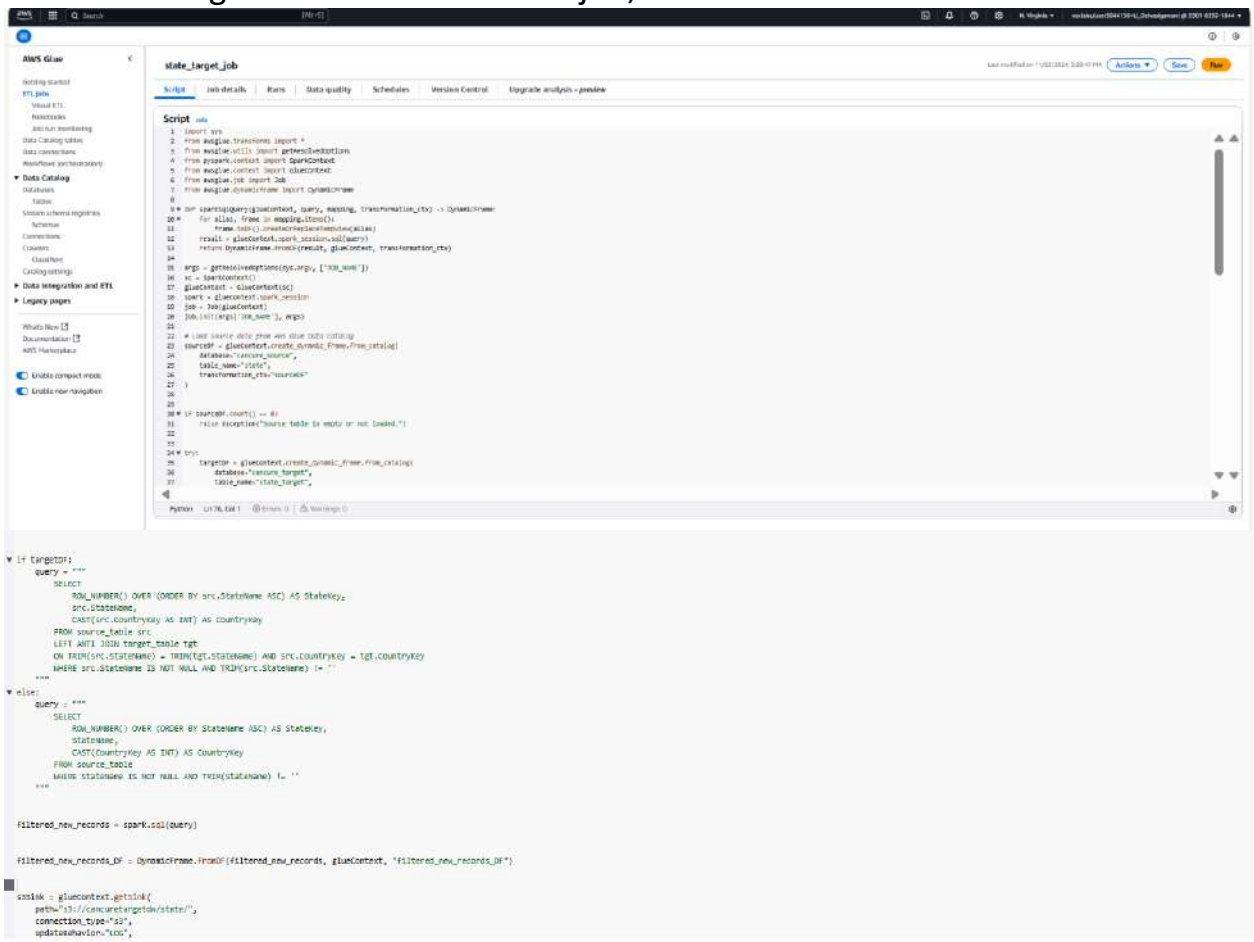
```
iltered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")


3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/country/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
)
3Sink.setCatalogInfo(catalogDatabase="cancure_target", catalogTableName="country_target")
3Sink.setFormat("glueparquet", compression="snappy")
3Sink.writeFrame(filtered_new_records_DF)


ob.commit()
```

And the job is run,



State Job:

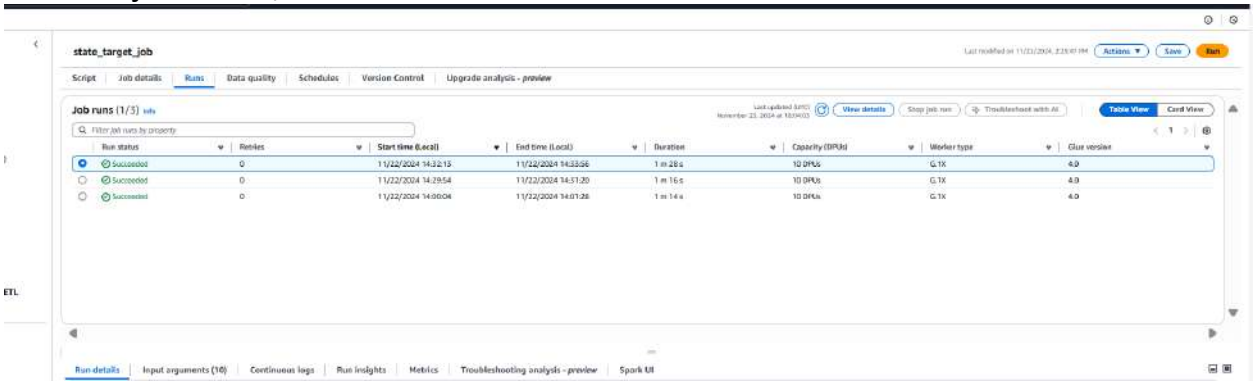
Now are doing the same for the state job,



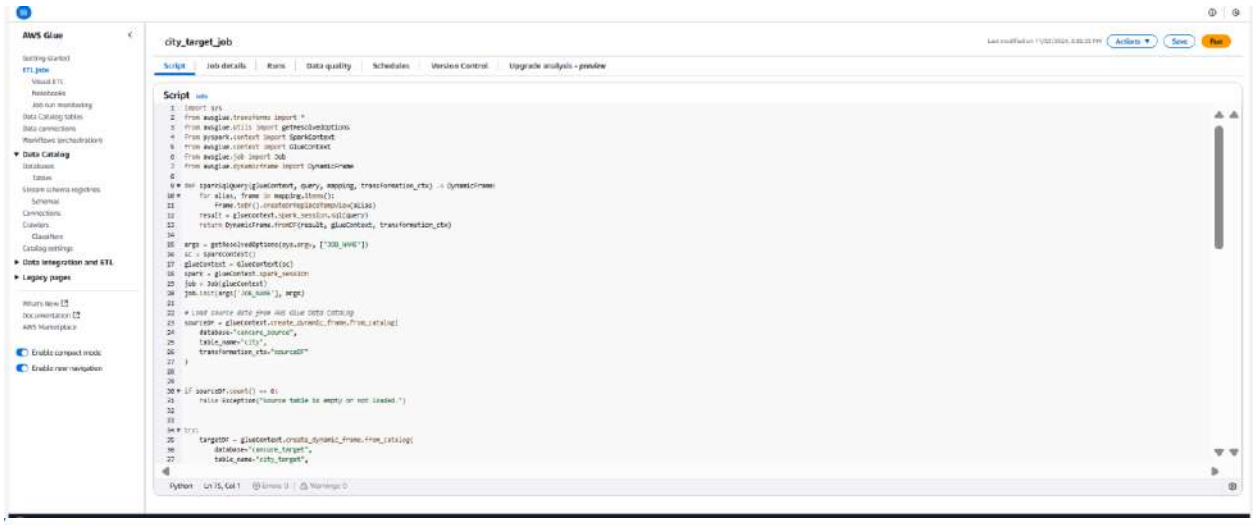
```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.StateName ASC) AS StateKey,
            src.StateName,
            CAST(src.CountryKey AS INT) AS CountryKey
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.StateName) = TRIM(tgt.StateName) AND src.CountryKey = tgt.CountryKey
        WHERE src.StateName IS NOT NULL AND TRIM(src.StateName) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY StateName ASC) AS StateKey,
            StateName,
            CAST(CountryKey AS INT) AS CountryKey
        FROM source_table
        WHERE StateName IS NOT NULL AND TRIM(StateName) != ''
    """


filtered_new_records = spark.sql(query)


filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

ssink = glueContext.getSink(
    path="s3://cancuretargetdw/state/",
    connection_type="s3",
    updateBehavior="LOG",
```

We are adding the surrogate keys and transferring it to the state datawarehouse table.

And the job is run,

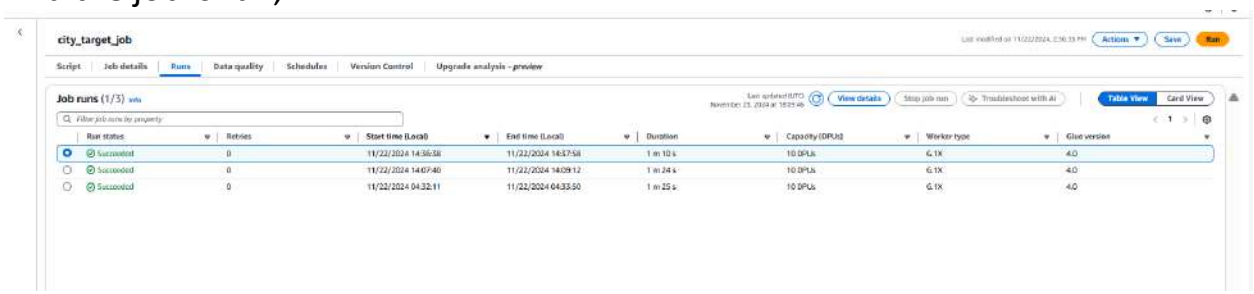


City:

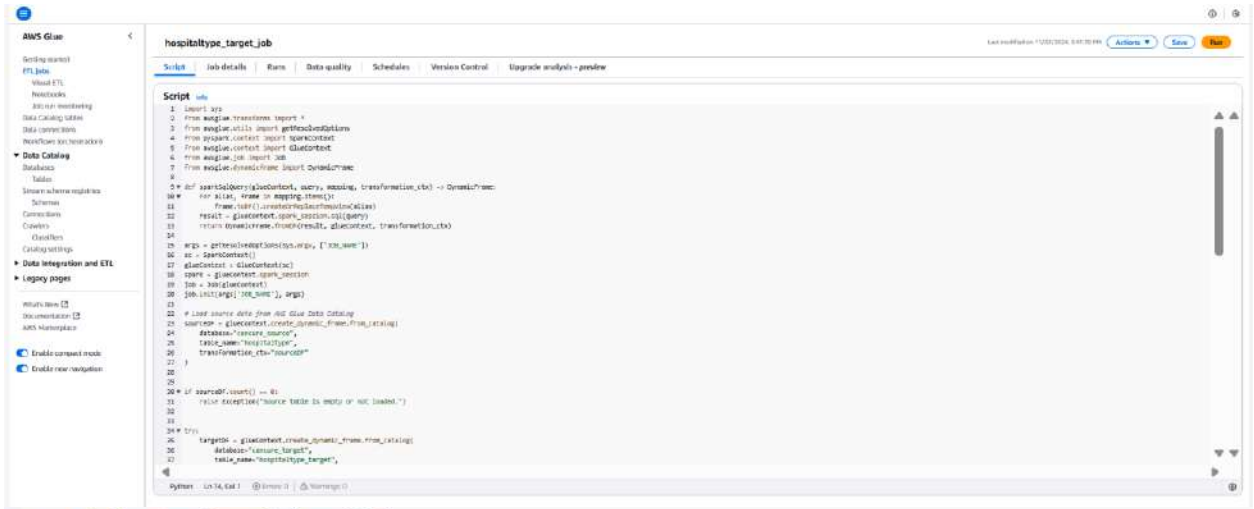



```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.CityName ASC) AS CityKey,
            src.CityName,
            CAST(src.Population AS INT) AS Population,
            CAST(src.StateKey AS INT) AS StateKey
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.CityName) = TRIM(tgt.CityName) AND src.StateKey = tgt.StateKey
        WHERE src.CityName IS NOT NULL AND TRIM(src.CityName) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY CityName ASC) AS CityKey,
            CityName,
            CAST(Population AS INT) AS Population,
            CAST(StateKey AS INT) AS StateKey
        FROM source_table
        WHERE CityName IS NOT NULL AND TRIM(CityName) != ''
    """

filtered_new_records = spark.sql(query)

filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

# Write the new records to the target table in S3 and update the Glue Catalog
s3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/city/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
```

The same is done for the city datawarehouse table.
And the job is run,

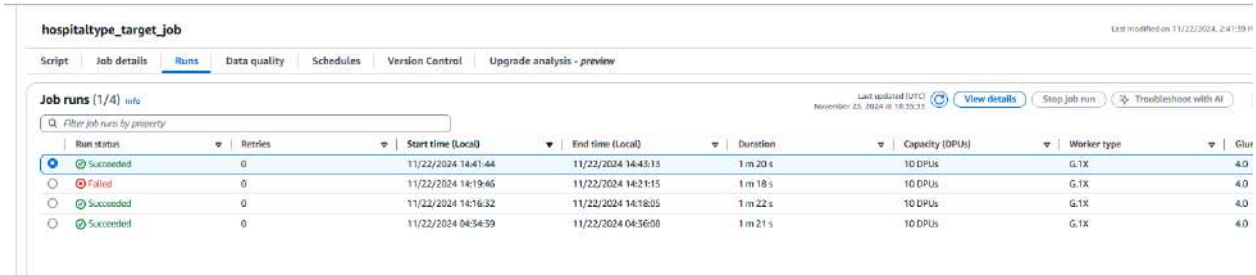


HospitalTypeJob:

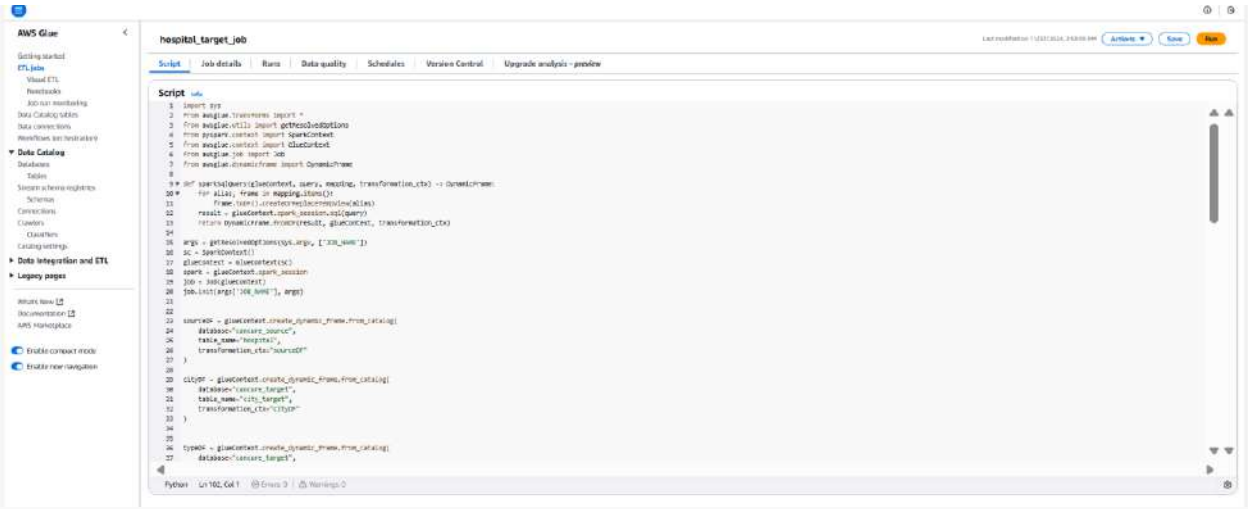



```
sourceDF.toDF().createOrReplaceTempView("source_table")


if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.TypeName ASC) AS TypeKey,
            src.TypeName
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.TypeName) = TRIM(tgt.TypeName)
        WHERE src.TypeName IS NOT NULL AND TRIM(src.TypeName) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY TypeName ASC) AS TypeKey,
            TypeName
        FROM source_table
        WHERE TypeName IS NOT NULL AND TRIM(TypeName) != ''
    """


filtered_new_records = spark.sql(query)


filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")


s3sink = glueContext.getSink(
    path="s3://cancuretargetdw/hospitaltype/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
```

For hospitaltype datawarehouse table also the surrogate key is added and loaded. Now the job is run ,



Hospital Job:

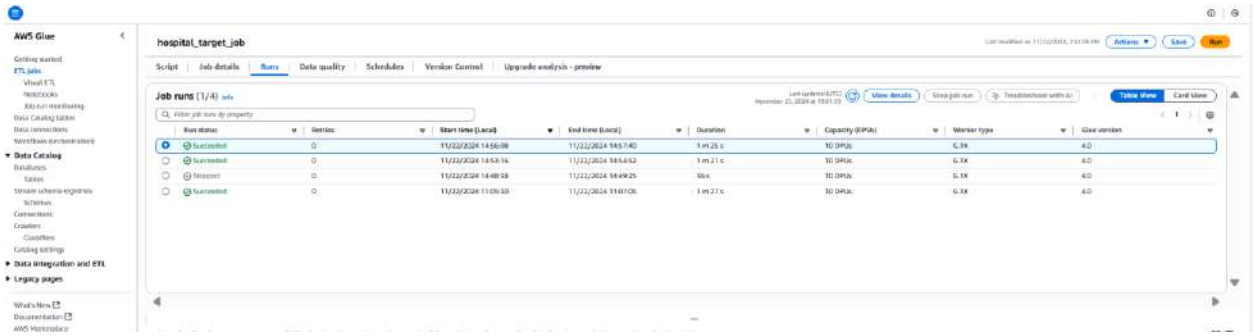



```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY h.hospitalid ASC) AS HospitalKey,  -- Generate sequential HospitalKey
            h.hospitalid,
            h.name,
            h.phoneno AS Phone_Number,
            h.email AS EmailID,
            c.citykey AS CityKey,  -- Replace location with CityKey from city_table
            t.typekey AS TypeKey   -- Replace type with TypeKey from type_table
        FROM hospital_table h
        LEFT ANTI JOIN target_table tgt
            ON TRIM(h.hospitalid) = TRIM(tgt.hospitalid)  -- Avoid duplicates
        INNER JOIN city_table c
            ON TRIM(h.location) = TRIM(c.cityname)  -- Match location with cityname
        INNER JOIN type_table t
            ON TRIM(h.type) = TRIM(t.typename)  -- Match type with typename
        WHERE h.hospitalid IS NOT NULL AND TRIM(h.hospitalid) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY h.hospitalid ASC) AS HospitalKey,  -- Generate sequential HospitalKey
            h.hospitalid,
            h.name,
            h.phoneno AS Phone_Number,
            h.email AS EmailID,
            c.citykey AS CityKey,  -- Replace location with CityKey from city_table
            t.typekey AS TypeKey   -- Replace type with TypeKey from type_table
        FROM hospital_table h
        INNER JOIN city_table c
            ON TRIM(h.location) = TRIM(c.cityname)  -- Match location with cityname
        INNER JOIN type_table t
            ON TRIM(h.type) = TRIM(t.typename)  -- Match type with typename
        WHERE h.hospitalid IS NOT NULL AND TRIM(h.hospitalid) != ''
    """
```

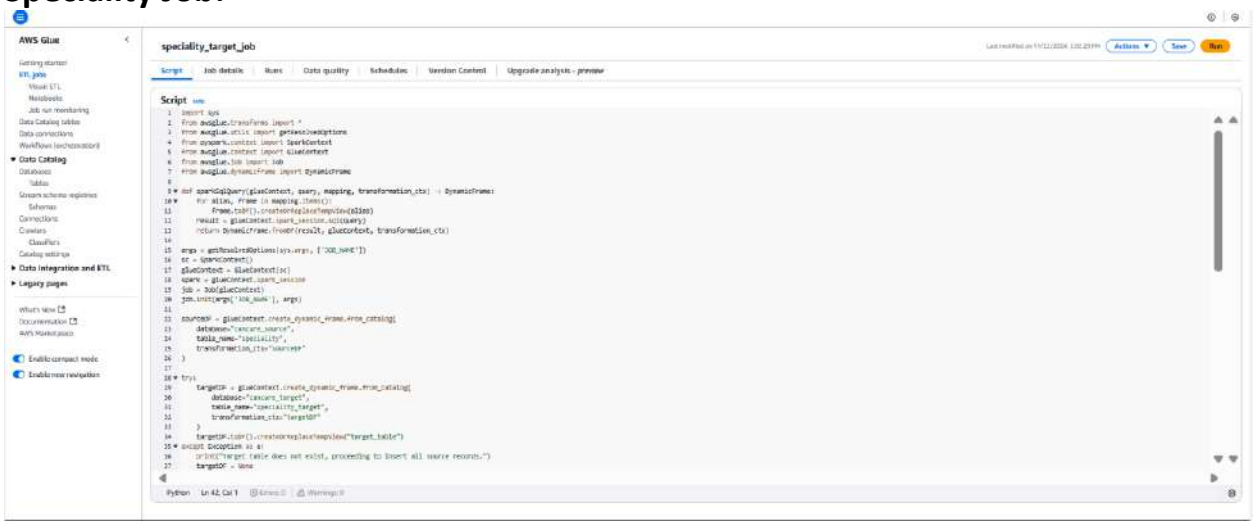
Now for the hospital table we join the citytable and the hospitaltype table to map the corresponding citykey and hospitaltypekey in the hospital datawarehouse table.

```
    """

filtered_new_records = spark.sql(query)


resultDynamicFrame = DynamicFrame.fromDF(filtered_new_records, glueContext, "resultDynamicFrame")

s3sink = glueContext.getSink(
    path="s3://cancuretarget4a/hospital/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
)
s3sink.setCatalogInfo(catalogDatabase="cancure_target", catalogTableName="hospital_target")
s3Sink.setFormat("glueparquet", compression="snappy")
s3Sink.writeFrame(resultDynamicFrame)

# Commit the job
job.commit()
```

And the job is run,



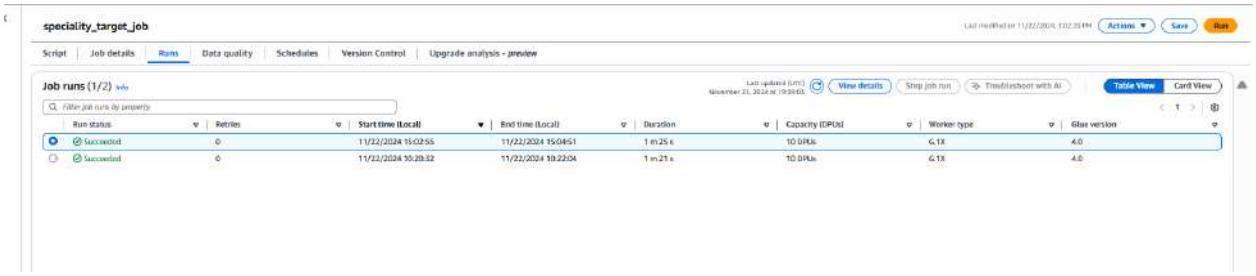
Speciality Job:



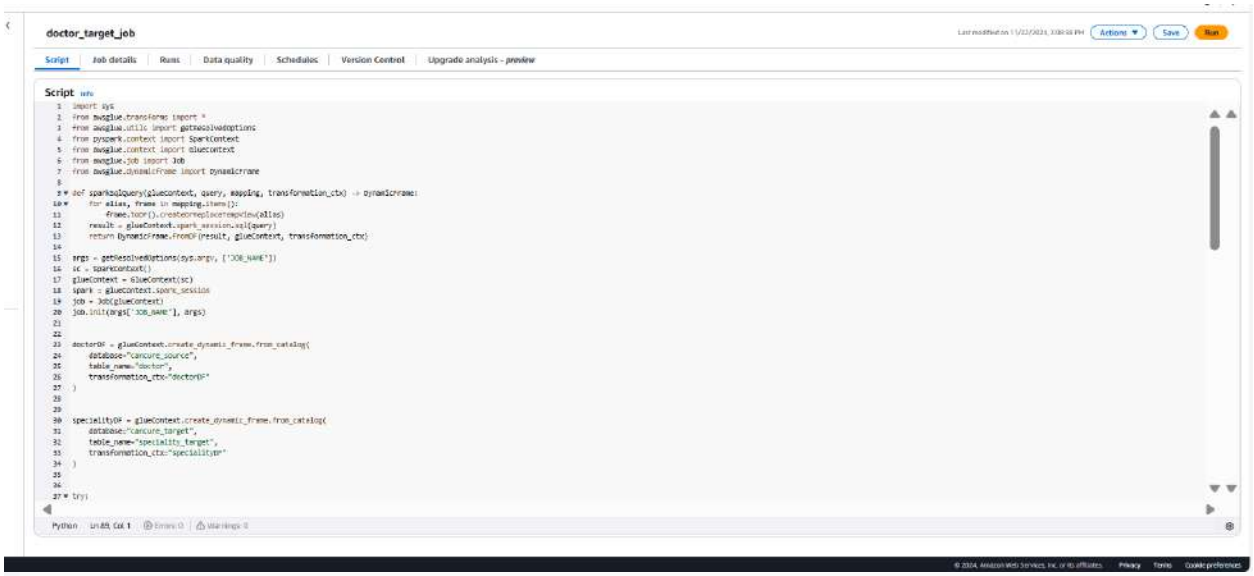
```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.Speciality ASC) AS SpecialityKey,  -- Generate unique keys starting from 1
            src.Speciality
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.Speciality) = TRIM(tgt.Speciality) -- Exclude already present records
        WHERE src.Speciality IS NOT NULL AND TRIM(src.Speciality) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY Speciality ASC) AS SpecialityKey,  -- Generate unique keys starting from 1
            Speciality
        FROM source_table
        WHERE Speciality IS NOT NULL AND TRIM(Speciality) != ''
    """

# Execute the query
filtered_new_records = spark.sql(query)

# Convert the filtered records back to a Glue DynamicFrame
filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

# Write the new records to the target table in S3 and update the Glue Catalog
s3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/speciality/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
)
```

hon Ln 42, Col 1 Errors: 0 Warnings: 0

As discussed in all the other tables above we add surrogate key for the speciality table and load it in the speciality datawarehouse table.
Now the job is run.



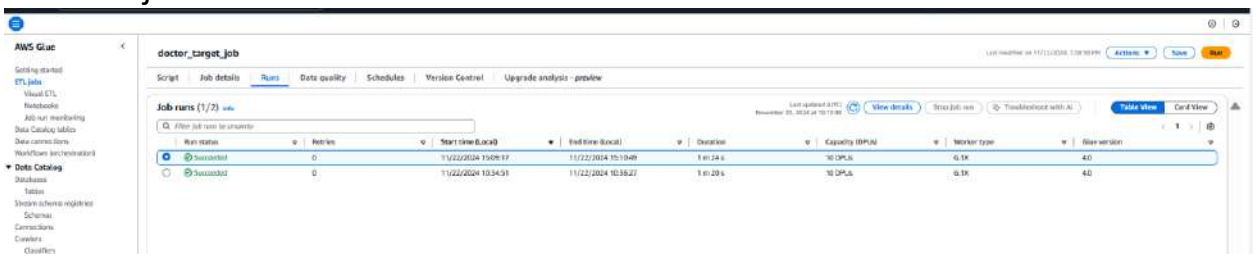
Doctor Job:



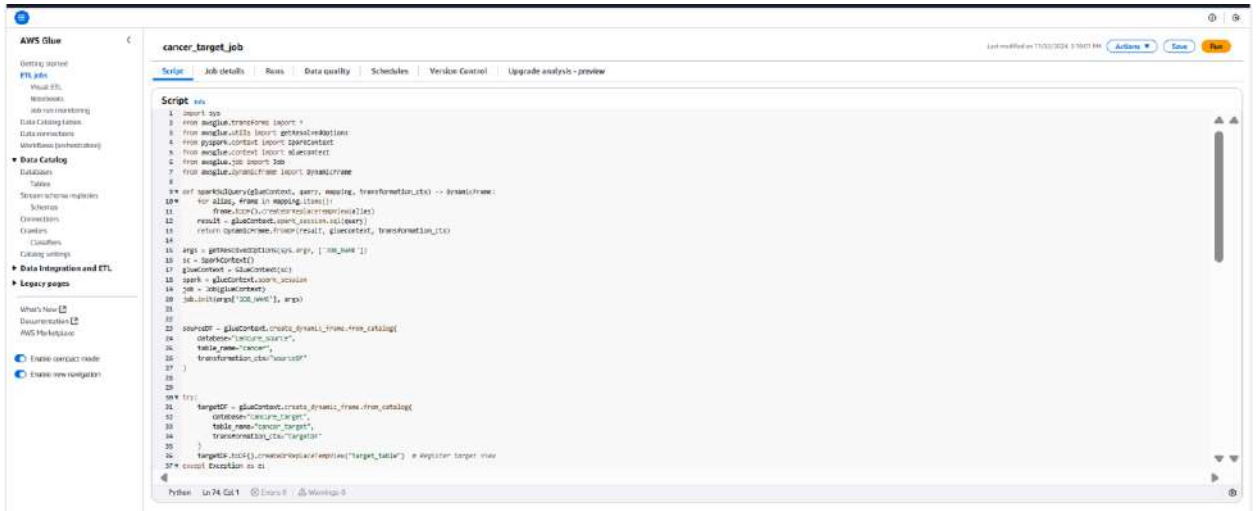

```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY d.doctorid ASC) AS DoctorKey,  -- Generate sequential DoctorKey
            d.doctorid,                                                -- Retain Doctor_ID
            d.name AS Doctor_Name,                                     -- Map Name to Doctor_Name
            d.email AS Email,                                          -- Retain Email
            CAST(d.yoe AS INT) AS YOE,                                 -- Cast YOE to INT
            s.specialitykey                                            -- Map speciality to SpecialityKey
        FROM doctor_table d
        LEFT ANTI JOIN target_table t
            ON TRIM(d.doctorid) = TRIM(t.doctorid)  -- Avoid duplicates
        INNER JOIN speciality_table s
            ON TRIM(d.speciality) = TRIM(s.speciality)  -- Match speciality with speciality_key
        WHERE d.doctorid IS NOT NULL AND TRIM(d.doctorid) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY d.doctorid ASC) AS DoctorKey,  -- Generate sequential DoctorKey
            d.doctorid,                                                -- Retain Doctor_ID
            d.name AS Doctor_Name,                                     -- Map Name to Doctor_Name
            d.email AS Email,                                          -- Retain Email
            CAST(d.yoe AS INT) AS YOE,                                 -- Cast YOE to INT
            s.specialitykey                                            -- Map speciality to SpecialityKey
        FROM doctor_table d
        INNER JOIN speciality_table s
            ON TRIM(d.speciality) = TRIM(s.speciality)  -- Match speciality with speciality_key
        WHERE d.doctorid IS NOT NULL AND TRIM(d.doctorid) != ''
    """


filtered_new_records = spark.sql(query)
```

```
    """


    filtered_new_records = spark.sql(query)


    resultDynamicFrame = DynamicFrame.fromDF(filtered_new_records, glueContext, "resultDynamicFrame")

    s3Sink = glueContext.getSink(
        path="s3://cancuretargetdw/doctor/",
        connection_type="s3",
        updateBehavior="LOG",
        partitionKeys=[],
        enableUpdateCatalog=True,
        transformation_ctx="s3Sink"
    )
    s3Sink.setCatalogInfo(catalogDatabase="cancure_target", catalogTableName="doctor_target")
    s3Sink.setFormat("glueparquet", compression="snappy")
    s3Sink.writeFrame(resultDynamicFrame)

    # Commit the job
    job.commit()
```

Now we join the speciality data warehouse and doctor source table to map the speciality key in the doctor datawarehouse table and we also add surrogate key to the doctor table.
Now the job is run.



Cancer Job:



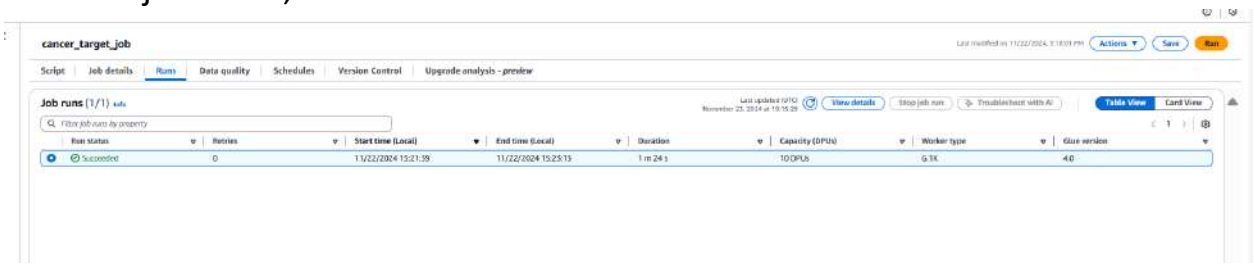
```
sourceDF.toDF().createOrReplaceTempView("source_table")

if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.stage, src.cancertype) AS CancerKey, -- Generate surrogate key
            src.stage,                                                           -- Keep the Stage column
            src.cancertype                                                       -- Keep the Cancer Type column
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.stage) = TRIM(tgt.stage) AND TRIM(src.cancertype) = TRIM(tgt.cancertype) -- Exclude duplicates
        WHERE src.stage IS NOT NULL AND src.cancertype IS NOT NULL
            AND TRIM(src.stage) != '' AND TRIM(src.cancertype) != ''  -- Exclude NULL or blank rows
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY stage, cancertype) AS CancerKey, -- Generate surrogate key
            stage,                                                       -- Keep the Stage column
            cancertype                                                   -- Keep the cancer Type column
        FROM source_table
        WHERE stage IS NOT NULL AND cancertype IS NOT NULL
            AND TRIM(stage) != '' AND TRIM(cancertype) != '' -- Exclude NULL or blank rows
    """


filtered_new_records = spark.sql(query)


filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

s3sink = glueContext.getSink(
    path="s3://cancuretargetdw/cancer/",
```

```
        WHERE stage IS NOT NULL AND cancertype IS NOT NULL
              AND TRIM(stage) != '' AND TRIM(cancertype) != '' -- Exclude NULL or blank rows
    """


filtered_new_records = spark.sql(query)


filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

s3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/cancer/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
)
s3Sink.setCatalogInfo(catalogDatabase="cancure_target", catalogTableName="cancer_target")
s3Sink.setFormat("glueparquet", compression="snappy")
s3Sink.writeFrame(filtered_new_records_DF)

# Commit the job
job.commit()

```

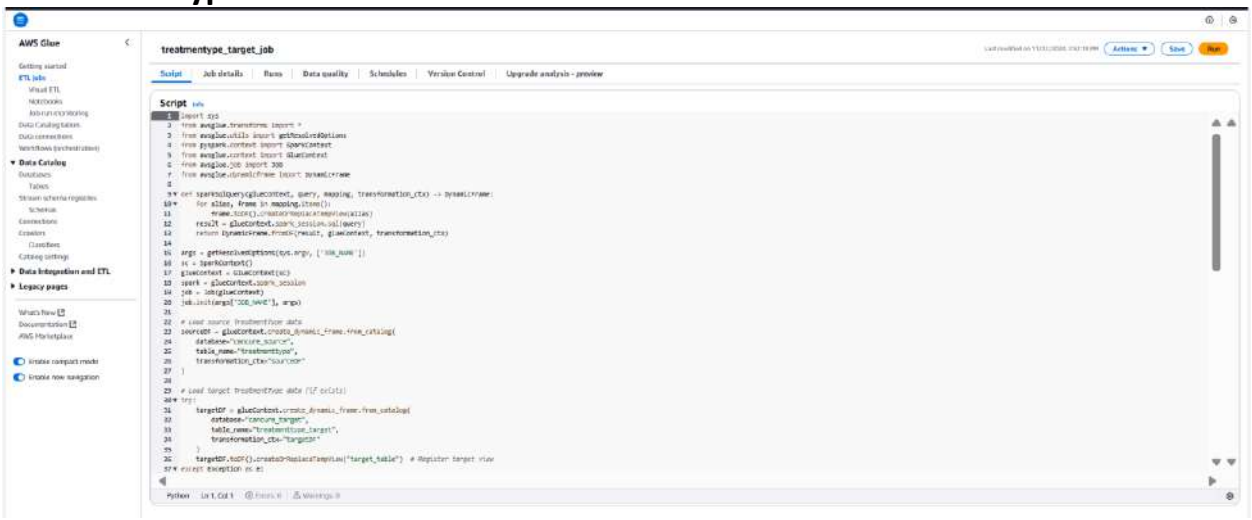
Python Ln 74, Col 1 Errors: 0 Warnings: 0

And similar to other tables we add surrogate key and load the data to the cancer data warehouse table.

Now the job is run,



Treatment type Job:

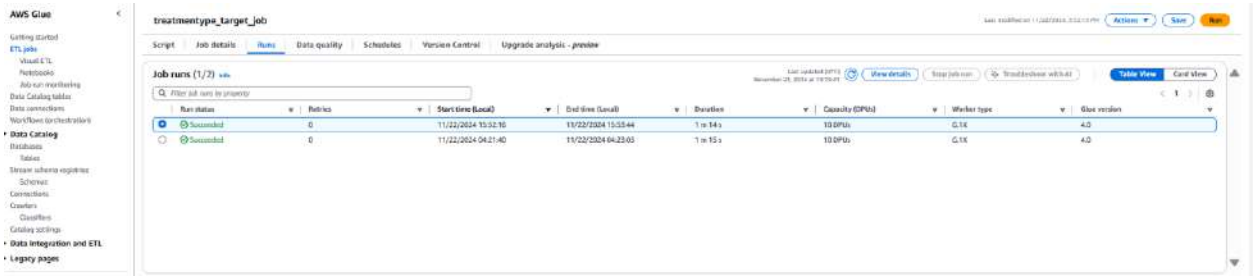


```
sourceDF.toDF().createOrReplaceTempView("source_table")

if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.TreatmentType ASC) AS TreatmentKey,  -- Generate unique keys starting from 1
            src.TreatmentType                                                    -- Include the existing TreatmentType column
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.TreatmentType) = TRIM(tgt.TreatmentType)                     -- Exclude already present records
        WHERE src.TreatmentType IS NOT NULL AND TRIM(src.TreatmentType) != ''  -- Exclude NULL or blank rows
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY TreatmentType ASC) AS TreatmentKey,  -- Generate unique keys starting from 1
            TreatmentType                                                    -- Include the existing TreatmentType column
        FROM source_table
        WHERE TreatmentType IS NOT NULL AND TRIM(TreatmentType) != ''       -- Exclude NULL or blank rows
    """
```

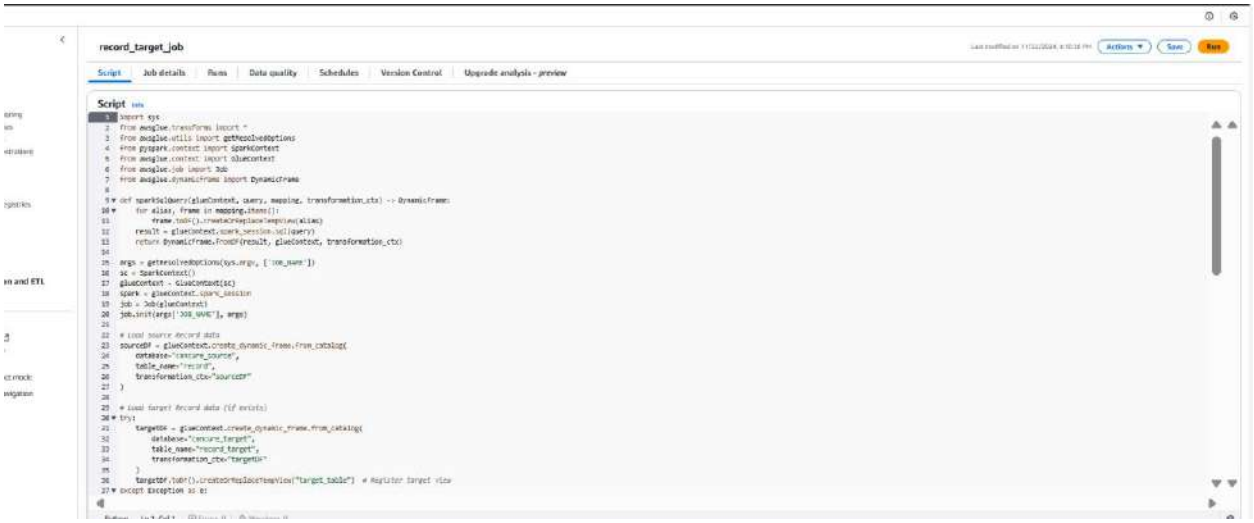
```
s3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/treatmenttype/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
)
s3Sink.setCatalogInfo(catalogDatabase="cancure_target", catalogTableName="treatmenttype_target")
s3Sink.setFormat("glueparquet", compression="snappy")
s3Sink.writeFrame(filtered_new_records_DF)

# Commit the job
job.commit()
```

Now we add the surrogate key to the treatment type datawarehouse table and load the data.

Now the job is run,



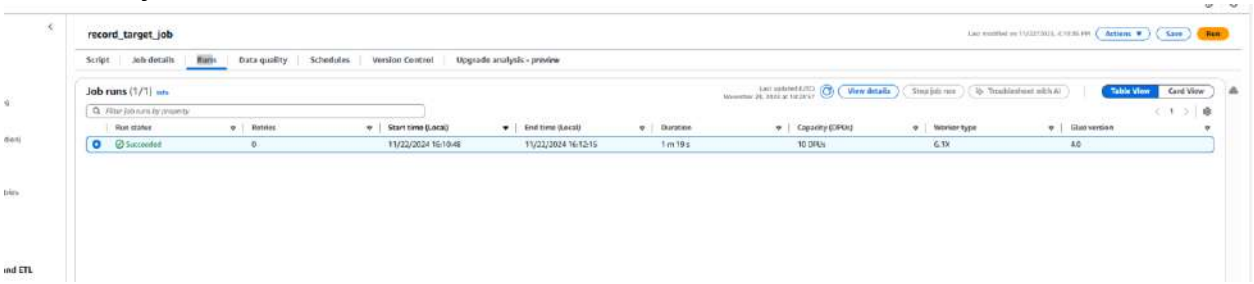
Record Job:



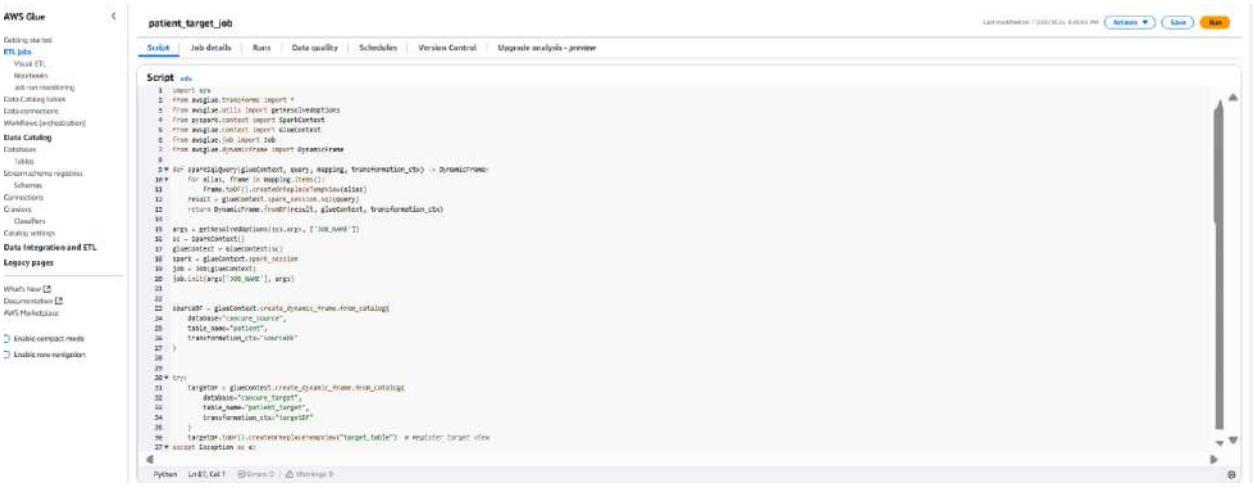
Script Info

```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.recordid ASC) AS RecordKey,  -- Generate unique keys starting from 1
            src.recordid AS RecordID,                                    -- Include RecordID
            src.treatmenttype AS TreatmentType,                          -- Include TreatmentType
            src.outcome AS Outcome                                       -- Include Outcome
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.recordid) = TRIM(tgt.RecordID)                       -- Exclude already present records
        WHERE src.recordid IS NOT NULL AND TRIM(src.recordid) != ''      -- Exclude NULL or blank rows
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY recordid ASC) AS RecordKey,  -- Generate unique keys starting from 1
            recordid AS RecordID,                                    -- Include RecordID
            treatmenttype AS TreatmentType,                          -- Include TreatmentType
            outcome AS outcome                                       -- Include outcome
        FROM source_table
        WHERE recordid IS NOT NULL AND TRIM(recordid) != ''          -- Exclude NULL or blank rows
    """


filtered_new_records = spark.sql(query)


filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")


s3Sink = glueContext.getSink(
    path="s3://cancuretargetda/record/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
```

Now we add the surrogatekey to the record datawarehouse table and load the data. Now the job is run,



Patient Job:



```
51
52 ▼ if targetDF:
53       query = """
54           SELECT
55               ROW_NUMBER() OVER (ORDER BY src.patientid) AS PatientKey, -- Generate surrogate key
56               src.patientid AS PatientID,                              -- Include PatientID
57               src.name AS Name,                                        -- Include Name
58               src.phoneno AS PhNo,                                     -- Include Phone Number
59               src.gender AS Gender,                                    -- Include Gender
60               src.diseasehistory AS DiseaseHistory,                   -- Include Disease History
61               CAST(src.age AS INT) AS Age                             -- Cast Age as INT
62           FROM source_table src
63           LEFT ANTI JOIN target_table tgt
64           ON TRIM(src.patientid) = TRIM(tgt.PatientID)  -- Avoid duplicates based on PatientID
65           WHERE src.patientid IS NOT NULL AND TRIM(src.patientid) != '' -- Exclude NULL or blank rows
66       """
67 ▼ else:
68       query = """
69           SELECT
70               ROW_NUMBER() OVER (ORDER BY patientid) AS PatientKey, -- Generate surrogate key
71               patientid AS PatientID,                              -- Include PatientID
72               name AS Name,                                        -- Include Name
73               phoneno AS PhNo,                                     -- Include Phone Number
74               gender AS Gender,                                    -- Include Gender
75               diseasehistory AS DiseaseHistory,                   -- Include Disease History
76               CAST(age AS INT) AS Age                             -- Cast Age as INT
77           FROM source_table
78           WHERE patientid IS NOT NULL AND TRIM(patientid) != '' -- Exclude NULL or blank rows
79       """
80
81
```

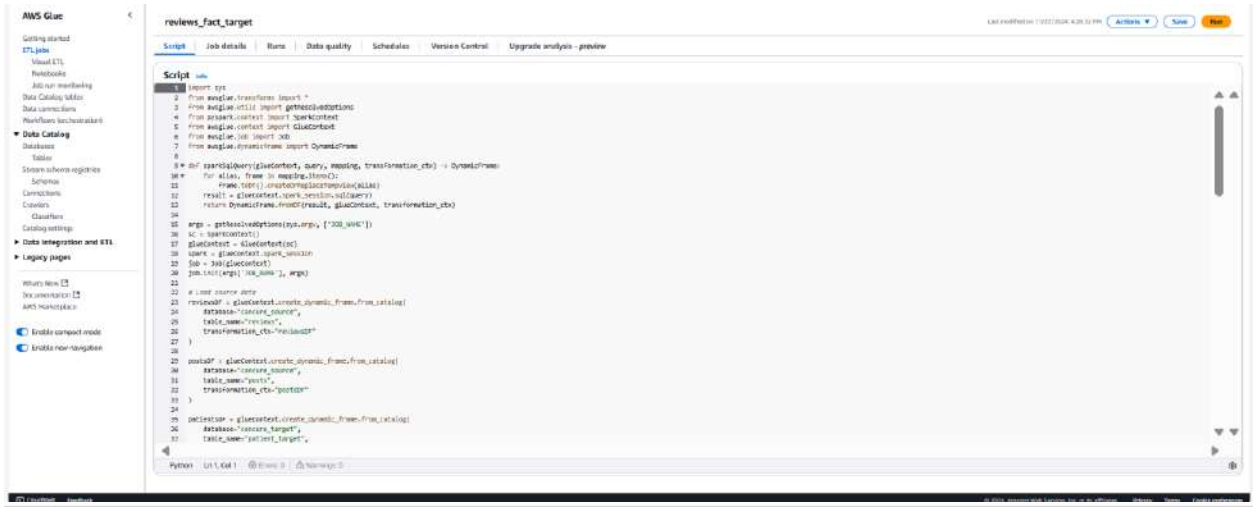
```
filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

s3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/patient/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
)
s3Sink.setCatalogInfo(catalogDatabase="cancure_target", catalogTableName="patient_target")
s3Sink.setFormat("glueparquet", compression="snappy")
s3Sink.writeFrame(filtered_new_records_DF)

# Commit the job
job.commit()
```

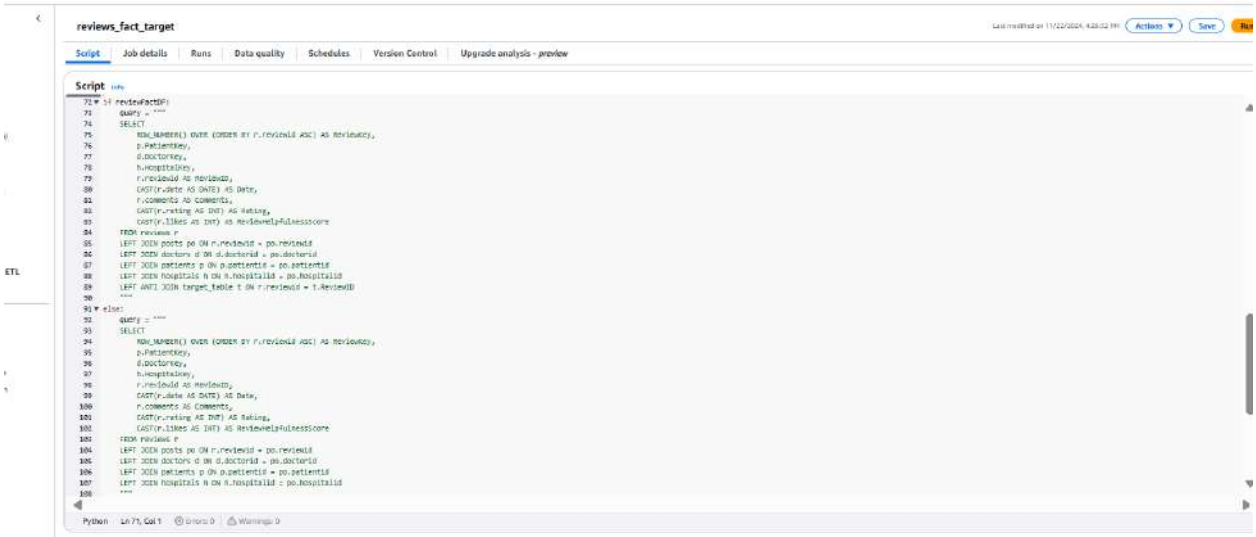
The same process is done for the patient job.

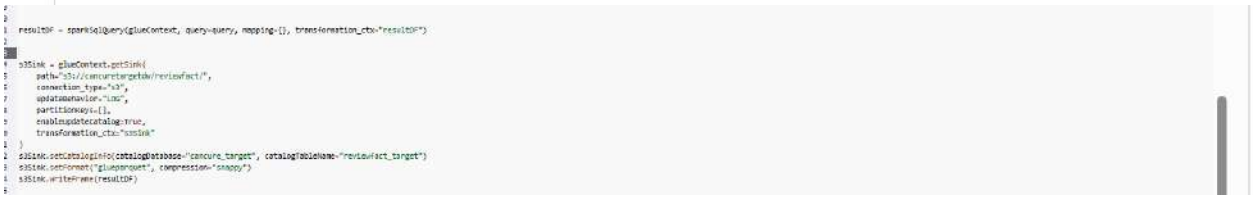
The job is run,

Fact Tables:

Review Fact:

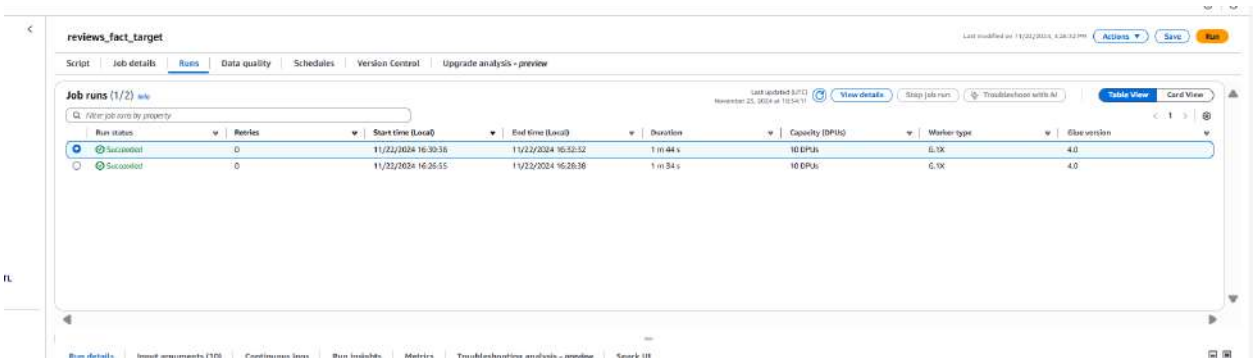






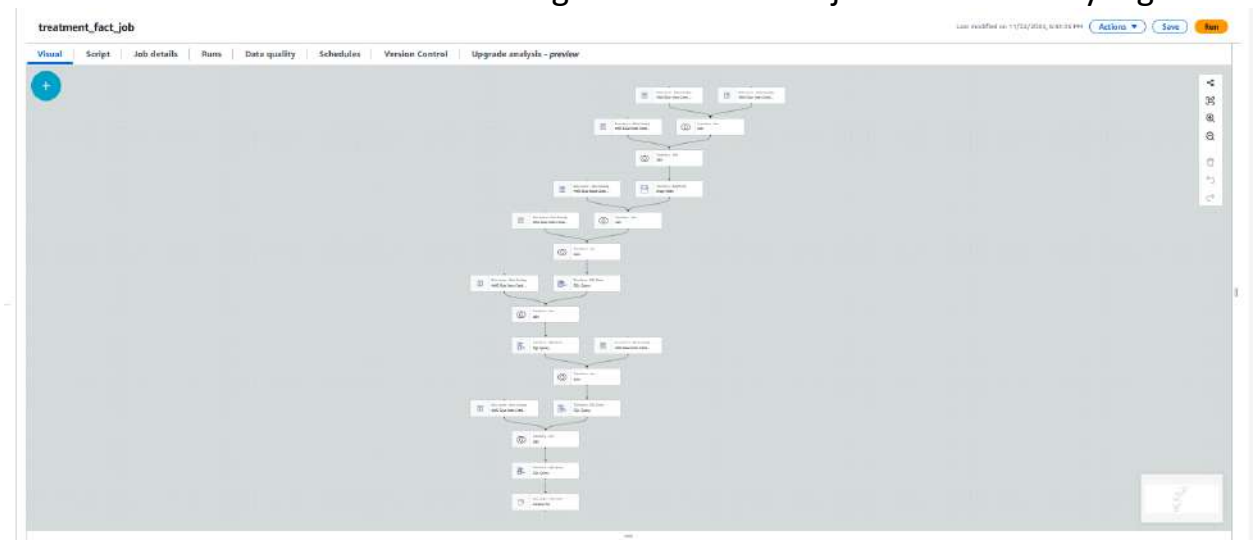
Now for the reviews fact table in addition to the source table we join the patient, hospital and doctor fact table to map their respective keys. And we also added the surrogate key for the review fact table.

The job is run,

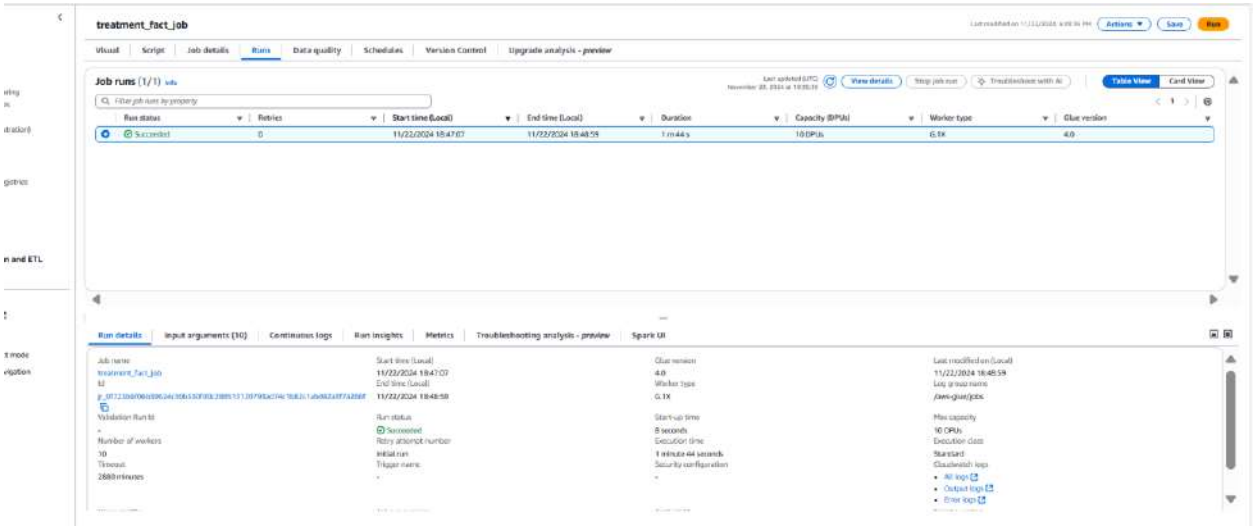


Treatment Fact Table:

For treatment fact table we are using visual AWS Glue job since it is very big:



For the treatment fact table in addition to the source table we map various data warehouse tables like patient, doctor, hospital, cancer, record in order to map their corresponding keys and we also add a surrogate key to the treatment fact table.



Thus all the jobs are successfully executed and the tables are loaded in the target database,

Now a general explanation of individual jobs are completed now the following are the transformations done in the process,

Transformations:

We have done multiple transformations in many tables,

City Table:

```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY src.CityName ASC) AS CityKey,
            src.CityName,
            CAST(src.Population AS INT) AS Population,
            CAST(src.StateKey AS INT) AS StateKey
        FROM source_table src
        LEFT ANTI JOIN target_table tgt
        ON TRIM(src.CityName) = TRIM(tgt.CityName) AND src.StateKey = tgt.StateKey
        WHERE src.CityName IS NOT NULL AND TRIM(src.CityName) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY CityName ASC) AS CityKey,
            CityName,
            CAST(Population AS INT) AS Population,
            CAST(StateKey AS INT) AS StateKey
        FROM source_table
        WHERE CityName IS NOT NULL AND TRIM(CityName) != ''
    """


filtered_new_records = spark.sql(query)

filtered_new_records_DF = DynamicFrame.fromDF(filtered_new_records, glueContext, "filtered_new_records_DF")

# Write the new records to the target table in s3 and update the Glue catalog
s3Sink = glueContext.getSink(
    path="s3://cancuretargetdw/city/",
    connection_type="s3",
    updateBehavior="LOG",
    partitionKeys=[],
    enableUpdateCatalog=True,
    transformation_ctx="s3Sink"
```

We have transformed the datatype of the incoming population column from string to int before loading it into the city datawarehouse table.

Doctor table

```
if targetDF:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY d.doctorid ASC) AS DoctorKey,  -- Generate sequential DoctorKey
            d.doctorid,                                                -- Retain Doctor_ID
            d.name AS Doctor_Name,                                     -- Map Name to Doctor_Name
            d.email AS Email,                                          -- Retain Email
            CAST(d.yoe AS INT) AS YOE,                                 -- Cast YOE to INT
            s.specialitykey                                            -- Map speciality to SpecialityKey
        FROM doctor_table d
        LEFT ANTI JOIN target_table t
            ON TRIM(d.doctorid) = TRIM(t.doctorid)  -- Avoid duplicates
        INNER JOIN speciality_table s
            ON TRIM(d.speciality) = TRIM(s.speciality)  -- Match speciality with speciality_key
        WHERE d.doctorid IS NOT NULL AND TRIM(d.doctorid) != ''
    """
else:
    query = """
        SELECT
            ROW_NUMBER() OVER (ORDER BY d.doctorid ASC) AS DoctorKey,  -- Generate sequential DoctorKey
            d.doctorid,                                                -- Retain Doctor_ID
            d.name AS Doctor_Name,                                     -- Map Name to Doctor_Name
            d.email AS Email,                                          -- Retain Email
            CAST(d.yoe AS INT) AS YOE,                                 -- Cast YOE to INT
            s.specialitykey                                            -- Map speciality to SpecialityKey
        FROM doctor_table d
        INNER JOIN speciality_table s
            ON TRIM(d.speciality) = TRIM(s.speciality)  -- Match speciality with speciality_key
        WHERE d.doctorid IS NOT NULL AND TRIM(d.doctorid) != ''
    """


filtered_new_records = spark.sql(query)
```

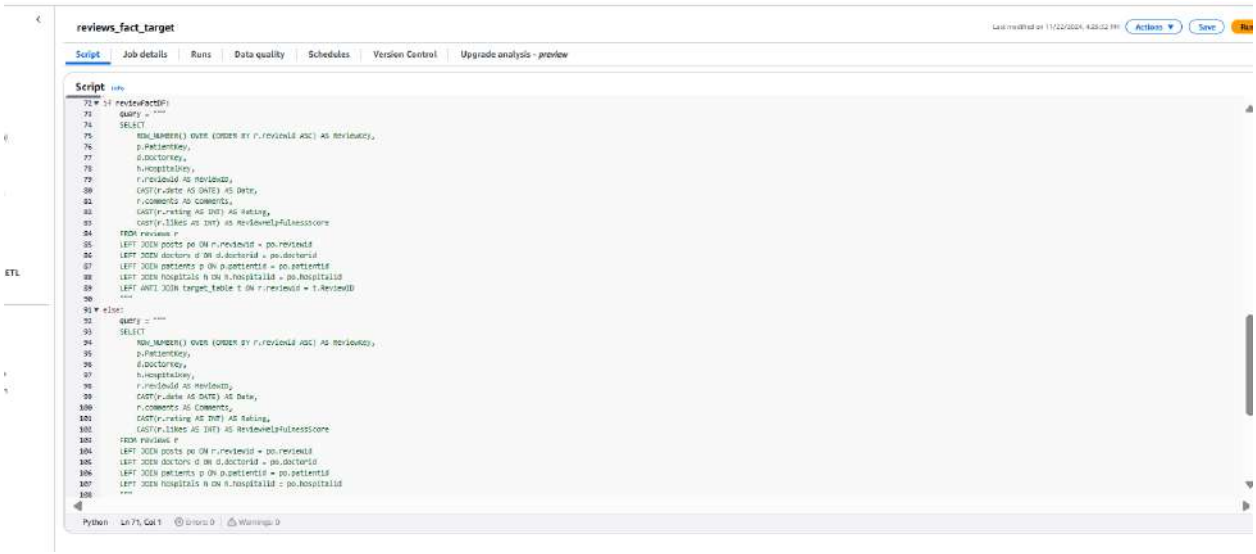
similarly even in the doctor table the Years of experience (YOE) column is transformed from from string to int before loading it into the doctor data warehouse table.

Patient:

```
51
52 if targetDF:
53     query = """
54         SELECT
55             ROW_NUMBER() OVER (ORDER BY src.patientid) AS PatientKey, -- Generate surrogate key
56             src.patientid AS PatientID,                               -- Include PatientID
57             src.name AS Name,                                         -- Include Name
58             src.phoneno AS PhNo,                                      -- Include Phone Number
59             src.gender AS Gender,                                     -- Include Gender
60             src.diseasehistory AS DiseaseHistory,                    -- Include Disease History
61             CAST(src.age AS INT) AS Age                               -- Cast Age as INT
62         FROM source_table src
63         LEFT ANTI JOIN target_table tgt
64         ON TRIM(src.patientid) = TRIM(tgt.PatientID)  -- Avoid duplicates based on PatientID
65         WHERE src.patientid IS NOT NULL AND TRIM(src.patientid) != '' -- Exclude NULL or blank rows
66     """
67 else:
68     query = """
69         SELECT
70             ROW_NUMBER() OVER (ORDER BY patientid) AS PatientKey, -- Generate surrogate key
71             patientid AS PatientID,                               -- Include PatientID
72             name AS Name,                                         -- Include Name
73             phoneno AS PhNo,                                      -- Include Phone Number
74             gender AS Gender,                                     -- Include Gender
75             diseasehistory AS DiseaseHistory,                    -- Include Disease History
76             CAST(age AS INT) AS Age                               -- Cast Age as INT
77         FROM source_table
78         WHERE patientid IS NOT NULL AND TRIM(patientid) != '' -- Exclude NULL or blank rows
79     """
80
81
```

Here the datatype of the age column is transformed from string to int before loading it into the patient datawarehouse table.

Review Fact:



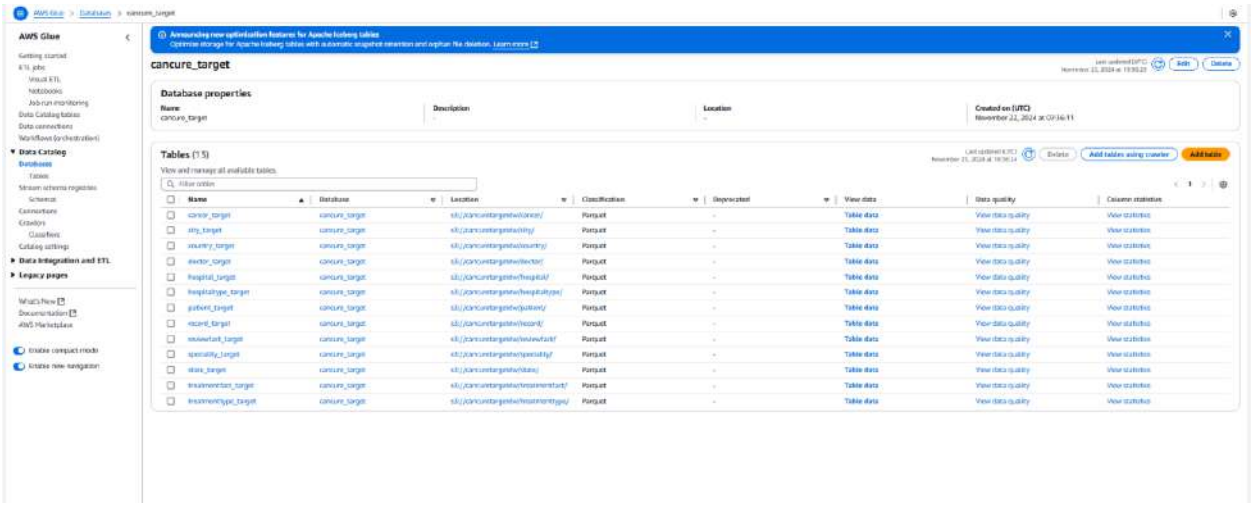
Similarly the datatype of rating and likes are transformed from string to int before loading in the target table.

Treatment Fact Table:

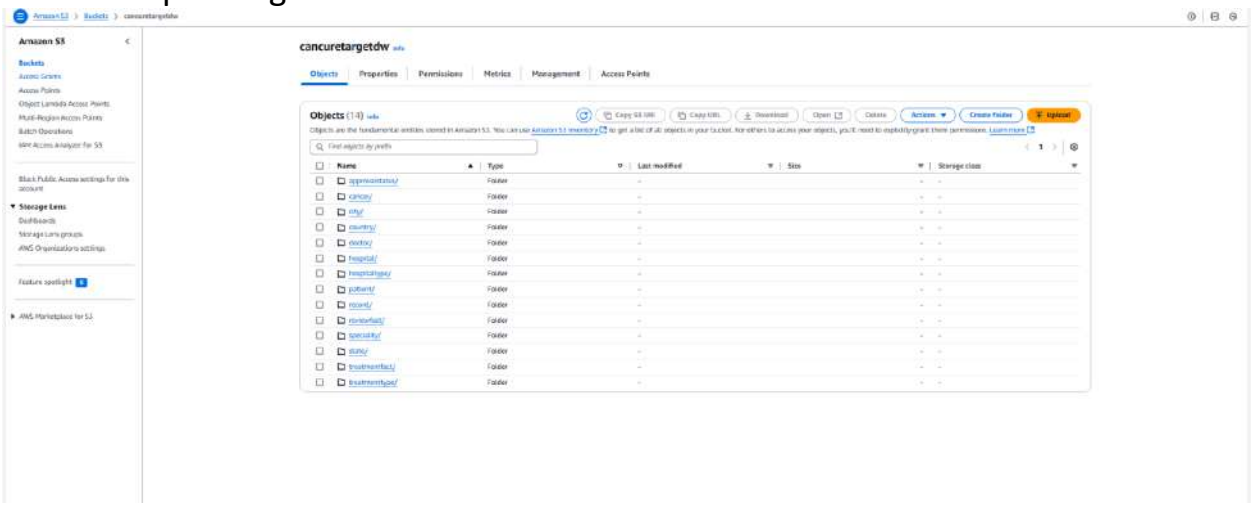
As we can see we have done multiple transformations in the treatment fact table. **The startday and endday string coming from the source table is converted to date and then subtracted to derive duration in int.**

Similarly the bill and the insurance string fields from the source table is converted to decimal before loading into the treatment fact table.

Now since all the tables are loaded into the datawarehouse catalog database named cancure_target, the below is the screenshot.

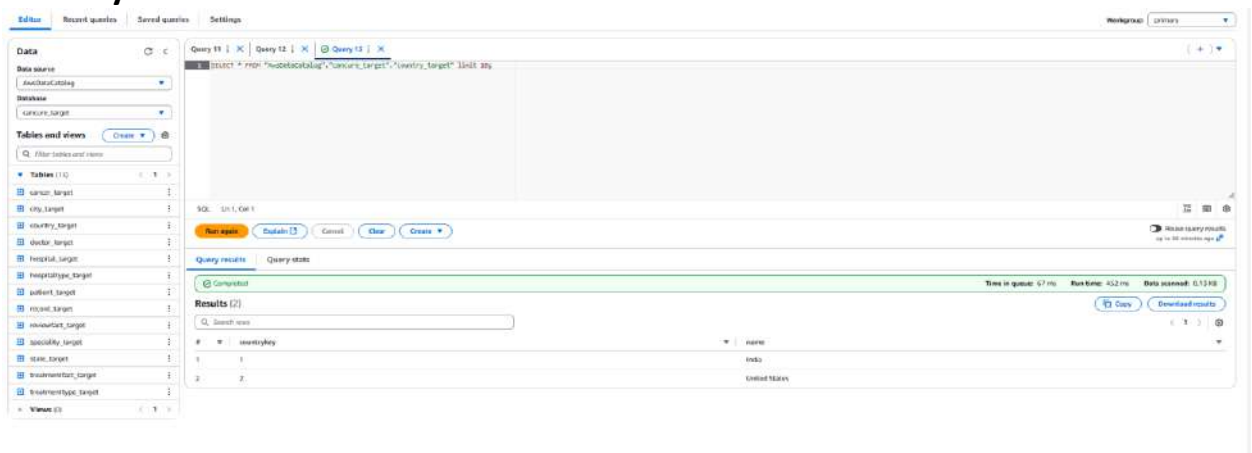


The corresponding S3 bucket location is also shown.

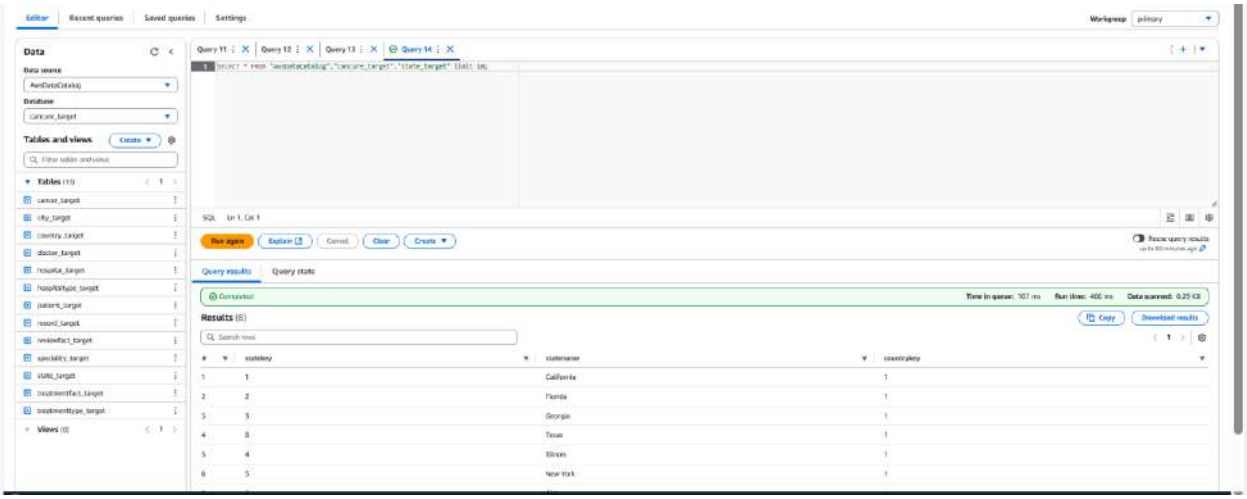


Now checking the outputs with AWS Athena,
As we can see all tables are loaded **with surrogate keys and required structure.**
For each table:

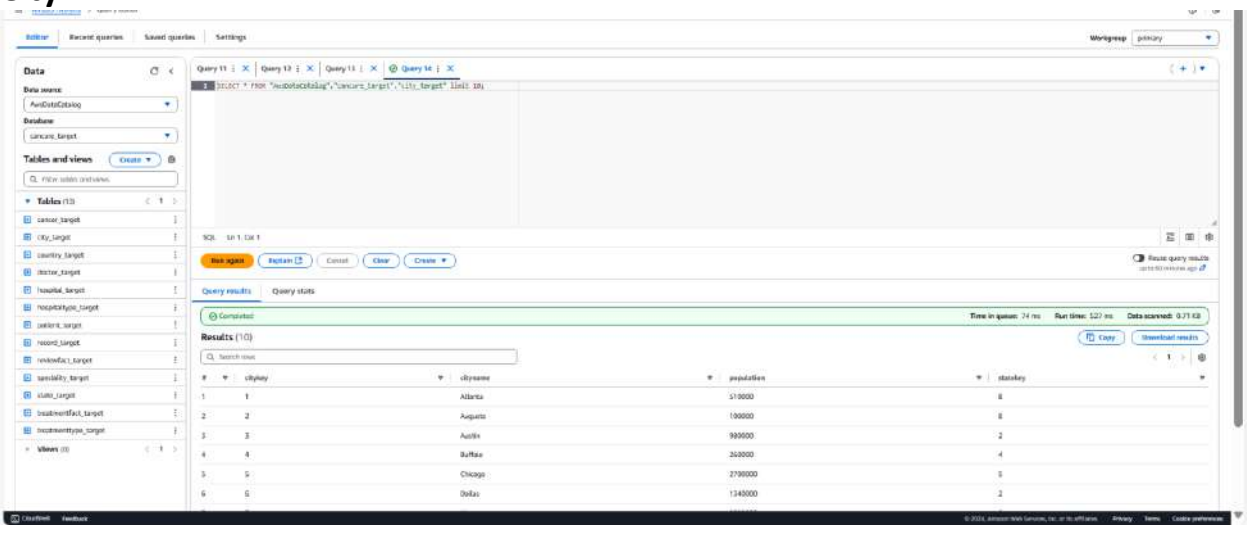
Country:



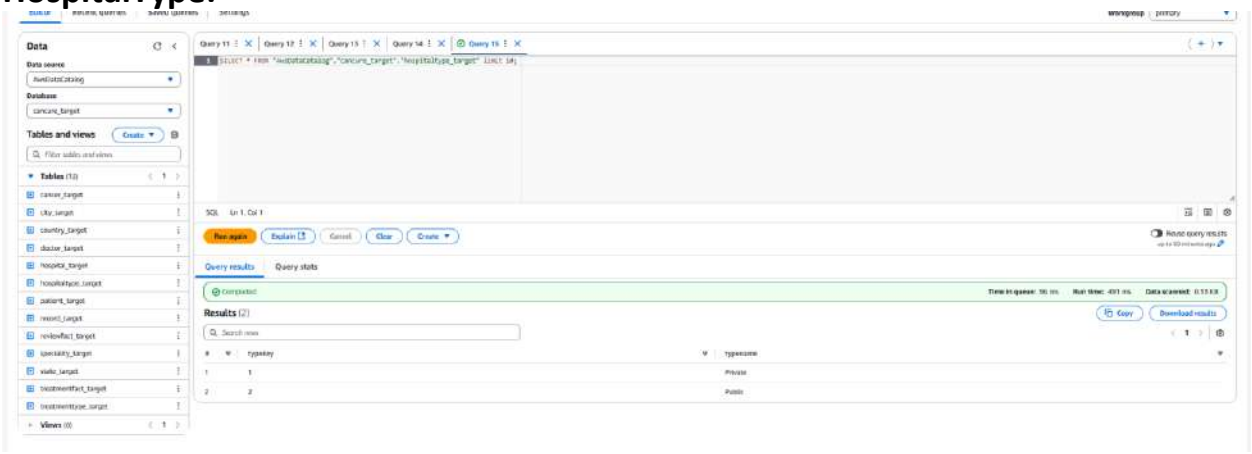
State:



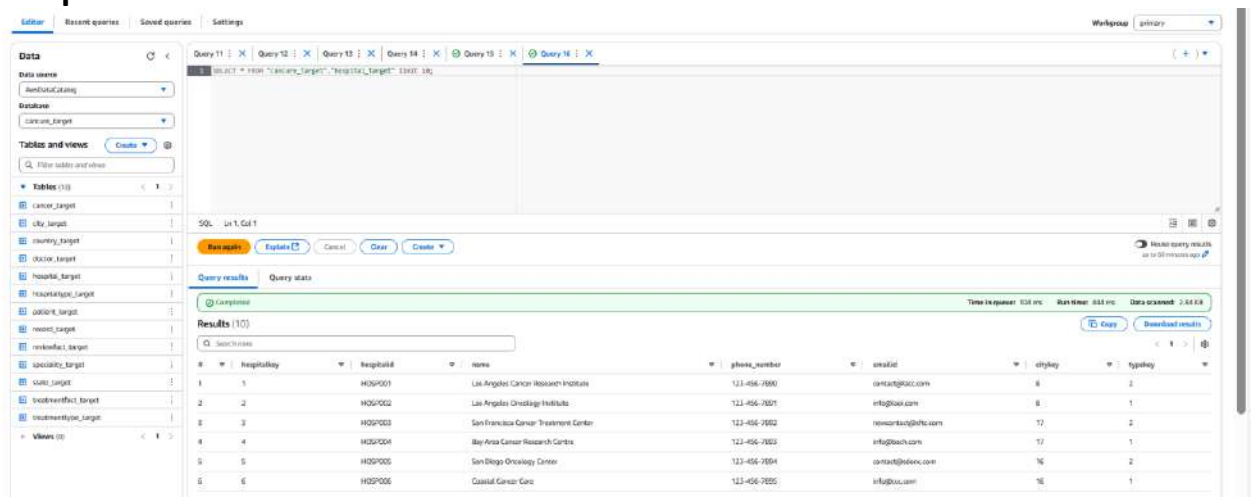
City:



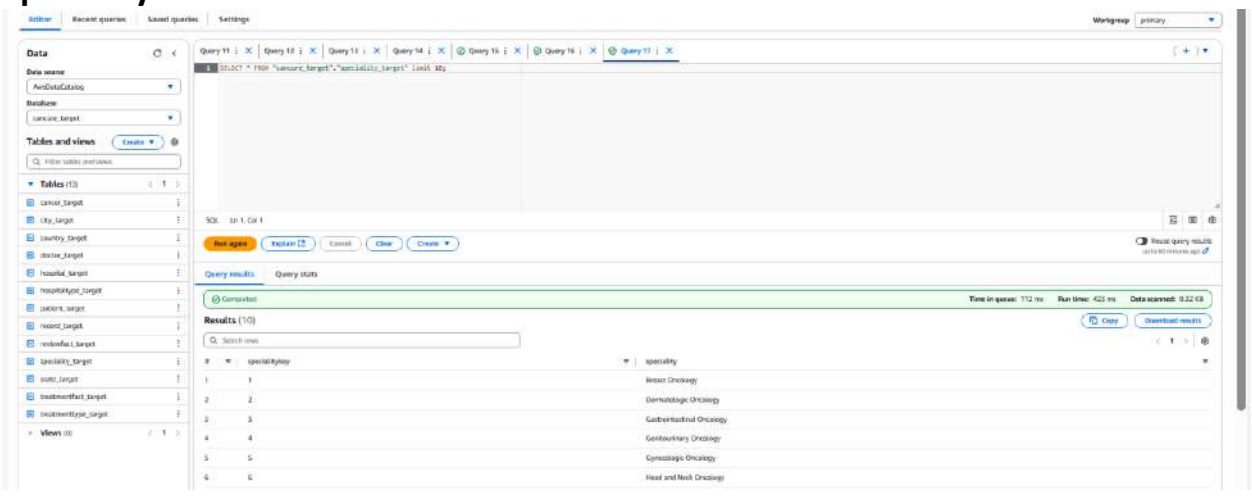
HospitalType:



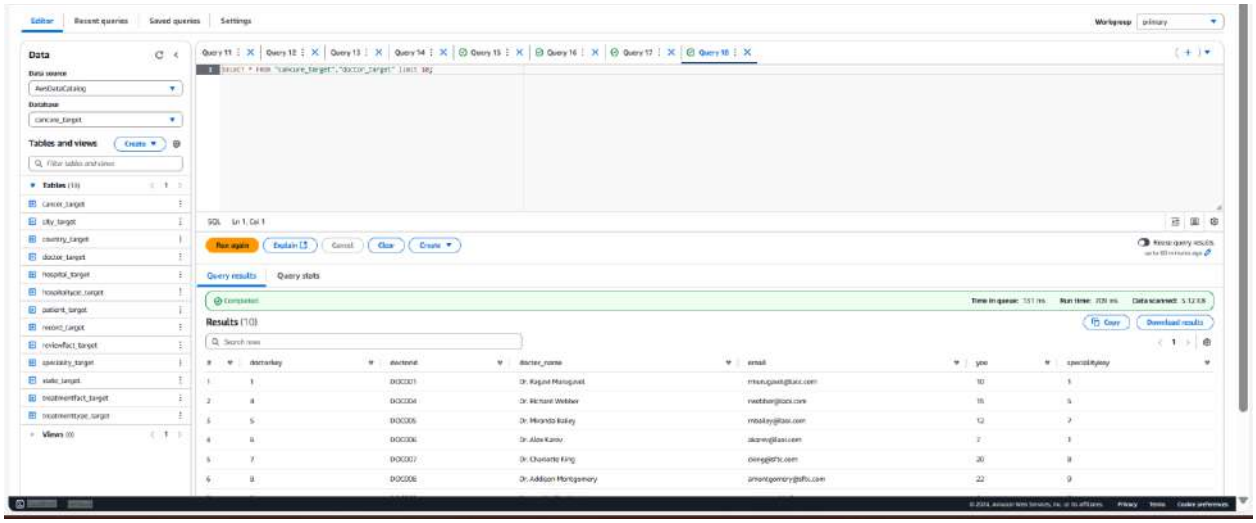
Hospital:



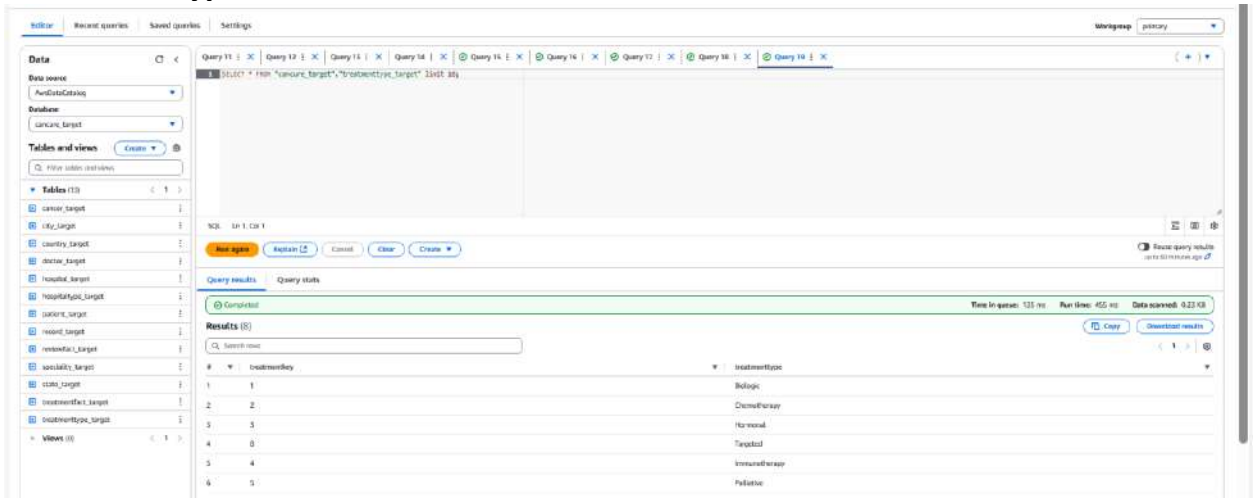
Speciality:



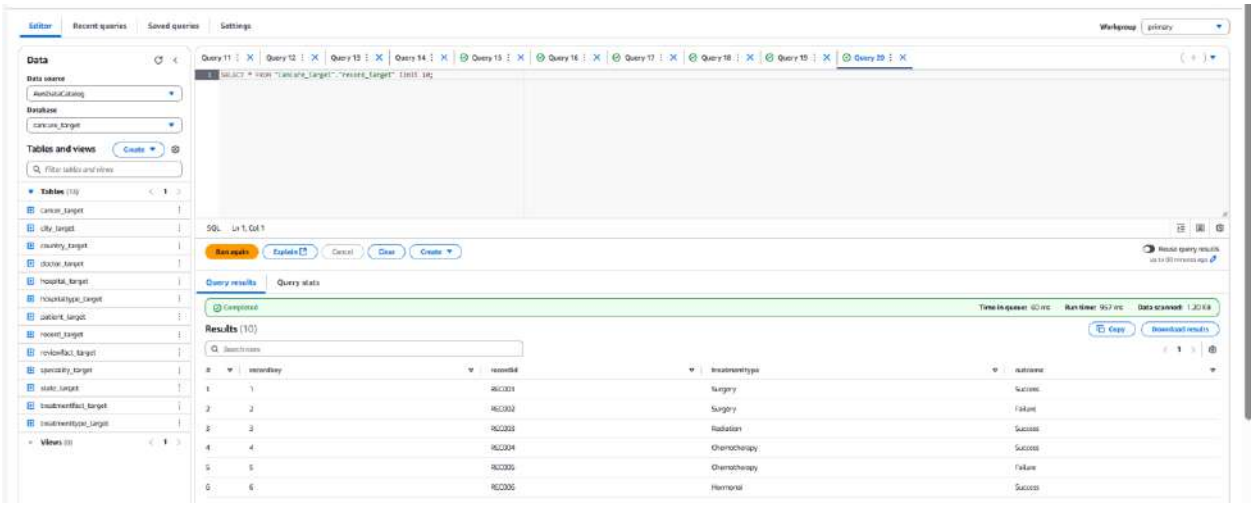
Doctor:



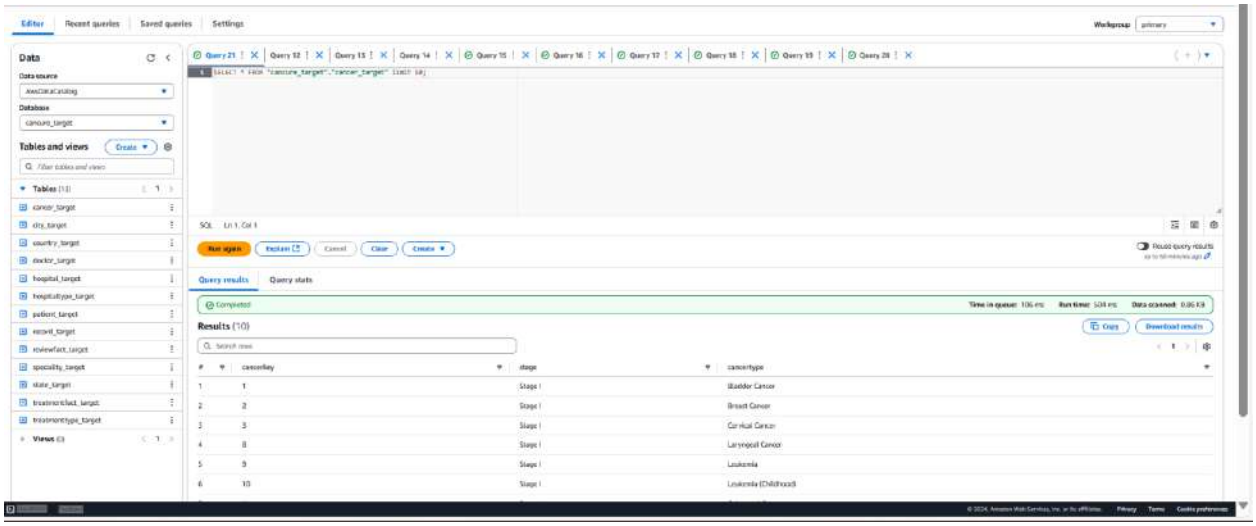
TreatmentType:



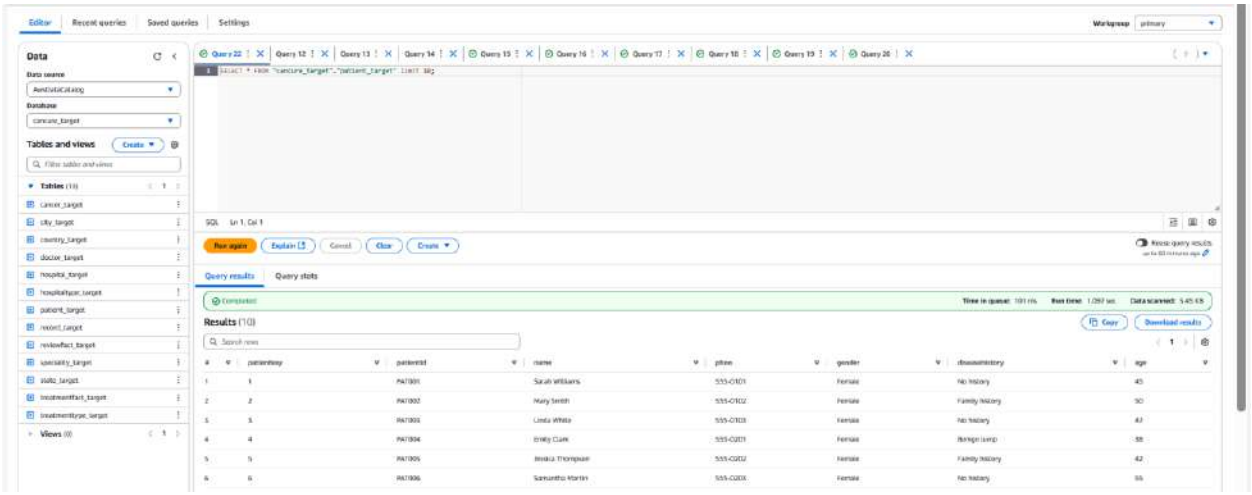
Record:



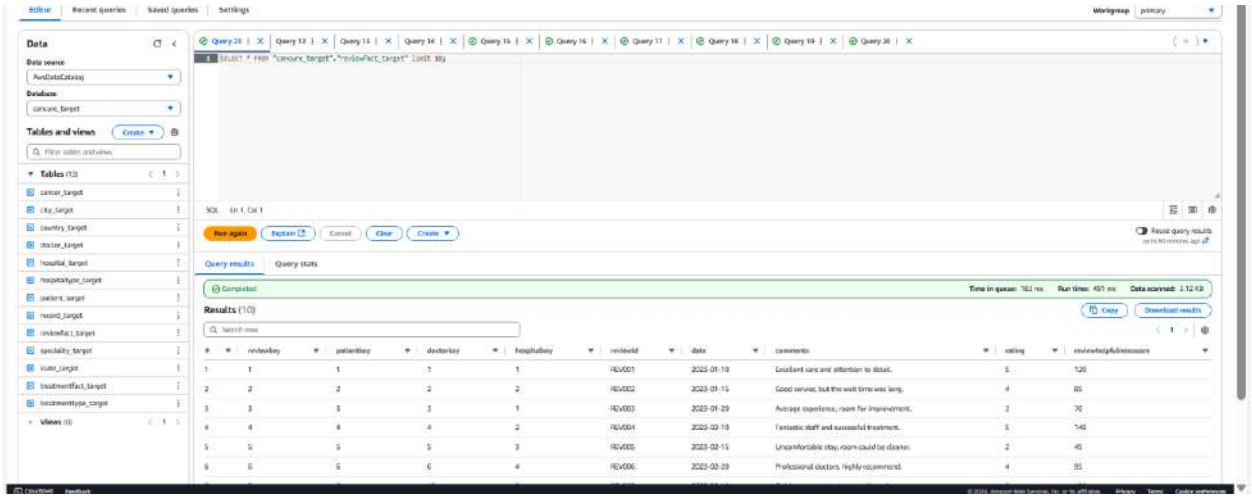
Cancer:



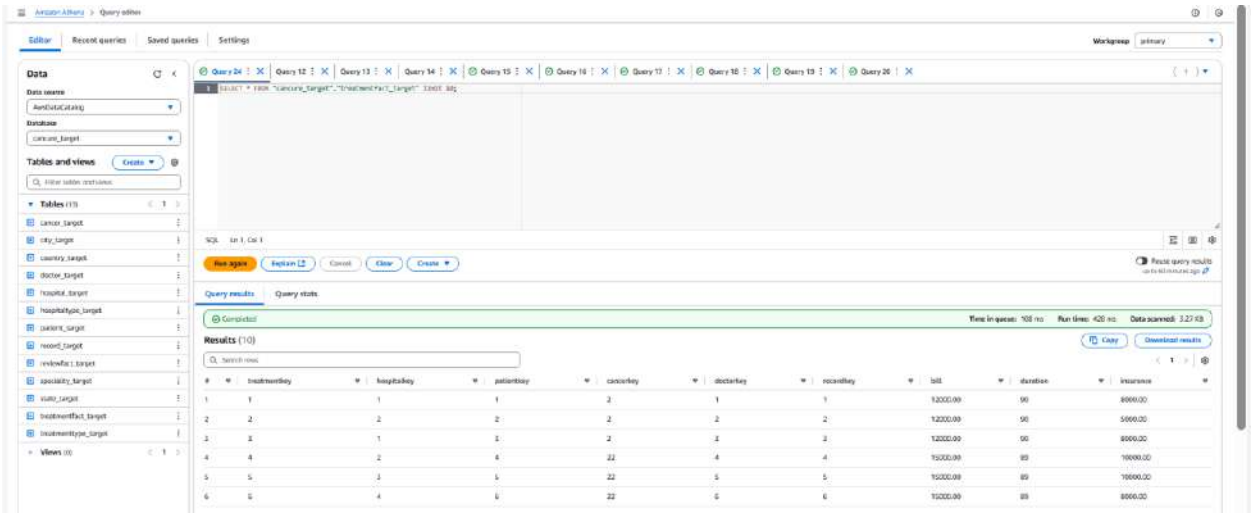
Patient:



Reviews Fact Table:



Treatment Fact Table:

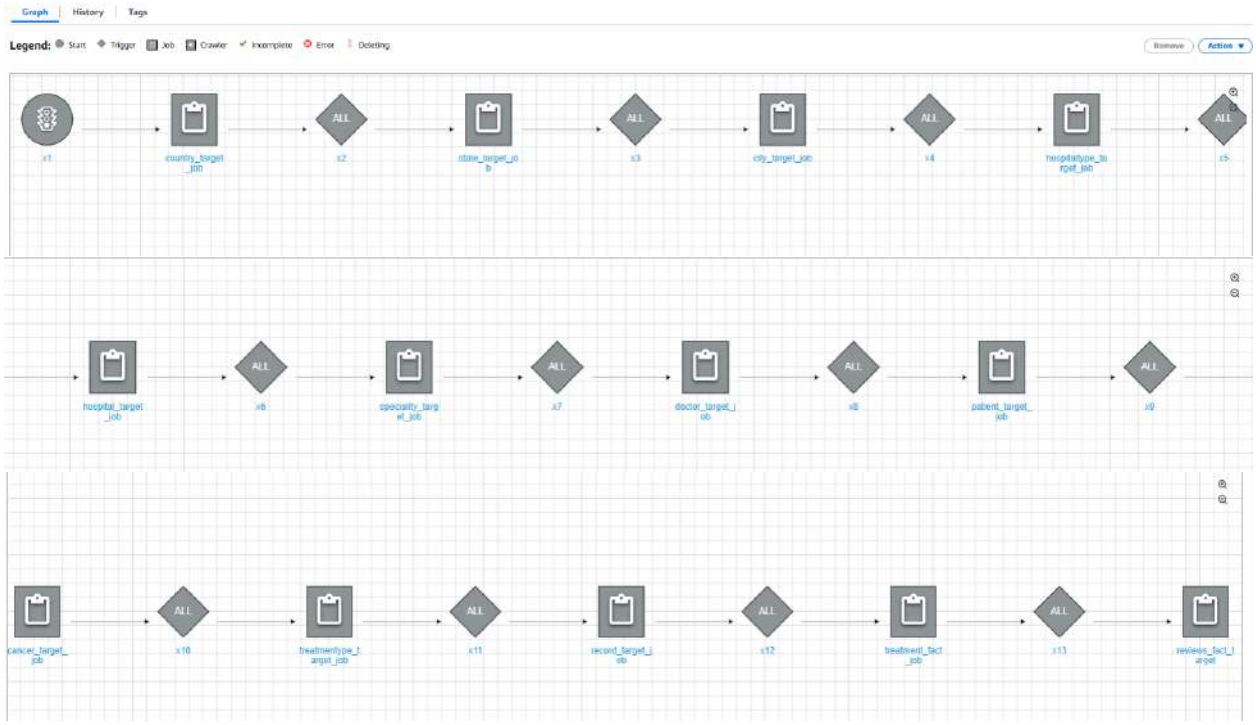


Thus all tables (dimensions and facts) are loaded properly.

Complete Workflow Pipeline:

We have also built a complete pipeline of the workflows and ran it , we are not running it again in the other account as it is incurring a lot of cost.

Now we are attaching the pictures of the workflow, below



After running this job the entire database got populated again with all the tables.

Thus we have also presented the logical architecture diagram and also the individual jobs and the final complete workflow orchestration pipeline diagram and loaded the data using both the methods. All the jobs without any dependency are added in the beginning followed by jobs which are dependent on the other jobs with treatment and review fact tables coming at the end.

Therefore we have satisfied all the requirements by **including 4+ tools to build the pipeline**, **presenting the architecture**(Individual jobs, logical architecture diagram, entire workflow pipeline), **Using programming language** like python to convert my source files into the required format and **using two different sources**, and **executing** and **loading all the jobs,pipeline and tables successfully.**

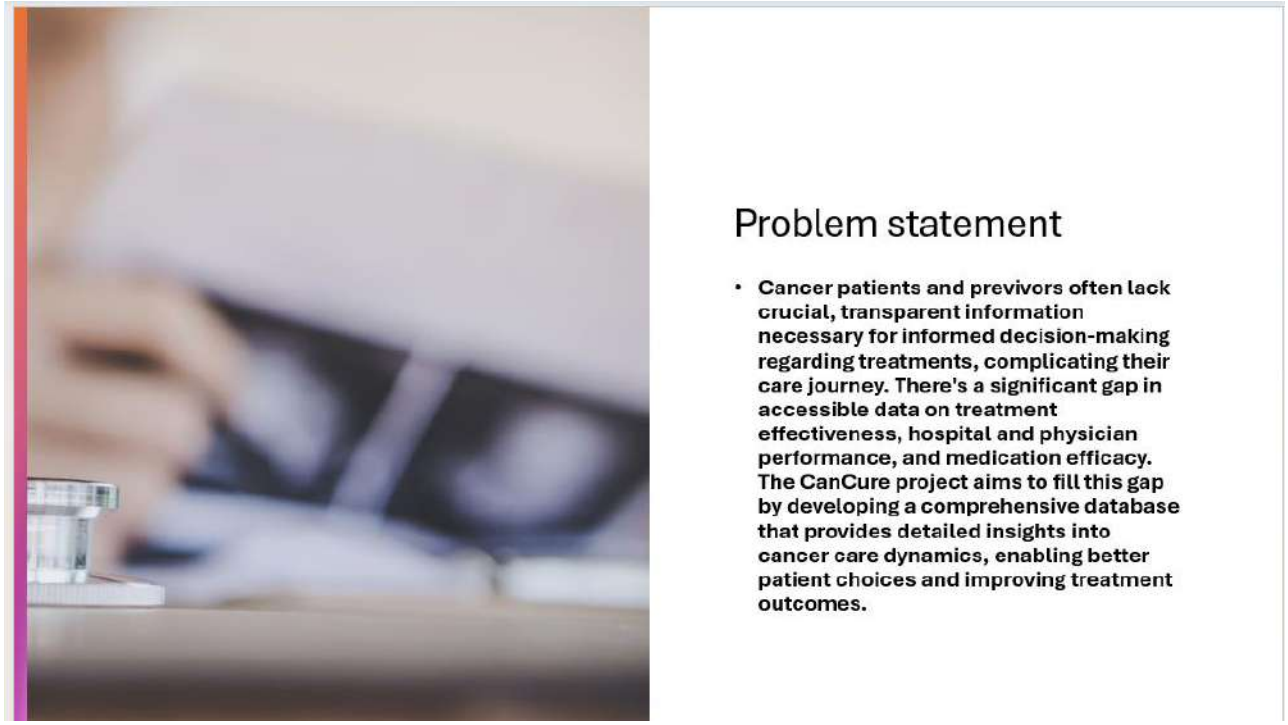
Conclusion:

The CanCure project successfully demonstrates the implementation of on-premise and cloud-based solutions for comprehensive data warehouses in the healthcare sector. By implementing a robust ETL pipeline using Talend and in the cloud using AWS Glue, S3 buckets, and Athena, the project captures and processes critical data, enabling meaningful insights. Dashboards created for treatment, reviews, and supplies provide actionable KPIs, such as treatment duration, hospital performance, and insurance coverage, empowering patients and stakeholders. This initiative bridges gaps in cancer care by improving transparency, enhancing decision-making, and driving better patient outcomes. The CanCure system exemplifies how technological advancements can revolutionize healthcare data analytics, setting a benchmark for future healthcare projects.

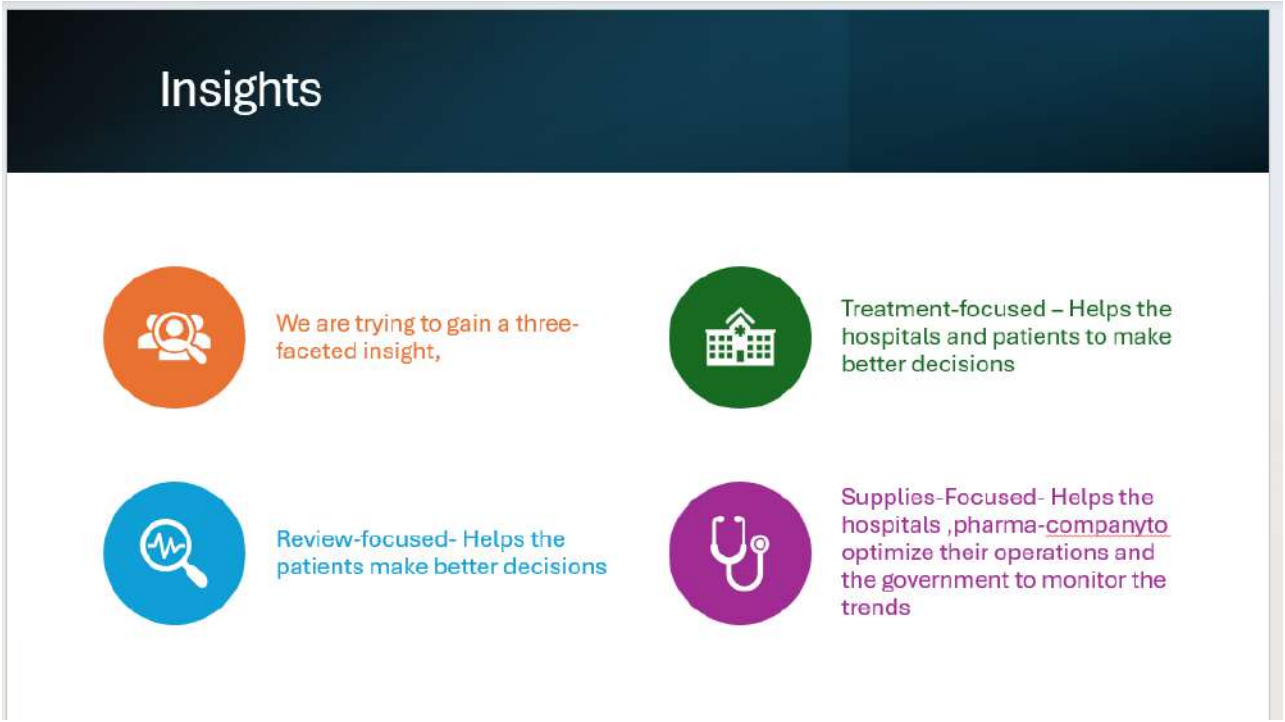
PRESENTATION MATERIAL:

SLIDE 1:

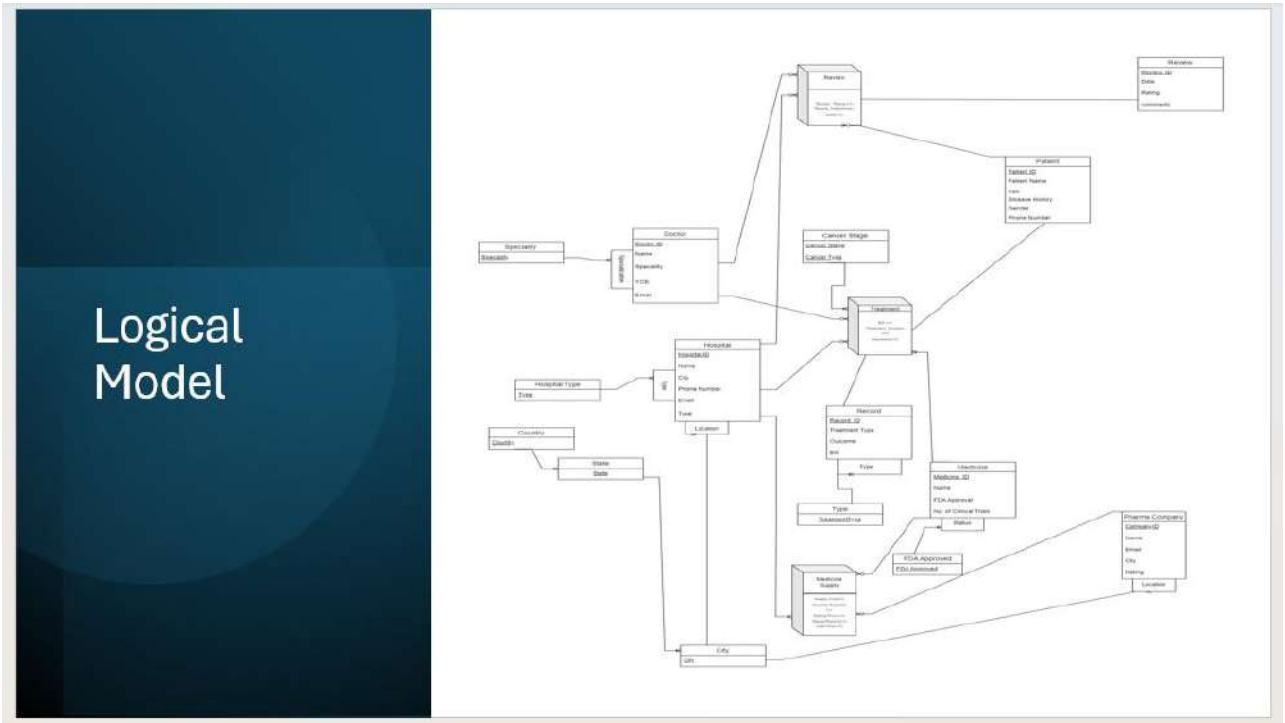
SLIDE 2



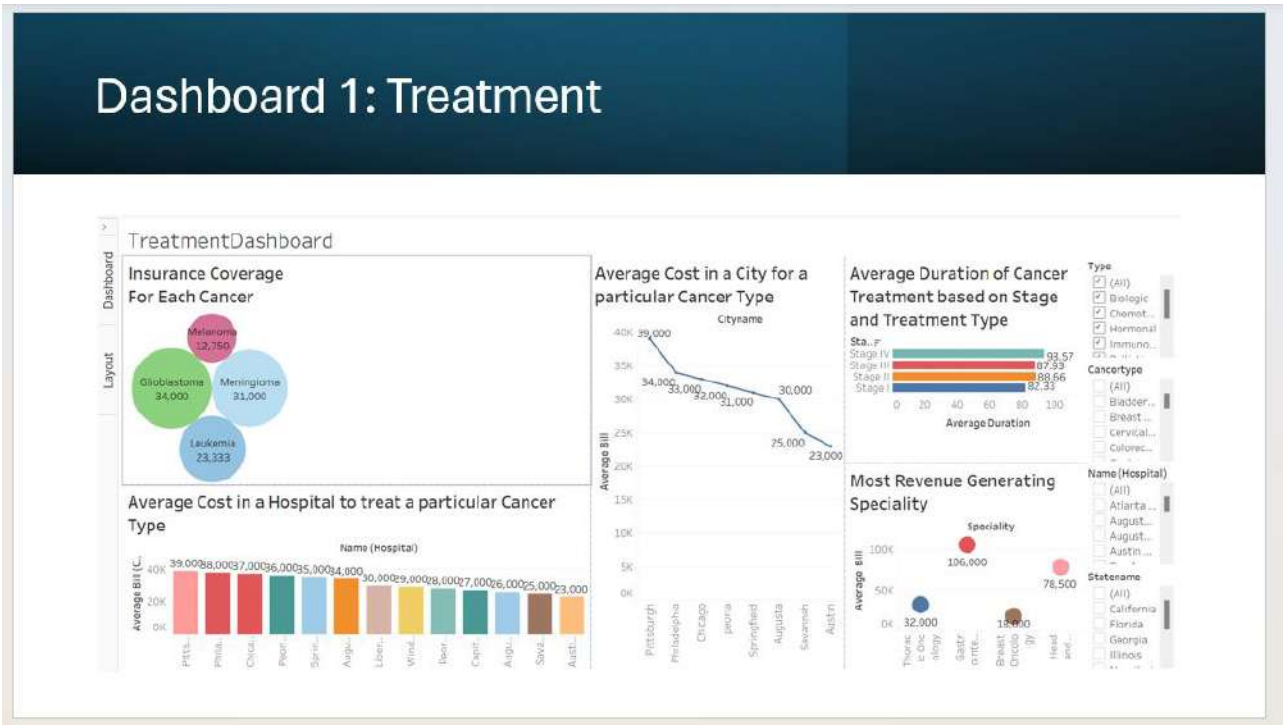
SLIDE 3:



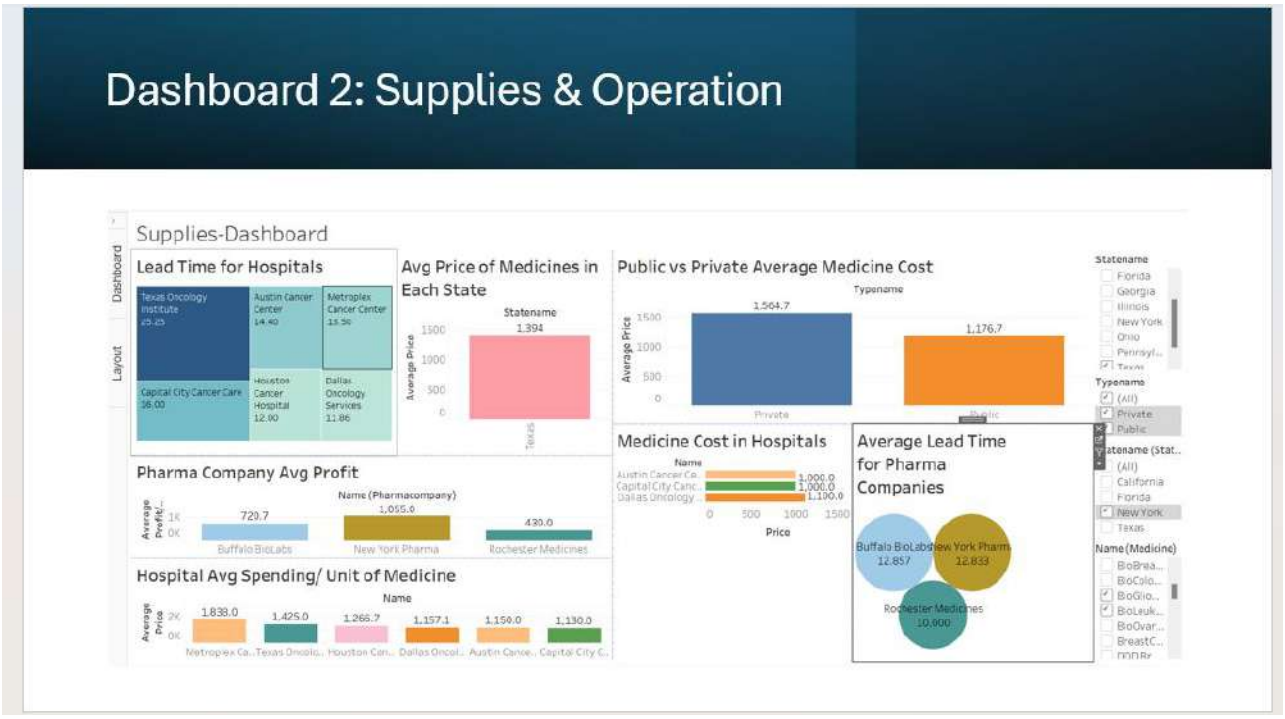
SLIDE 4:



SLIDE 5:



SLIDE 6:



SLIDE 7:

SLIDE 8:

THANK YOU!