INTRODUCTION

System Background:

The Medical Clinic Management System is a comprehensive database system put in place by QualityLife Healthcare with the aim of streamlining and enhancing the operations of a busy medical clinic. The system was created in response to the demand for data-driven decision-making in the healthcare sector, the increased complexity of patient care, and the requirement for effective provider management.

Features and Objectives:

1. Patient Management: The system allows clinic staff to efficiently manage patient information, including demographics, medical history, and appointment scheduling.
2. Provider Management: It enables healthcare providers to maintain their profiles, including specialization, licenses, and contact information, making it easy for patients to choose the right provider for their needs.
3. Appointment Scheduling: Patients can conveniently schedule appointments online or through the clinic's front desk. The system optimizes appointment slots, minimizes wait times, and ensures efficient allocation of resources.
4. Tracking of procedures: Names, descriptions, and related costs of procedures carried out throughout appointments are documented to help with invoicing and reporting.

Intended Report:

Provider Performance Report. This report evaluates the performance of healthcare providers based on metrics such as appointment counts, average satisfaction ratings, appointment durations, and wait times. It assists in recognizing top-performing providers and areas that may need improvement.

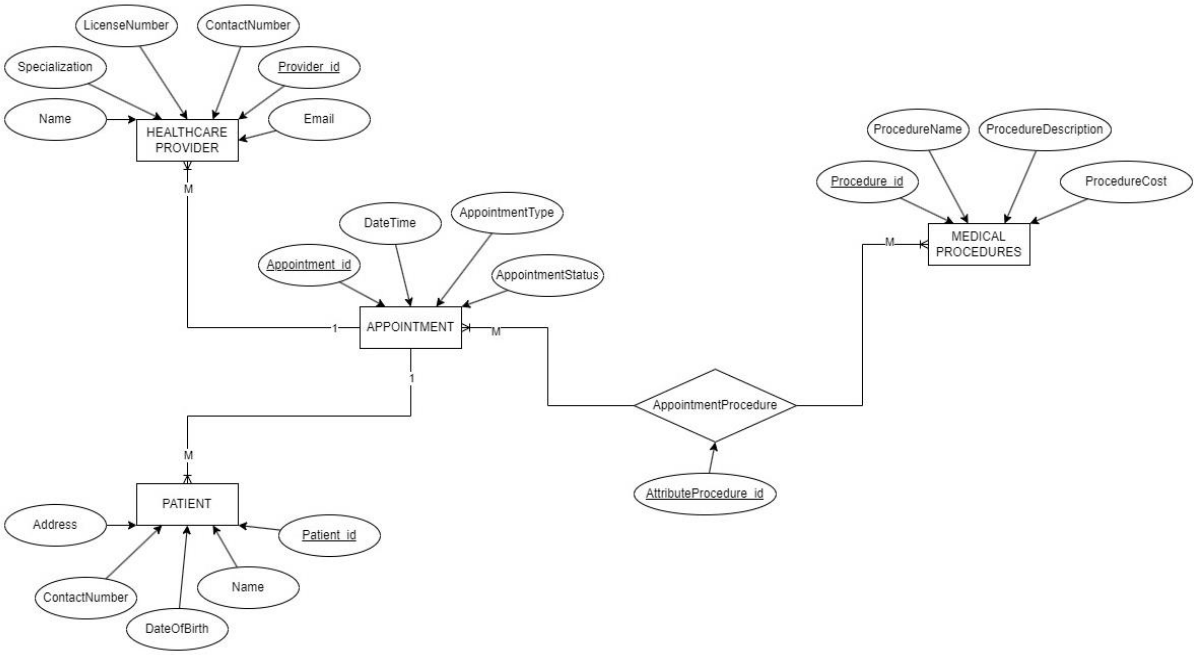

Figure 1. Entity-Relational Diagram

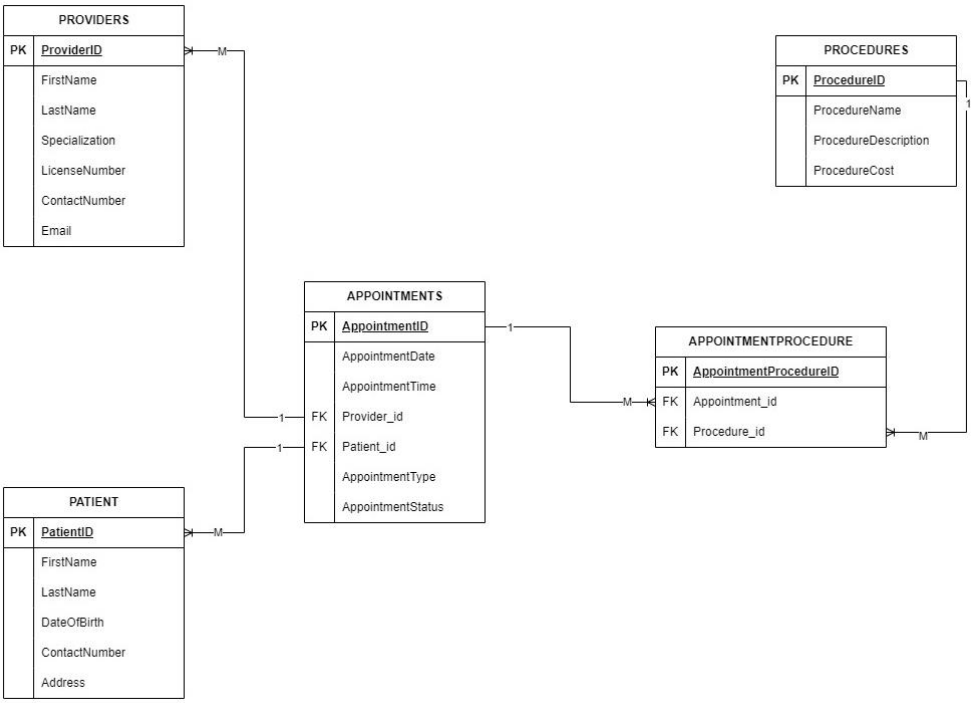

Figure 2. Normalized Relational Model

After creating ERD Figure 1 on page 1, we proceeded to convert it into a Normalized Relational Model (NRM), as depicted in Figure 2. Utilizing the NRM, we can derive the dimension tables necessary for our data warehouse. These dimension tables, in conjunction with the Fact table, will enable us to generate the desired reports mentioned on page 1. Please refer to the table below for the derived dimension tables.

Methodology of Dimensional Normal Form:

DIMENSION TABLE	ATTRIBUTES	DIMENSION TABLE	ATTRIBUTES
Time	TimeID (Primary Key)	Provider	ProviderID (Primary Key)
	Date		FirstName
	Day of the Week		LastName
	Month		Specialization
	Quarter		LicenseNumber
	Year		ContactNumber
Specialization	SpecializationID (Primary Key)		Email
	Specialization		

FACT TABLE	ATTRIBUTES
Provider Performance	PerformanceID (Prime Key)
	TimeID (Foreign Key)
	ProviderID (Foreign Key)
	SpecializationID (Foreign Key)
	AppointmentCount
	AverageSatisfactionRating
	AverageAppointmentDuration
	AverageWaitTime

In the Dimensional Normal Form (DNF), each attribute is stored in its own dimension table, facilitating efficient storage and retrieval of time-related data. In this design, there are three Dimension Tables: Time, Provider, and Specialization. The Time and Provider Dimensions were derived from the NRM displayed in Figure 2 on page 2, while the Specialization Dimension, and the attributes found in the Fact Table named "Provider Performance" in Figure 3 below were sourced from another database.

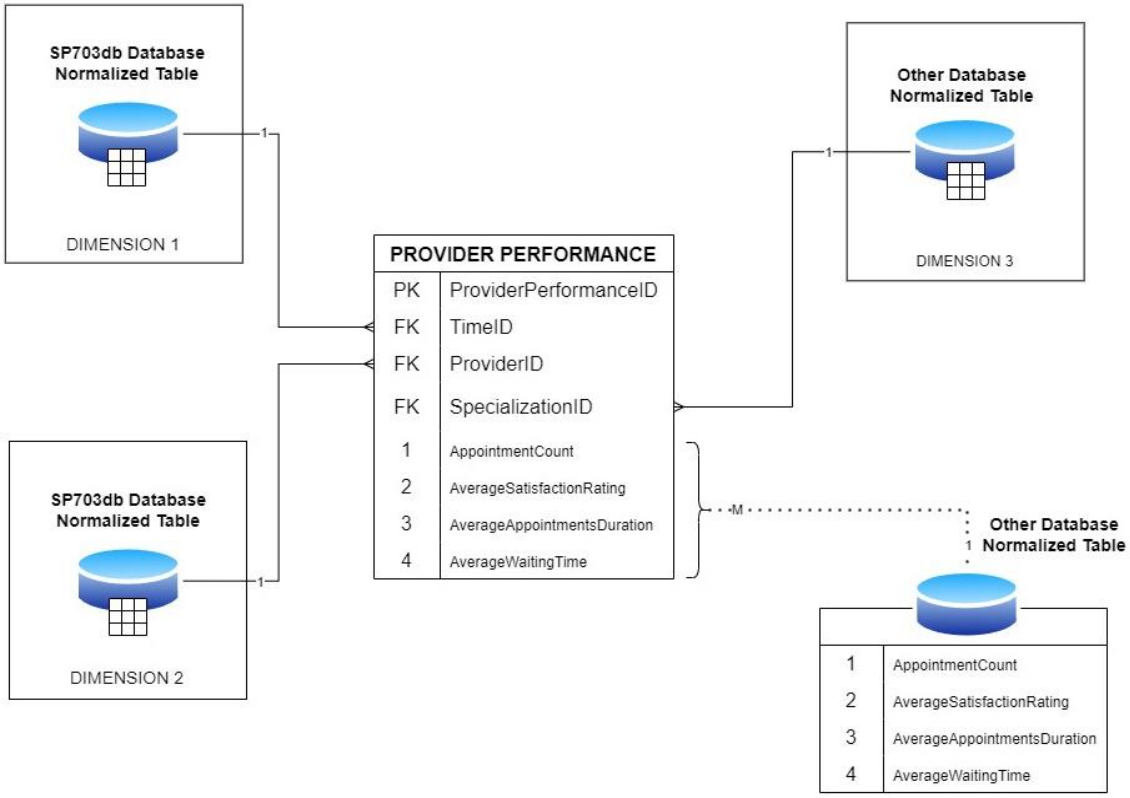

Figure 3. SP703dw Data Warehouse source databases

In Figure 3, we can see that Dimension 1 (Time) and Dimension 2 (Provider) were derived from the same database, named SP703db. On the other hand, Dimension 3 (Specialization) and the attributes within the Fact Table (Provider Performance) were sourced from a different database, as mentioned previously.

The Fact Table plays a critical role in a data warehousing system, as it provides a structured and efficient means of storing and analyzing data. By structuring dimension tables in this manner, the design enhances query performance for the Provider Performance fact table. See Figure 4 on page 4 for details.

For clarification, we will only create the database SP703db and use the Normalized Relational Model shown in Figure 2 on page 2 as our guide for the structure.

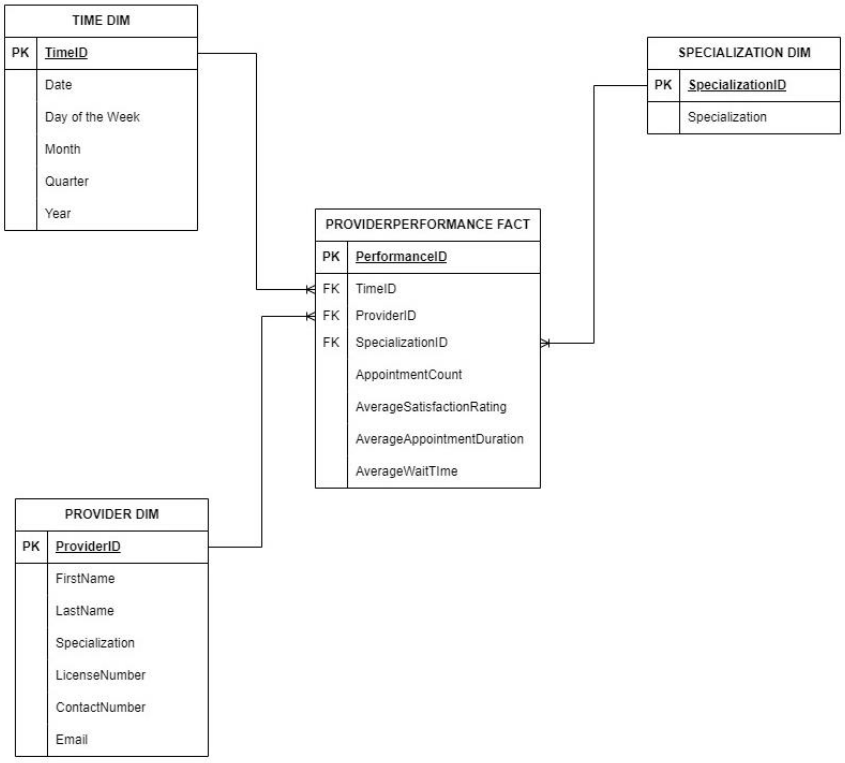

Figure 4. Dimensional Model (Fact & Dimensions table)

CREATING DATABASE

SQL script for Database in PostgreSQL (based on the NRM Figure 2 on page 2)

Query | Query History

```sql
-- Create the Patients table
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    DateOfBirth DATE,
    ContactNumber VARCHAR(20),
    Address VARCHAR(255)
);
```

```sql
-- Create the Providers table
CREATE TABLE Providers (
    ProviderID INT PRIMARY KEY,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    Specialization VARCHAR(255),
    LicenseNumber VARCHAR(255),
    ContactNumber VARCHAR(20),
    Email VARCHAR(255)
);
```

```sql
-- Create the Appointments table
CREATE TABLE Appointments (
    AppointmentID INT PRIMARY KEY,
    AppointmentDate DATE,
    AppointmentTime TIME,
    PatientID INT,
    ProviderID INT,
    AppointmentType VARCHAR(50),
    AppointmentStatus VARCHAR(50),
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (ProviderID) REFERENCES Providers(ProviderID)
);
```

```sql
-- Create the Procedures table
CREATE TABLE Procedures (
    ProcedureID INT PRIMARY KEY,
    ProcedureName VARCHAR(255),
    ProcedureDescription TEXT,
    ProcedureCost DECIMAL(10, 2)
);
```

```sql
-- Create the AppointmentProcedure junction table
CREATE TABLE AppointmentProcedure (
    AppointmentProcedureID INT PRIMARY KEY,
    AppointmentID INT,
    ProcedureID INT,
    FOREIGN KEY (AppointmentID) REFERENCES Appointments(AppointmentID),
    FOREIGN KEY (ProcedureID) REFERENCES Procedures(ProcedureID)
);
```

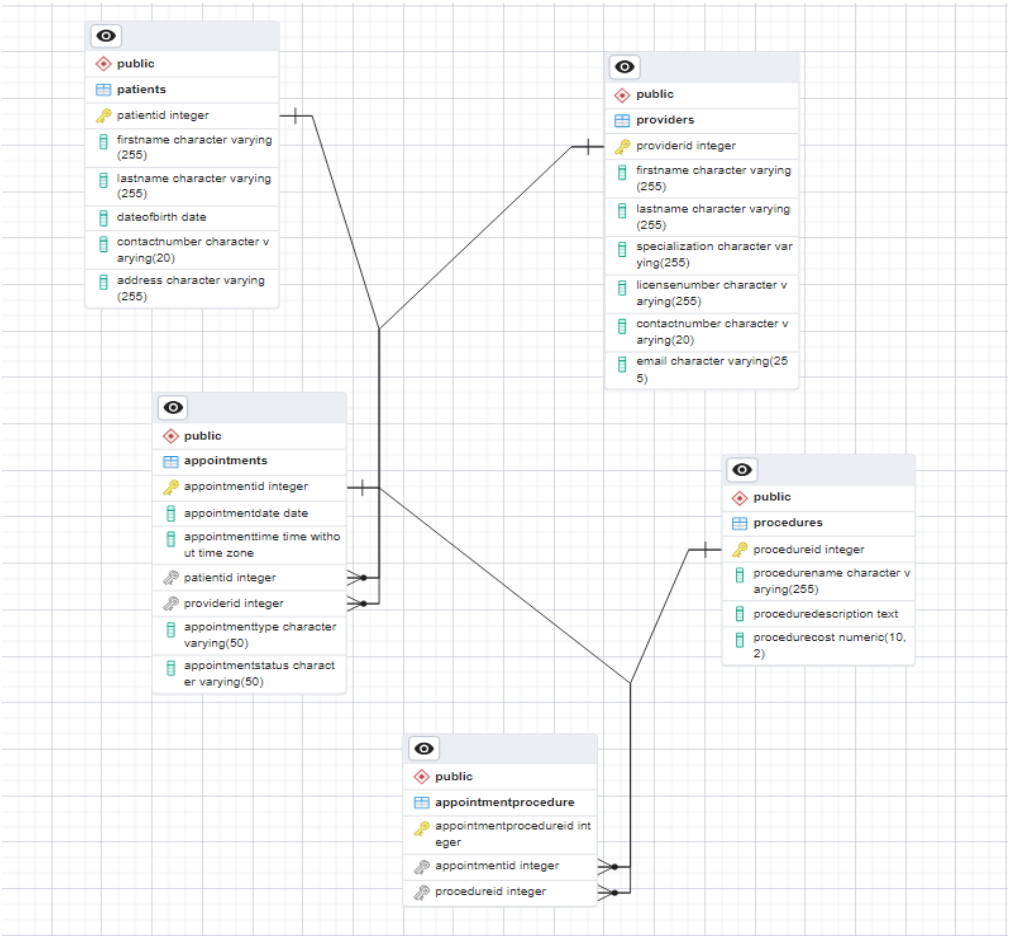

Figure 5. Database SP703db (PostgreSQL)

After creating the **database in PostgreSQL named "SP703db"** and creating the tables using the NRM, we will use Python + SQL **to load the dummy data** instead of using SQL only.

LOADING DUMMY DATA TO SP703db

SQL + Python (Jupyter Notebook)

Setting credentials to access database to establish a database connection.

```
PGHOST = 'localhost'
PGDATABASE = 'SP703db'
PGUSER = '*****' # hiding sensitive info
PGPASSWORD = '*****' # hiding sensitive info
```

```
def create_connect():
    dbconn = psycopg2.connect(
        dbname=PGDATABASE,
        user=PGUSER,
        password=PGPASSWORD,
        host=PGHOST,
        port='5432'
    )
    return dbconn
```

Loading Patients data into patients table in SP703db

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Insert data from csv file to Patient table in SP703 db
for item, row in patientdf.iterrows():
    cur.execute(
        '''INSERT INTO patients (PatientID, FirstName, LastName, DateOfBirth, ContactNumber, Address)
           VALUES (%s, %s, %s, %s, %s, %s) ON CONFLICT (PatientID) DO NOTHING''',
        (row['PatientID'], row['FirstName'], row['LastName'],
         row['DateOfBirth'], row['ContactNumber'], row['Address']))


# commit the changes
dbconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM patients')
patientdb_tbl = pd.DataFrame(cur.fetchall(), columns=patientdf.columns)

# Close the cursor and connection
cur.close()
dbconn.close()
```

SQL query in PostgreSQL

```
SELECT patientid, firstname, lastname, dateofbirth, contactnumber, address
    FROM public.patients;
```

Data Output | Messages | Notifications

	patientid [PK] integer	firstname character varying (255)	lastname character varying (255)	dateofbirth date	contactnumber character varying (20)	address character varying (255)
1	1	John	Doe	1985-05-15	123-456-7890	123 Main St
2	2	Jane	Smith	1990-08-20	987-654-3210	456 Elm St
3	3	David	Johnson	1998-03-10	555-555-5555	789 Oak Ave
4	4	Emily	Williams	1980-11-28	111-222-3333	321 Pine Rd
5	5	Michael	Brown	1975-07-12	444-777-8888	555 Cedar Ln
6	6	Susan	Anderson	1982-09-05	222-333-4444	789 Maple St
7	7	Robert	White	1993-04-30	666-999-1111	234 Birch Ave
8	8	Linda	Miller	1977-12-15	888-333-2222	567 Oak St
9	9	William	Moore	1989-06-22	444-666-7777	890 Pine Rd
10	10	Mary	Johnson	1987-03-08	777-888-9999	456 Elm St

Loading Providers data into Providers table in SP703db

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Insert data from csv file to Providers table in SP703 db
for item, row in providerdf.iterrows():
    cur.execute(
        '''INSERT INTO providers (ProviderID, FirstName, LastName, Specialization,
            LicenseNumber, ContactNumber, Email)
           VALUES (%s, %s, %s, %s, %s, %s, %s) ON CONFLICT (ProviderID) DO NOTHING''',
        (row['ProviderID'], row['FirstName'], row['LastName'],
         row['Specialization'], row['LicenseNumber'], row['ContactNumber'],
         row['Email']))


# commit the changes
dbconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM providers')
providerdb_tbl = pd.DataFrame(cur.fetchall(), columns=providerdf.columns)

# Close the cursor and connection
cur.close()
dbconn.close()
```

SQL query in PostgreSQL

```
SELECT providerid, firstname, lastname, specialization, licensenumber, contactnumber, email
    FROM public.providers;
```

Data Output | Messages | Notifications

	providerid [PK] integer	firstname character varying (255)	lastname character varying (255)	specialization character varying (255)	licensenumber character varying (255)	contactnumber character varying (20)	email character varying (255)
1	1	Dr. Sarah	Anderson	Cardiology	MD123456	555-123-7890	sarah.anderson@example.com
2	2	Dr. Robert	Clark	Orthopedics	MD789012	444-987-6543	robert.clark@example.com
3	3	Dr. Lisa	Smith	Internal Medicine	MD345678	777-333-2222	lisa.smith@example.com
4	4	Dr. John	Wilson	Neurology	MD234567	888-555-9999	john.wilson@example.com
5	5	Dr. Emily	Jones	Dermatology	MD567890	999-111-2222	emily.jones@example.com

Loading Appointments data into Appointments table in SP703db

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Insert data from csv file to Appointments table in SP703 db
for item, row in appointmentdf.iterrows():
    cur.execute(
        '''INSERT INTO appointments (AppointmentID, AppointmentDate, AppointmentTime, PatientID,
                                     ProviderID, AppointmentType, AppointmentStatus)
           VALUES (%s, %s, %s, %s, %s, %s, %s) ON CONFLICT (AppointmentID) DO NOTHING''',
        (row['AppointmentID'], row['AppointmentDate'], row['AppointmentTime'],
         row['PatientID'], row['ProviderID'], row['AppointmentType'],
         row['AppointmentStatus']))


# commit the changes
dbconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM appointments')
appointmentdb_tbl = pd.DataFrame(cur.fetchall(), columns=appointmentdf.columns)

# Close the cursor and connection
cur.close()
dbconn.close()
```

SQL query in PostgreSQL

Query | Query History

```
SELECT appointmentid, appointmentdate, appointmenttime, patientid, providerid, appointmenttype, appointmentstatus
    FROM public.appointments;
```

Data Output | Messages | Notifications

	appointmentid [PK] integer	appointmentdate date	appointmenttime time without time zone	patientid integer	providerid integer	appointmenttype character varying (50)	appointmentstatus character varying (50)
1	1	2023-10-10	09:00:00	1	1	General Cardiology Check-up	Scheduled
2	2	2023-10-11	10:30:00	2	2	Orthopedic Assessment	Confirmed
3	3	2023-10-12	14:15:00	3	1	Cardiology Check-up	Scheduled
4	4	2023-10-13	11:45:00	4	4	Neurological Evaluation	Confirmed
5	5	2023-10-14	08:30:00	5	5	Dermatological Examination	Scheduled
6	6	2023-10-15	13:30:00	6	2	Orthopedic Assessment	Confirmed
7	7	2023-10-16	10:15:00	7	5	Dermatological Examination	Scheduled
8	8	2023-10-17	15:45:00	8	1	Cardiology Check-up	Confirmed
9	9	2023-10-18	09:30:00	9	4	Neurological Evaluation	Scheduled
10	10	2023-10-19	11:00:00	10	3	Internal Medicine Check-up	Confirmed

Loading Procedure data into Procedure table in SP703db

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Insert data from csv file to Procedures table in SP703 db
for item, row in proceddf.iterrows():
    cur.execute(
        '''INSERT INTO procedures (ProcedureID, ProcedureName, ProcedureDescription,
                                   ProcedureCost)
           VALUES (%s, %s, %s, %s) ON CONFLICT (ProcedureID) DO NOTHING''',
        (row['ProcedureID'], row['ProcedureName'], row['ProcedureDescription'],
         row['ProcedureCost']))


# commit the changes
dbconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM procedures')
proceddb_tbl = pd.DataFrame(cur.fetchall(), columns=proceddf.columns)

# Close the cursor and connection
cur.close()
dbconn.close()
```

SQL query in PostgreSQL

```
SELECT procedureid, procedurename, proceduredescription, procedurecost
    FROM public.procedures;
```

Data Output | Messages | Notifications

	procedureid [PK] integer	procedurename character varying (255)	proceduredescription text	procedurecost numeric (10,2)
1	1	Cardiology Check-up	Comprehensive cardiology check-up including ECG and echocardiogram.	250.00
2	2	Orthopedics Consultation	Orthopedic assessment and consultation, including X-ray examination if needed.	150.00
3	3	Internal Medicine Check-up	Comprehensive internal medicine check-up and health assessment.	200.00
4	4	Neurology Consultation	Neurological evaluation, consultation, and diagnostic tests as required.	180.00
5	5	Dermatology Check-up	Dermatological examination, consultation, and minor skin procedures if necessa...	120.00

Loading AppointmentProcedure data into AppointmentProcedure table in SP703db

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Insert data from csv file to AppointmentProcedure table in SP703 db
for item, row in apptproceddf.iterrows():
    cur.execute(
        '''INSERT INTO appointmentprocedure (AppointmentProcedureID, AppointmentID, ProcedureID)
           VALUES (%s, %s, %s) ON CONFLICT (AppointmentProcedureID) DO NOTHING''',
        (row['AppointmentProcedureID'], row['AppointmentID'], row['ProcedureID']))


# commit the changes
dbconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM appointmentprocedure')
apptproceddb_tbl = pd.DataFrame(cur.fetchall(), columns=apptproceddf.columns)

# Close the cursor and connection
cur.close()
dbconn.close()
```

SQL query in PostgreSQL

```
SELECT appointmentprocedureid, appointmentid, procedureid
    FROM public.appointmentprocedure;
```

Data Output | Messages | Notifications

	appointmentprocedureid [PK] integer	appointmentid integer	procedureid integer
1	1	1	1
2	2	2	2
3	3	3	1
4	4	4	4
5	5	5	5
6	6	6	2
7	7	7	5
8	8	8	1
9	9	9	4
10	10	10	3

CREATING DATA WAREHOUSE

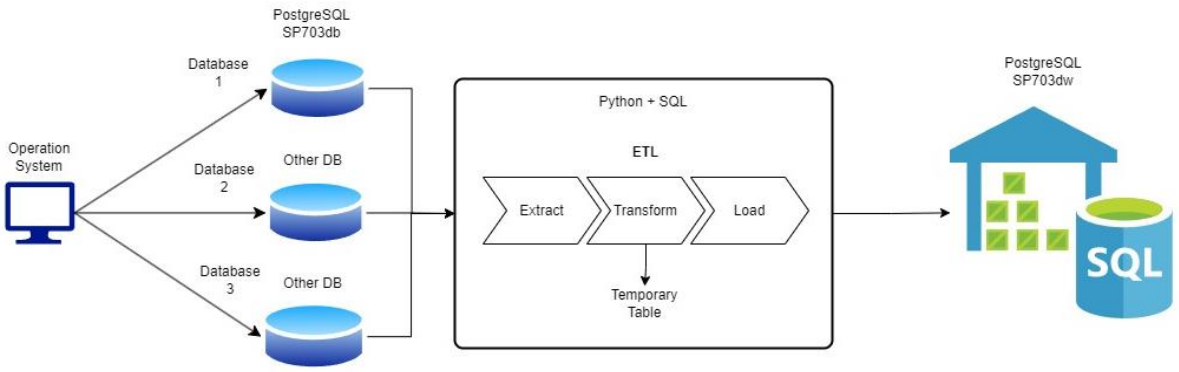

Figure 6. Data warehouse architecture

This is the diagram of the data warehouse architecture for the proposed system. As discussed on page 2 - 3, the Fact table (Provider Performance) which is in the SP703dw data warehouse will get its dimensions and data from 3 databases.

SQL script for Data warehouse in PostgreSQL (Based on the Dimensional Model Figure 4 on page 4)

Query | Query History

```
-- Create Dimension Tables
CREATE TABLE Time_dim (
    TimeID INT NOT NULL PRIMARY KEY,
    Date DATE,
    DayOfWeek INT,
    Month INT,
    Quarter INT,
    Year INT
);
```

```
CREATE TABLE Provider_dim (
    ProviderID INT NOT NULL PRIMARY KEY,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    Specialization VARCHAR(255),
    LicenseNumber VARCHAR(255),
    ContactNumber VARCHAR(20),
    Email VARCHAR(255)
);
```

```sql
CREATE TABLE Specialization_dim (
    SpecializationID INT NOT NULL PRIMARY KEY,
    Specialization VARCHAR(255)
);
-- Create Fact Table
CREATE TABLE ProviderPerformance_fact (
    PerformanceID INT PRIMARY KEY,
    TimeID INT,
    ProviderID INT,
    SpecializationID INT,
    AppointmentCount INT,
    AverageSatisfactionRating DECIMAL(5, 2),
    AverageAppointmentDuration DECIMAL(5, 2),
    AverageWaitTime DECIMAL(5, 2),
    FOREIGN KEY (TimeID) REFERENCES Time(TimeID),
    FOREIGN KEY (ProviderID) REFERENCES Provider(ProviderID),
    FOREIGN KEY (SpecializationID) REFERENCES Specialization(SpecializationID)
);
```

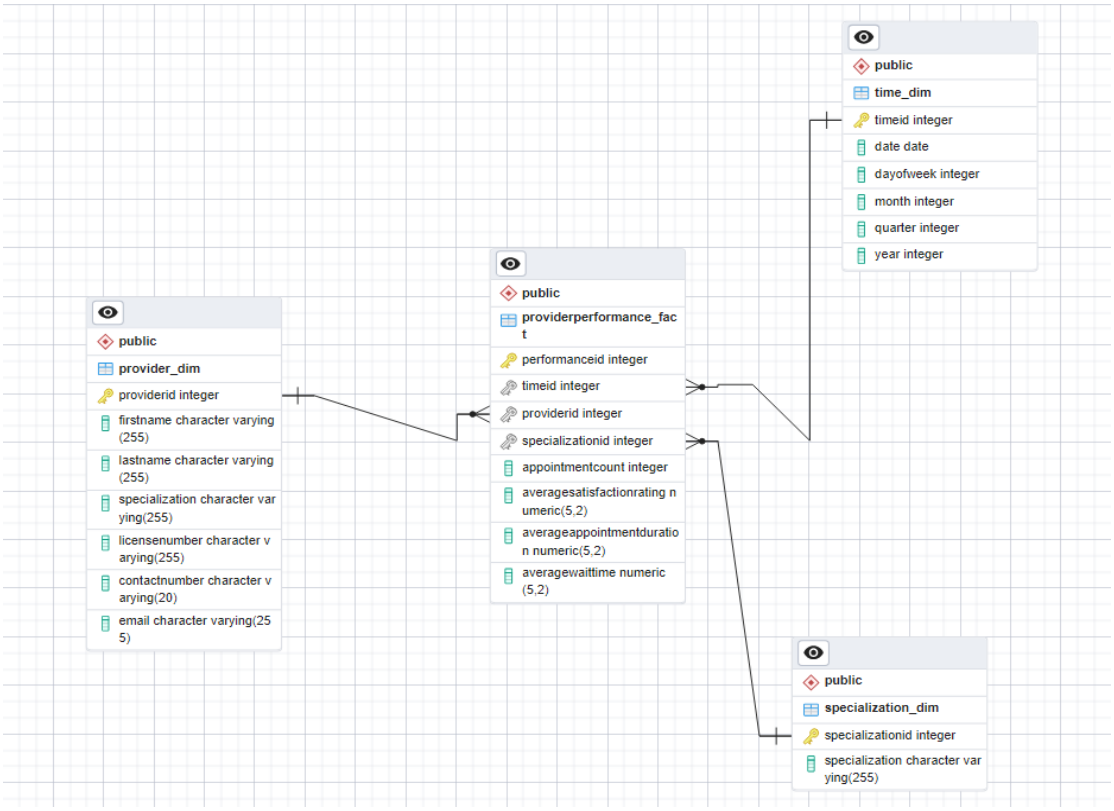

Figure 7. Data warehouse SP703dw (PostgreSQL)

Next is we need to extract, then transform the data from our database SP703db to ensure it aligns with the format and structure required before loading it to the SP703dw data warehouse. See Figure 8 on page 11.

EXTRACTING DATA FROM SP703db DATABASE

Python + SQL script

Providers data from SP703db database

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Execute SQL queries to retrieved data
cur.execute('SELECT * FROM providers')
providerdf = pd.DataFrame(cur.fetchall(), columns=providerdf.columns)

# Close the cursor and connection
cur.close()
dbconn.close()
```

Result from fetching data from database SP703db

This data doesn't need transformation as it has the same structure in the SP703dw data warehouse.

```
providerdf
```

	ProviderID	FirstName	LastName	Specialization	LicenseNumber	ContactNumber	Email
0	1	Dr. Sarah	Anderson	Cardiology	MD123456	555-123-7890	sarah.anderson@example.com
1	2	Dr. Robert	Clark	Orthopedics	MD789012	444-987-6543	robert.clark@example.com
2	3	Dr. Lisa	Smith	Internal Medicine	MD345678	777-333-2222	lisa.smith@example.com
3	4	Dr. John	Wilson	Neurology	MD234567	888-555-9999	john.wilson@example.com
4	5	Dr. Emily	Jones	Dermatology	MD567890	999-111-2222	emily.jones@example.com

Appointments data from SP703db database

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# Execute SQL queries to retrieved data
cur.execute('SELECT appointmentid, appointmentdate FROM appointments')
appointdf = pd.DataFrame(cur.fetchall(), columns=appointmentdb_tbl.columns[:2])

# Close the cursor and connection
cur.close()
dbconn.close()
```

Result from fetching data from database SP703db.

Next is we need to manipulate this data during transformation stage to meet the required structure before loading it to SP703dw data warehouse

```
# result from fetching data from database SP703db
appointdf
```

	AppointmentID	AppointmentDate	Date	DayOfWeek	Month	Quarter	Year
0	1	2023-10-10	10	Tuesday	October	4	2023
1	2	2023-10-11	11	Wednesday	October	4	2023
2	3	2023-10-12	12	Thursday	October	4	2023
3	4	2023-10-13	13	Friday	October	4	2023
4	5	2023-10-14	14	Saturday	October	4	2023
5	6	2023-10-15	15	Sunday	October	4	2023
6	7	2023-10-16	16	Monday	October	4	2023
7	8	2023-10-17	17	Tuesday	October	4	2023
8	9	2023-10-18	18	Wednesday	October	4	2023
9	10	2023-10-19	19	Thursday	October	4	2023

Specialization data from "other database"

Note: We will just assume that this data comes from "other database" within the medical clinic, this has already been discussed in the documentation figure 8 page 11

```
# this is a list of specialization by the providers, but it will add more if the provider does have two specialization.
# but for now, we will just assume a 1:1 ratio.
spec = pd.read_excel('Specializationdata.xlsx')
spec
```

	SpecializationID	Specialization
0	1	Cardiology
1	2	Orthopedics
2	3	Internal Medicine
3	4	Neurology
4	5	Dermatology

Fact Table data from "other database" and SP703db

Note: We will just assume that this data comes from "other database" within the medical clinic, this has already been discussed in the documentation figure 8 page 11

```
spec_dict = {}

# storing the providers id and specialization into a dictionary
for i, r in providerdf.iterrows():
    spec_dict[r['Specialization']] = r['ProviderID']

spec_dict
```

```
{'Cardiology': 1,
 'Orthopedics': 2,
 'Internal Medicine': 3,
 'Neurology': 4,
 'Dermatology': 5}
```

```
# data for AppointmentCount, AverageSatisfactionRating, AverageAppointmentDuration, and AverageWaitingTime
mixdata = pd.read_excel('Ratingsetcdata.xlsx')
mixdata
```

	AppointmentCount	AverageSatisfactionRating	AverageAppointmentDuration	AverageWaitingTime
0	1	4.6	30	5
1	1	4.4	40	6
2	1	4.2	25	4
3	1	4.1	45	3
4	1	4.8	35	5
5	1	4.5	38	6
6	1	4.7	42	1
7	1	4.3	28	10
8	1	4.0	50	5
9	1	4.9	33	6

Note: Code block below will extract data comes from the SP703db database not from the "other database"

```
#creating connection
dbconn = create_connect()

# Create a cursor
cur = dbconn.cursor()

# new column names for extracted data
cols = ['TimeID', 'ProviderID']

# Execute SQL queries to retrieved data
cur.execute('SELECT appointmentid, providerid FROM appointments')
patxprovdf = pd.DataFrame(cur.fetchall(), columns=cols)

# Close the cursor and connection
cur.close()
dbconn.close()
```

```
patxprovdf
```

	TimeID	ProviderID
0	1	1
1	2	2
2	3	1
3	4	4
4	5	5
5	6	2
6	7	5
7	8	1
8	9	4
9	10	3

TRANSFORMING DATA FROM SP703db DATABASE

Next is we need to transform some of the data based on the structure of Dimension and Fact tables in SP703dw Datawarehouse. Transforming the Appointment table.

```
# Convert 'AppointmentDate' column to datetime
appointdf['AppointmentDate'] = pd.to_datetime(appointdf['AppointmentDate'])

# Split the 'AppointmentDate' column into separate columns
appointdf['Date'] = appointdf['AppointmentDate'].dt.date
appointdf['DayOfWeek'] = appointdf['AppointmentDate'].dt.day
appointdf['Month'] = appointdf['AppointmentDate'].dt.month
appointdf['Quarter'] = appointdf['AppointmentDate'].dt.quarter
appointdf['Year'] = appointdf['AppointmentDate'].dt.year

# dropping unnecessary column
trfndappointdf = appointdf.drop(columns='AppointmentDate')
trfndappointdf
```

	AppointmentID	Date	DayOfWeek	Month	Quarter	Year
0	1	2023-10-10	10	10	4	2023
1	2	2023-10-11	11	10	4	2023
2	3	2023-10-12	12	10	4	2023
3	4	2023-10-13	13	10	4	2023
4	5	2023-10-14	14	10	4	2023
5	6	2023-10-15	15	10	4	2023
6	7	2023-10-16	16	10	4	2023
7	8	2023-10-17	17	10	4	2023
8	9	2023-10-18	18	10	4	2023
9	10	2023-10-19	19	10	4	2023

Merging the data from SP703 database and "other database" for the Provider Performance Fact Table

```
# Add a new column 'Specialization' based on 'ProviderID'
patxprovdf['SpecializationID'] = patxprovdf['ProviderID'].replace(spec_dict)
```

```
factdw_tbl = pd.concat([patxprovdf, mixdata], axis=1)
factdw_tbl
```

	TimeID	ProviderID	SpecializationID	AppointmentCount	AverageSatisfactionRating	AverageAppointmentDuration	AverageWaitingTime
0	1	1	1	1	4.6	30	5
1	2	2	2	1	4.4	40	6
2	3	1	1	1	4.2	25	4
3	4	4	4	1	4.1	45	3
4	5	5	5	1	4.8	35	5
5	6	2	2	1	4.5	38	6
6	7	5	5	1	4.7	42	1
7	8	1	1	1	4.3	28	10
8	9	4	4	1	4.0	50	5
9	10	3	3	1	4.9	33	6

LOADING DATA TO SP70dw DATA WAREHOUSE

We will load the transformed data into SP703dw Datawarehouse but first we need to connect to PostgreSQL to access the data warehouse.

Setting credentials to access data warehouse to establish a data warehouse connection.

```
PGHOST = 'localhost'
PGDATAWAREHOUSE = 'SP703dw'
PGUSER = '******' # need to hide sensitive info
PGPASSWORD = '******' # need to hide sensitive info
```

```
def create_connectdw():
    dwconn = psycopg2.connect(
        dbname=PGDATAWAREHOUSE,
        user=PGUSER,
        password=PGPASSWORD,
        host=PGHOST,
        port='5432'
    )
    return dwconn
```

Load the Providers data from SP702 database to Providers Dimension in SP703dw data warehouse.

```
#creating connection
dwconn = create_connectdw()

# Create a cursor
cur = dwconn.cursor()

# Insert data from providers table into SP703 dw provider_dim
for item, row in providerdf.iterrows():
    cur.execute(
        '''INSERT INTO provider_dim (ProviderID, FirstName, LastName,
                                     Specialization, LicenseNumber, ContactNumber, Email)
           VALUES (%s, %s, %s, %s, %s, %s, %s) ON CONFLICT (ProviderID) DO NOTHING''',
        (row['ProviderID'], row['FirstName'], row['LastName'],
         row['Specialization'], row['LicenseNumber'], row['ContactNumber'],
         row['Email']))


# commit the changes
dwconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM provider_dim')
providerdw_tbl = pd.DataFrame(cur.fetchall(), columns=providerdf.columns)

# Close the cursor and connection
cur.close()
dwconn.close()
```

SQL query in PostgreSQL

Query | Query History

```
SELECT providerid, firstname, lastname, specialization, licensenumber, contactnumber, email
    FROM public.provider_dim;
```

Data Output | Messages | Notifications

	providerid [PK] integer	firstname character varying (255)	lastname character varying (255)	specialization character varying (255)	licensenumber character varying (255)	contactnumber character varying (20)	email character varying (255)
1	1	Dr. Sarah	Anderson	Cardiology	MD123456	555-123-7890	sarah.anderson@example.com
2	2	Dr. Robert	Clark	Orthopedics	MD789012	444-987-6543	robert.clark@example.com
3	3	Dr. Lisa	Smith	Internal Medicine	MD345678	777-333-2222	lisa.smith@example.com
4	4	Dr. John	Wilson	Neurology	MD234567	888-555-9999	john.wilson@example.com
5	5	Dr. Emily	Jones	Dermatology	MD567890	999-111-2222	emily.jones@example.com

Load the transformed Appointment data from SP702 database to Time Dimension in SP703dw data warehouse

```
#creating connection
dwconn = create_connectdw()

# Create a cursor
cur = dwconn.cursor()

# Insert data from transformed Appointment table into SP703 dw time_dim
for item, row in trfndappointdf.iterrows():
    cur.execute(
        '''INSERT INTO time_dim (TimeID, Date, DayOfWeek, Month, Quarter, Year)
           VALUES (%s, %s, %s, %s, %s, %s) ON CONFLICT (TimeID) DO NOTHING''',
        (row['AppointmentID'], row['Date'], row['DayOfWeek'],
         row['Month'], row['Quarter'], row['Year']))


# commit the changes
dwconn.commit()

#we need to declare new columns
cols = ['TimeID', 'Date', 'DayOfWeek', 'Month', 'Quarter', 'Year']

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM time_dim')
timedw_tbl = pd.DataFrame(cur.fetchall(), columns=cols)

# Close the cursor and connection
cur.close()
dwconn.close()
```

SQL query in PostgreSQL

Query | Query History

```
1  SELECT timeid, date, dayofweek, month, quarter, year
2      FROM public.time_dim;
```

Data Output | Messages | Notifications

	timeid [PK] integer	date date	dayofweek integer	month integer	quarter integer	year integer
1	1	2023-10-10	10	10	4	2023
2	2	2023-10-11	11	10	4	2023
3	3	2023-10-12	12	10	4	2023
4	4	2023-10-13	13	10	4	2023
5	5	2023-10-14	14	10	4	2023
6	6	2023-10-15	15	10	4	2023
7	7	2023-10-16	16	10	4	2023
8	8	2023-10-17	17	10	4	2023
9	9	2023-10-18	18	10	4	2023
10	10	2023-10-19	19	10	4	2023

Load the Specialization data from "other database" to Specialization Dimension in SP703dw data warehouse

```
#creating connection
dwconn = create_connectdw()

# Create a cursor
cur = dwconn.cursor()

# Insert data from "other database" table in SP703 dw
for item, row in spec.iterrows():
    cur.execute(
        '''INSERT INTO specialization_dim (SpecializationID, Specialization)
           VALUES (%s, %s) ON CONFLICT (SpecializationID) DO NOTHING''',
        (row['SpecializationID'], row['Specialization']))


# commit the changes
dwconn.commit()

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM specialization_dim')
specdw_tbl = pd.DataFrame(cur.fetchall(), columns=spec.columns)

# Close the cursor and connection
cur.close()
dwconn.close()
```

SQL query in PostgreSQL

Query | Query History

```
1  SELECT specializationid, specialization
2      FROM public.specialization_dim;
```

Data Output | Messages | Notifications

	specializationid [PK] integer	specialization character varying (255)
1	1	Cardiology
2	2	Orthopedics
3	3	Internal Medicine
4	4	Neurology
5	5	Dermatology

Load the data from SP703db database to Provider Performance Fact in SP703dw data warehouse.

```
#creating connection
dwconn = create_connectdw()

# Create a cursor
cur = dwconn.cursor()

# getting the latest key
latest_key = 0
cur.execute('SELECT MAX(performanceid) AS key FROM providerperformance_fact')
get_key = pd.DataFrame(cur.fetchall(), columns=['key'])
if get_key['key'].item() == None:
    latest_key = 0
else:
    latest_key = get_key['key'].item()


# Insert data from other database + SP703db table into providerperformance fact table in SP703dw
for item, row in factdw_tbl.iterrows():
    latest_key+=1
    cur.execute(
        '''INSERT INTO providerperformance_fact (PerformanceID, TimeID, ProviderID, SpecializationID,
           AppointmentCount, AverageSatisfactionRating, AverageAppointmentDuration, AverageWaitTime)
           VALUES (%s, %s, %s, %s, %s, %s, %s, %s) ON CONFLICT (PerformanceID) DO NOTHING''',
        (latest_key, row['TimeID'], row['ProviderID'], row['SpecializationID'],
         row['AppointmentCount'], row['AverageSatisfactionRating'],
        row['AverageAppointmentDuration'], row['AverageWaitingTime']))

# commit the changes
dwconn.commit()

cols = ['PerformanceID', 'TimeID', 'ProviderID', 'SpecializationID',
        'AppointmentCount', 'AverageSatisfactionRating', 'AverageAppointmentDuration', 'AverageWaitTime']

# Execute SQL queries to check if we successfully inserted the data
cur.execute('SELECT * FROM providerperformance_fact')
specdw_tbl = pd.DataFrame(cur.fetchall(), columns=cols)

# Close the cursor and connection
cur.close()
dwconn.close()
```

SQL query in PostgreSQL

Query | Query History

```
SELECT performanceid, timeid, providerid, specializationid, appointmentcount, averagesatisfactionrating,
       averageappointmentduration,
       averagewaittime
    FROM public.providerperformance_fact;
```

Data Output | Messages | Notifications

	performanceid [PK] integer	timeid integer	providerid integer	specializationid integer	appointmentcount integer	averagesatisfactionrating numeric (5,2)	averageappointmentduration numeric (5,2)	averagewaittime numeric (5,2)
1	1	1	1	1	1	4.60	30.00	5.00
2	2	2	2	2	1	4.40	40.00	6.00
3	3	3	1	1	1	4.20	25.00	4.00
4	4	4	4	4	1	4.10	45.00	3.00
5	5	5	5	5	1	4.80	35.00	5.00
6	6	6	2	2	1	4.50	38.00	6.00
7	7	7	5	5	1	4.70	42.00	1.00
8	8	8	1	1	1	4.30	28.00	10.00
9	9	9	4	4	1	4.00	50.00	5.00
10	10	10	3	3	1	4.90	33.00	6.00

After loading all of the data to datawarehouse SP703dw, we will do some test queries.

Query | Query History

```
-- sample query in the datawarehouse SP703dw

SELECT  prov.firstname,
        ROUND(AVG(averagesatisfactionrating), 2) AS Ratings,
        spec.specialization
FROM public.providerperformance_fact per
    LEFT JOIN public.provider_dim prov ON per.providerid = prov.providerid
    LEFT JOIN public.specialization_dim spec ON per.specializationid = spec.specializationid
GROUP BY prov.firstname, spec.specialization;
```

Data Output | Messages | Notifications

	firstname character varying (255)	ratings numeric	specialization character varying (255)
1	Dr. Emily	4.75	Dermatology
2	Dr. John	4.05	Neurology
3	Dr. Lisa	4.90	Internal Medicine
4	Dr. Robert	4.45	Orthopedics
5	Dr. Sarah	4.37	Cardiology

The result table above is achieved by joining the 2 dimension table with the fact table