

Hospital Appointments App

Author: Ihar Makeyenka

Group: 22-LR-JS1

Subject: Databases Course work

OLTP

[OLTP Schema](#)

[OLTP Overview](#)

[Data generation using python:](#)

[Loading generated data to database using sql script:](#)

OLAP

[OLAP Schema](#)

[OLAP Overview](#)

ETL

[What It Does](#)

[Key Points to Notice](#)

Instructions

[Chapter 1: Which tools I used](#)

[Chapter 2: Loading data](#)

[Chapter 3: OLAP and ETL](#)

[Chapter 4. Insight queries](#)

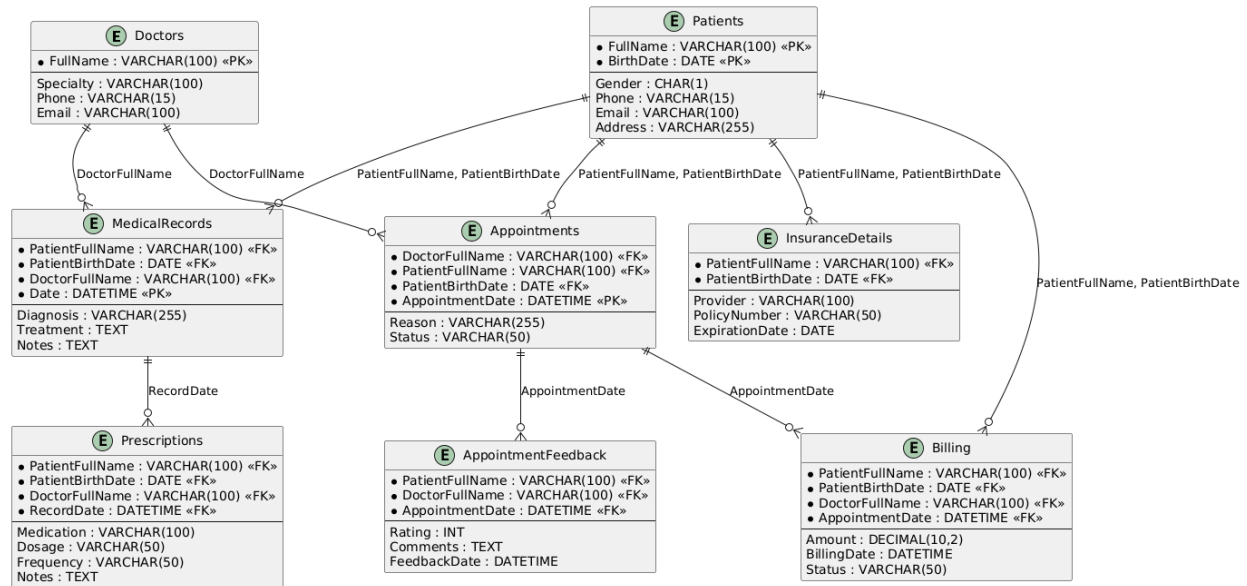
[Chapter 5. Power BI](#)

OLTP

This OLTP solution tracks the day-to-day operations of a healthcare setting—such as doctor assignments, patient appointments, medical records, and billing—so that information remains both accurate and immediately available for transactional updates.

I decided to use composite keys just to simplify ETL and make OLAP more visual and useful. I don't think that is suitable for "normal" business use-cases, but for course work, why not, still usable.

OLTP Schema



OLTP Overview

1. Tables

- **Doctors:** Tracks each doctor's full name, specialty, phone, and email.
- **Patients:** Stores patient details (full name, birthdate, gender, contact info).
- **Appointments:** Links doctors to patients, capturing appointment date, reason, and status.
- **MedicalRecords:** Logs diagnoses, treatments, and notes per patient visit.
- **Billing:** Records payment amounts, billing dates, and statuses.
- **InsuranceDetails:** Holds insurance provider data (policy number, expiration).

2. Relations

- **Doctors → Appointments:** One doctor can have many appointments.
- **Patients → Appointments:** One patient can schedule many appointments.
- **Patients → MedicalRecords:** One patient can have multiple records.
- **Appointments → Billing:** Billing references the same date and participants as each appointment.

3. Insights

- **Operational Lookups:** "Which patient is seeing which doctor today?"
- **Quick Stats:** "How many upcoming appointments this week?" or "How many unpaid bills?"
- **Immediate Updates:** Staff can check the latest patient contact info, appointment times, and billing status instantly.

4. Benefits

- **Real-Time Access:** Always up-to-date info for scheduling, billing, and patient status.
- **Accurate Linking:** Minimizes duplication (one record per patient, one record per doctor).
- **Faster Service:** Quick retrieval of who owes payments, who has which appointment, etc.
- **Foundation for Analytics:** Structured relationships make it easy to feed data into an OLAP system for deeper analysis.

Data generation using python:

The Python script generates realistic data for an OLTP system involving tables for doctors, patients, appointments, medical records, billing, prescriptions, insurance details, and appointment feedback. It uses the **Faker** library to generate random but realistic names, dates, addresses, diagnoses, treatments, and other data fields. The script writes this generated data into CSV files for each table, ready to be used in a database.

Loading generated data to database using sql script:

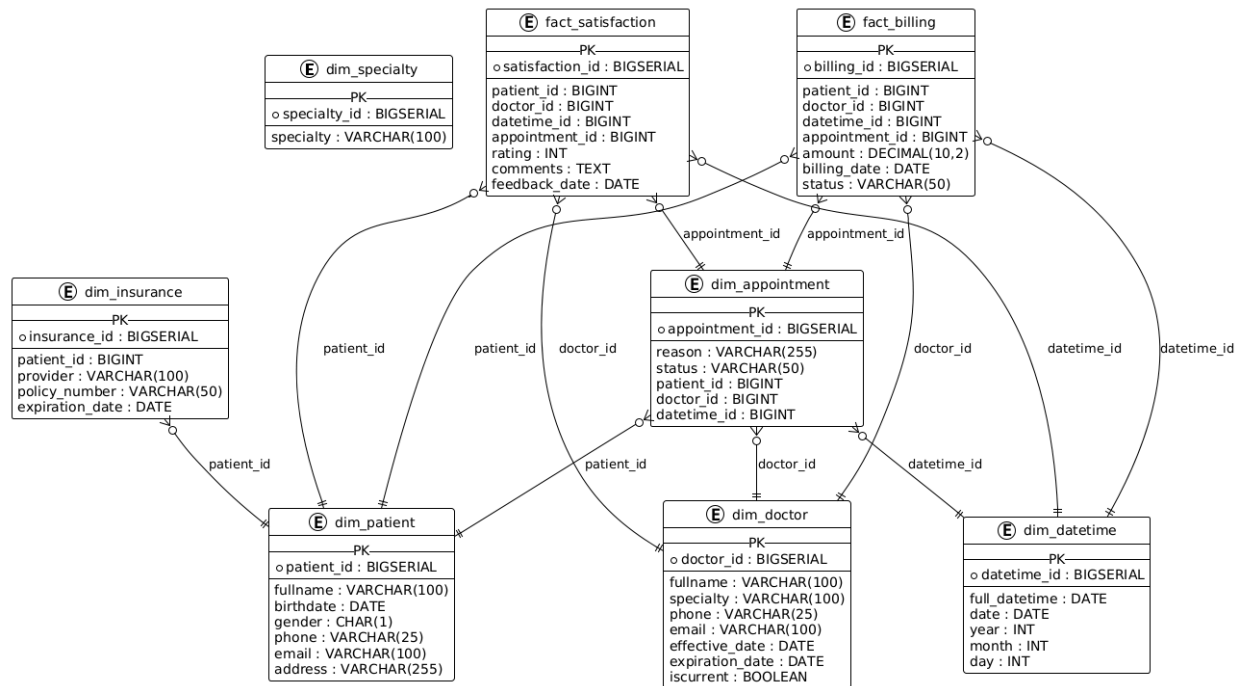
The SQL script is designed to load the data from the generated CSV files into PostgreSQL tables. It first creates the necessary tables (Doctors, Patients, Appointments, Medical Records, etc.) if they don't already exist. Then, it loads the data from the CSV files into temporary tables and inserts the records into the main tables, ensuring that no duplicates are added. This is done by checking if a record already exists before inserting it. Finally, the temporary tables are cleaned up after the data is successfully loaded.

This combination of Python and SQL scripts helps automate the data generation and loading process, ensuring that the database can be populated with realistic data without overwriting any existing records.

OLAP

This OLAP solution is designed to support advanced analytics and reporting in a healthcare setting by consolidating transactional data into structured dimensions and facts. It enables efficient trend analysis, revenue tracking, and patient satisfaction insights while preserving historical data for in-depth evaluation over time.

OLAP Schema



OLAP Overview

1. Tables

- **dim_doctor**: Tracks doctors' historical data (full name, specialty, phone, email), with SCD Type 2 fields like effective/expiration dates.
- **dim_patient**: Stores patient attributes (name, birthdate, gender, contact) to provide consistent patient lookups.
- **dim_datetime**: Holds date-only info (year, month, day) to group or filter results by date.
- **dim_appointment**: Captures appointment reason, status, plus links to patients, doctors, and timestamps.
- **dim_insurance**: Contains insurance data (policy, provider) linked to each patient.
- **dim_specialty**: Manages unique specialty names.
- **fact_billing**: Aggregates financial transactions (amount, billing date) with references to patient, doctor, appointment, and date.
- **fact_satisfaction**: Records patient feedback (rating, comments) tied to patient, doctor, appointment, and date.

2. Relations

- **Fact Tables → Dimension Tables:** For example, fact_billing links to dim_patient, dim_doctor, dim_datetime, and dim_appointment. This allows slicing revenue by doctor, date, or appointment type.
- **dim_appointment → dim_patient/dim_doctor/dim_datetime:** Tracks which patient saw which doctor on which date.
- **dim_insurance → dim_patient:** Ties an insurance entry to the correct patient ID.

3. How to Get Insights

- **Aggregations:** Summarize amounts in fact_billing by year/month, or see how satisfaction ratings vary by specialty.
- **Trends Over Time:** Use dim_datetime to compare revenue or feedback across quarters or years.
- **Slicing and Dicing:** Filter metrics by doctor, datetime or any dimension field

ETL

Now the tough part. This ETL process takes raw data from the OLTP system and transforms it into a structured format for the OLAP database, making it ready for analysis. It handles everything from cleaning messy data to linking tables together so we can pull insights like revenue trends or patient feedback ratings.

What It Does

1. Extract

- Grabs data from the OLTP tables (like doctors, patients, and appointments) and puts it into staging tables.
- This is the raw data dump—it's messy and needs cleaning.

2. Transform

- **Clean-Up:** Filters out invalid or outdated data (e.g., appointments with missing patient names, old billing records).
- **Linking:** Matches data across tables (e.g., linking billing info to the correct appointment).
- **Track Changes:** For things like doctor details, it keeps track of updates (SCD Type 2) so we don't lose history when something changes.
- **Standardize:** Fixes inconsistencies, like making sure all dates are formatted properly.
- **Add IDs:** Creates useful keys like appointment_id to simplify relationships in the OLAP tables.

3. Load

- **Dimensions:** Fills in the descriptive tables like `dim_doctor`, `dim_patient`, and `dim_datetime` to give context to facts.
- **Facts:** Populates tables like `fact_billing` (for revenue) and `fact_satisfaction` (for feedback). These tables hold the numbers we care about.

Key Points to Notice

- **Data Validation:** Messy data from the OLTP is cleaned up in staging—bad entries are filtered out so only valid data makes it to the warehouse.
- **History Tracking:** Changes to things like a doctor’s specialty or phone number are saved in `dim_doctor`, so we can see the full timeline.
- **Relationships:** Fact tables are tied to dimension tables (e.g., billing records are linked to patients, doctors, and appointment dates). This makes it easy to slice data in any way.
- **Appointment Linking:** Since OLTP uses composite keys, the process generates a clean `appointment_id` for the OLAP system, simplifying analysis.
- **Staging Cleanup:** Once data is loaded into the warehouse, the staging tables are cleared to keep things tidy.

Instructions

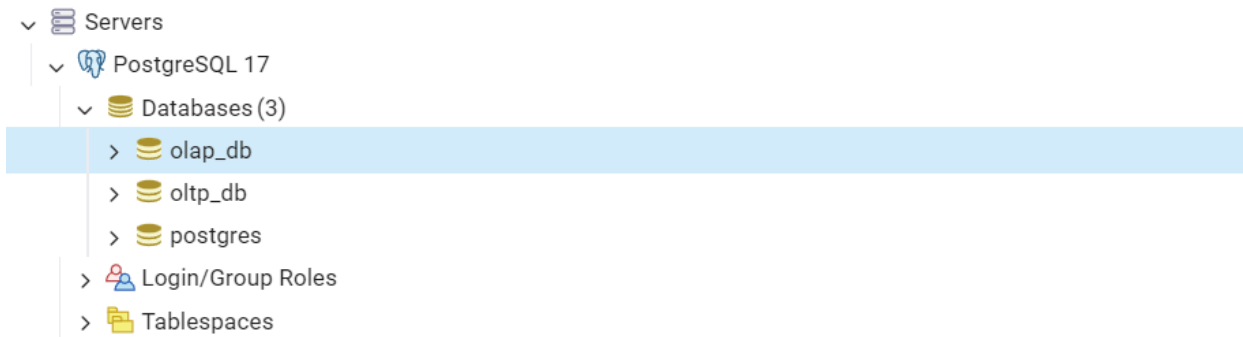
Chapter 1: Which tools I used

Repository link: <https://github.com/4llower/hospital-OLTP-OLAP-ETL>

| Purpose | Tool |
|-------------------------|---|
| Database management | pgAdmin4 8.14 |
| Database | PostgreSQL 17 |
| Data generation | Python3, pandas, faker |
| Analytics visualization | PowerBI 2.138.1452.0 64-bit (November 2024) |

Chapter 2: Loading data

1. I created two empty databases in pgAdmin4:

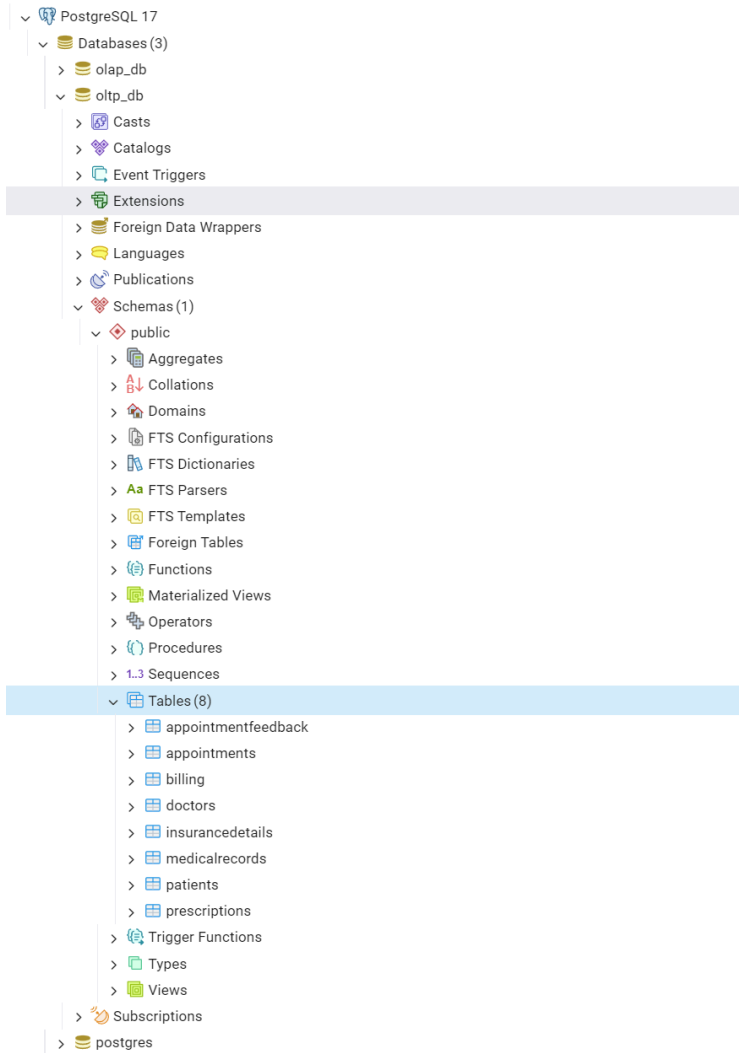


2. Pull the repository <https://github.com/4llower/hospital-OLTP-OLAP-ETL>

3. Let's open oltp_db query tool and paste script (load_data_csv) that placed in my repository by path: **./oltp/data/load_data_csv.sql**.

4. And change all paths **D:/EHU/Databases/coursework/oltp/data/generated/** to path where you pull the repository on your computer.

After executing the script data should be loaded from csv and you can see tables in oltp_db public schema:



Check some tables if they consist the data, if some tables doesn't contain a data please

verify the path from step 3:

Query Query History

1 SELECT * FROM public.doctors
2 ORDER BY doctorid ASC

Data Output Messages Notifications

| | doctorid [PK] integer | fullname character varying (100) | specialty character varying (100) | phone character varying (25) | email character varying (100) |
|----|--------------------------|-------------------------------------|--------------------------------------|---------------------------------|----------------------------------|
| 1 | 1 | Robert Smith | Dermatology | 6859604662 | grayjennifer@example.net |
| 2 | 2 | Jasmine Henry | Neurology | 585-731-0641 | lauragilmore@example.net |
| 3 | 3 | Lisa Garcia | Neurology | +1-864-895-5831x3428 | icox@example.net |
| 4 | 4 | Barbara Bailey | Pediatrics | 735.426.5565 | ryan88@example.org |
| 5 | 5 | Jonathan Foster | Cardiology | 511-509-8788 | sototrevor@example.org |
| 6 | 6 | Jonathan Thomas | Neurology | (268)508-7232 | savannah30@example.org |
| 7 | 7 | Samantha Molina | Neurology | +1-572-766-7806x94580 | morgan54@example.net |
| 8 | 8 | Cody Neal | Dermatology | 348-481-6701x9145 | cjohnson@example.org |
| 9 | 9 | John Paul | Cardiology | 884.997.0700x76935 | ilee@example.org |
| 10 | 10 | Jonathan Ross | Orthopedics | (223)748-8260x1225 | pdavenport@example.net |
| 11 | 11 | Lori Brown | Dermatology | +1-511-793-4911 | johnsonrachel@example.net |
| 12 | 12 | Bradley Martin | Neurology | +1-813-738-6607 | morgansandy@example.com |
| 13 | 13 | Teresa Ingram | Pediatrics | 792-906-7967x250 | aharrington@example.com |
| 14 | 14 | Mary Nunez | Cardiology | +1-361-975-5523x7029 | villanuevaelizabeth@example.net |

Chapter 3: OLAP and ETL

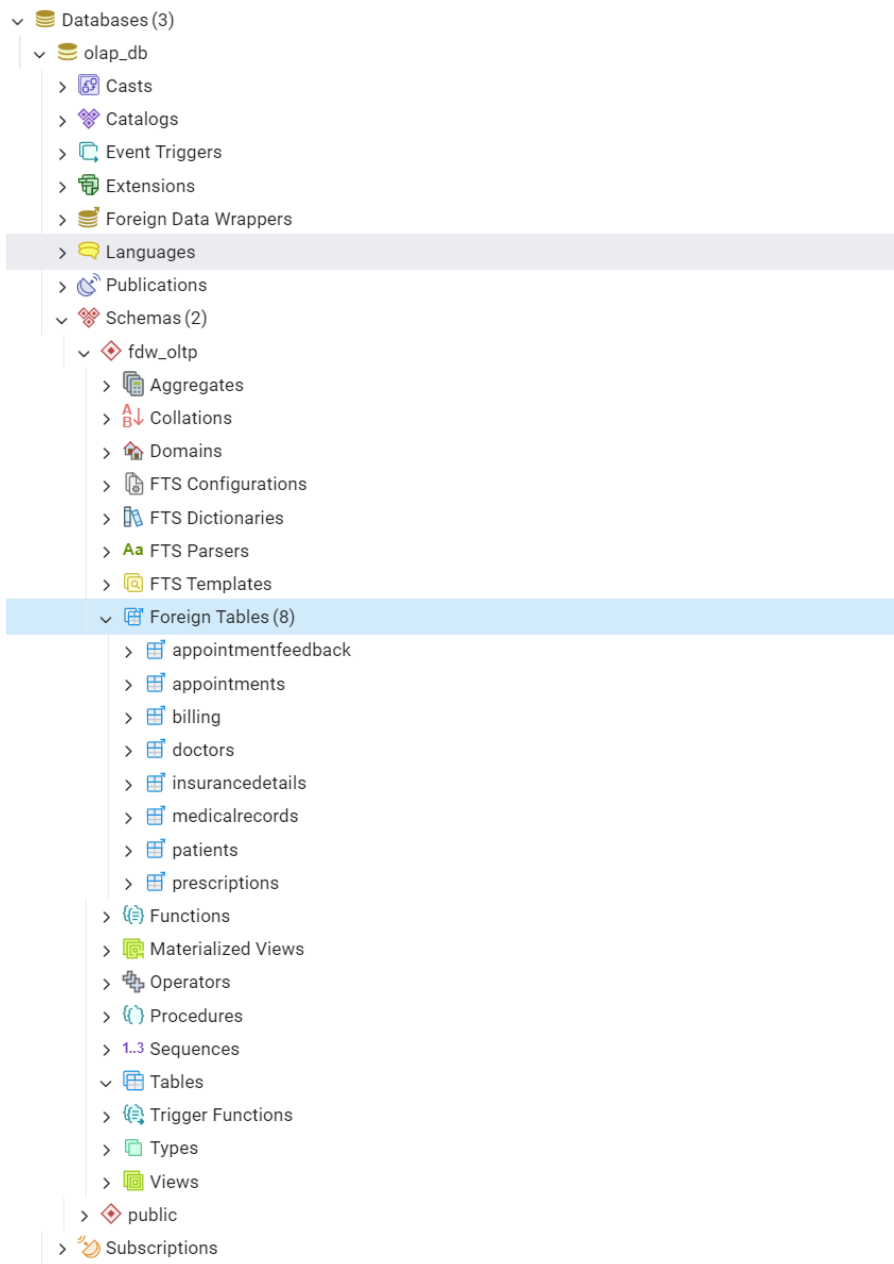
1. I used foreign data wrapper to have links for tables from OLTP. That's why firstly in olap_db we need to initialize the foreign data wrapper to make everything work.

Script for fdw placed in **./olap/init_foreign_data_wrapper.sql**

Paste the script in **olap_db** query editor. If you have other credentialis than just postgres/postgres use them :)

After executing the script you should see new schema in oltp_db that will contain a lot of foreign

table:

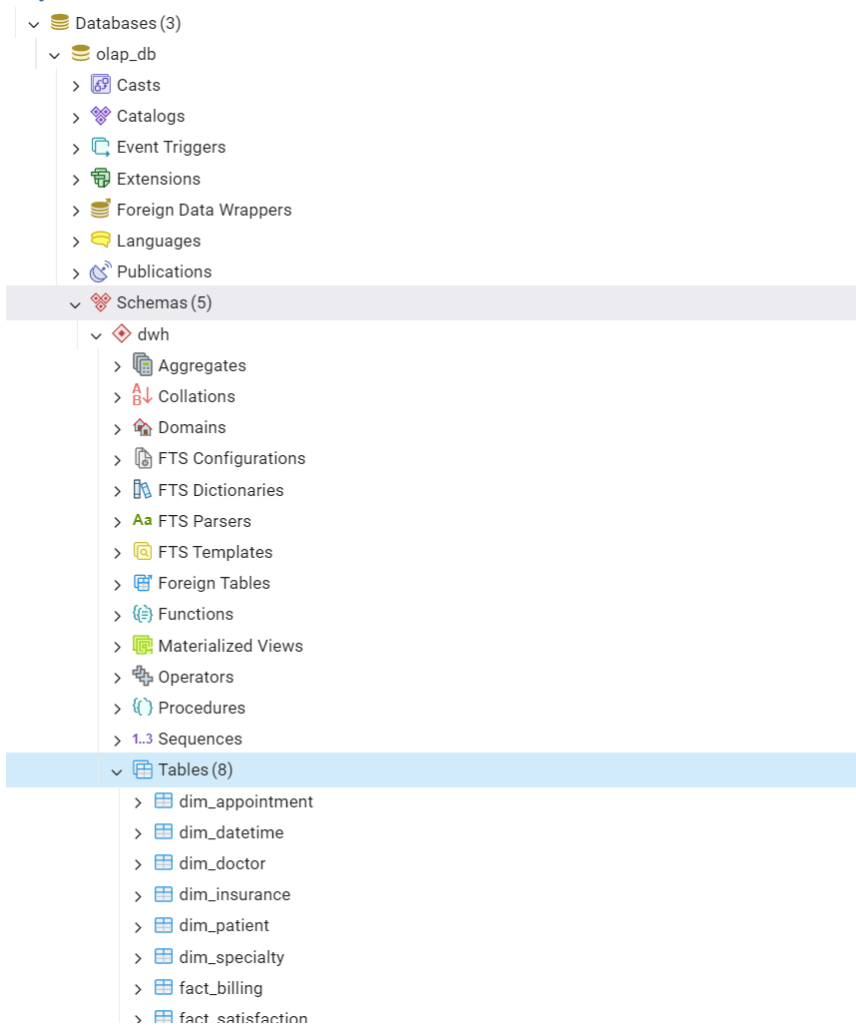


2. LET'S FINALLY EXECUTE THE ETL MONSTER SCRIPT THAT WILL DO ALL THE JOB, I really proud of myself that it works, it took a lot of time. Maybe it's not perfect -> but it's my child :)

It's placed **./olap/etl.sql**

Paste this script to olap_db query editor. After executing the script you should see a lot of new dimension/facts table in dwh schema. Also in etl script I created a reference schema with table that contain medical speciality that allowed for dwh, it's just formal point to prove that we CAN FILTER if we want :)

You should see something like this:



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the Object Explorer shows the database structure for 'PostgreSQL 17'. Under 'Databases (3)', 'olap_db' is expanded, and within its 'Schemas (5)', the 'dw' schema is selected. The 'Tables (8)' list under 'dw' includes 'fact_billing', which is highlighted. The main pane shows a query window titled 'dw.fact_billing/olap_db/postgres@PostgreSQL 17'. The query is:

```
1 SELECT * FROM dw.fact_billing
2 ORDER BY billing_id ASC
```

Below the query window, the 'Data Output' tab is active, showing a table with 14 rows and 9 columns. The columns are: billing_id (PK) bigint, patient_id bigint, doctor_id bigint, datetime_id bigint, appointment_id bigint, amount numeric (10,2), billing_date date, and status character varying (50). The data is sorted by billing_id in ascending order.

| | billing_id (PK) bigint | patient_id bigint | doctor_id bigint | datetime_id bigint | appointment_id bigint | amount numeric (10,2) | billing_date date | status character varying (50) |
|----|------------------------|-------------------|------------------|--------------------|-----------------------|-----------------------|-------------------|-------------------------------|
| 1 | 1 | 42 | 95 | 1 | 87 | 390.02 | 2025-01-02 | Overdue |
| 2 | 2 | 13 | 90 | 1 | 62 | 321.35 | 2025-01-02 | Paid |
| 3 | 3 | 65 | 52 | 1 | 76 | 117.05 | 2025-01-02 | Paid |
| 4 | 4 | 88 | 33 | 1 | 98 | 442.39 | 2025-01-02 | Paid |
| 5 | 5 | 79 | 24 | 1 | 6 | 134.46 | 2025-01-02 | Paid |
| 6 | 6 | 49 | 17 | 1 | 77 | 248.34 | 2025-01-02 | Overdue |
| 7 | 7 | 65 | 16 | 1 | 34 | 111.89 | 2025-01-02 | Overdue |
| 8 | 8 | 52 | 15 | 1 | 23 | 299.77 | 2025-01-02 | Paid |
| 9 | 9 | 79 | 98 | 2 | 24 | 380.67 | 2025-01-07 | Pending |
| 10 | 10 | 22 | 76 | 2 | 18 | 330.63 | 2025-01-07 | Paid |
| 11 | 11 | 54 | 70 | 2 | 41 | 104.31 | 2025-01-07 | Pending |
| 12 | 12 | 51 | 37 | 2 | 55 | 123.96 | 2025-01-07 | Pending |
| 13 | 13 | 5 | 25 | 2 | 14 | 371.50 | 2025-01-07 | Pending |
| 14 | 14 | 88 | 4 | 2 | 59 | 241.38 | 2025-01-07 | Paid |

Chapter 4. Insight queries

You need to paste queries to oltp_db in olap_db depending on the name of query.

1. `./insights/oltp_1.sql (oltp_db)`

Query

Query History

1

-- Patients with the Most Appointments in the Last 3 Months

2

▼

SELECT

3

a.PatientFullName AS patient_name,

4

COUNT(*) AS total_appointments

5

FROM Appointments a

6

WHERE a.AppointmentDate >= (CURRENT_DATE - INTERVAL '3 months')

7

GROUP BY a.PatientFullName

8

ORDER BY COUNT(*) DESC

9

LIMIT 5;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

| | patient_name character varying (100) 🔒 | total_appointments bigint 🔒 |
|---|---|--------------------------------|
| 1 | Margaret Trevino | 5 |
| 2 | Leslie Franklin | 5 |
| 3 | Kimberly Rodriguez | 4 |
| 4 | Margaret Wood | 3 |
| 5 | Eric Chan Jr. | 3 |

2. `./insights/oltp_2.sql` (oltp_db query)

Trust me if it will be more data with data more that current_timestamp there will be data

;))

The screenshot shows a SQL query tool interface. The top section is titled "Query" and "Query History". The query text is as follows:

```
1  -- Upcoming Appointments by Doctor, no rows because no data after current timestamp -> but it's working query trust me :)
2  SELECT
3      d.FullName AS doctor_name,
4      a.AppointmentDate,
5      a.Reason,
6      a.Status
7  FROM Appointments a
8  JOIN Doctors d
9      ON d.FullName = a.DoctorFullName
10 WHERE a.AppointmentDate >= CURRENT_TIMESTAMP
11 ORDER BY a.AppointmentDate ASC;
```

The bottom section is titled "Data Output", "Messages", and "Notifications". Below this is a toolbar with icons for various actions. The output schema is displayed in a table:

| doctor_name | appointmentdate | reason | status |
|-------------------------|-----------------|-------------------------|------------------------|
| character varying (100) | date | character varying (255) | character varying (50) |

1. ./insights/olap_1.sql (olap_db query tool)

olap_db/postgres@PostgreSQL 17* x

olap_db/postgres@PostgreSQL 17

Query Query History

```
1  -- Total Billing Amount by Doctor (Year to Date)
2  SELECT
3      dd.fullname AS doctor_name,
4      SUM(fb.amount) AS total_billing
5  FROM dwh.fact_billing fb
6  JOIN dwh.dim_doctor dd
7      ON dd.doctor_id = fb.doctor_id
8  JOIN dwh.dim_datetime dt
9      ON dt.datetime_id = fb.datetime_id
10 WHERE dt.year = EXTRACT(YEAR FROM CURRENT_DATE)
11 GROUP BY dd.fullname
12 ORDER BY SUM(fb.amount) DESC;
13
```

Data Output Messages Notifications

| | doctor_name character varying (100) | total_billing numeric |
|----|--|--------------------------|
| 1 | Tanya Schmidt | 1314.51 |
| 2 | Barbara Bailey | 1157.24 |
| 3 | Allison Owens | 1073.81 |
| 4 | Samuel Ryan | 1024.68 |
| 5 | Heather Nguyen | 987.39 |
| 6 | April Stewart | 970.38 |
| 7 | Veronica Galloway | 963.25 |
| 8 | Oscar Richardson | 934.15 |
| 9 | Christopher Cruz | 892.95 |
| 10 | Colleen Torres | 870.56 |
| 11 | Laurie Martinez | 852.36 |
| 12 | Amber Rodriguez | 846.00 |
| 13 | Denise Sandoval | 845.05 |
| 14 | Harry Hall | 807.45 |

Total rows: 46 Query complete 00:00:00.134

2. ./insights/olap_2.sql (olap_db query tool)

The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'olap_db/postgres@PostgreSQL 17*'. The query editor contains the following SQL code:

```
1  -- Average Satisfaction Rating by Specialty
2  SELECT
3      s.specialty AS specialty,
4      ROUND(AVG(fs.rating), 2) AS avg_rating
5  FROM dwh.fact_satisfaction fs
6  JOIN dwh.dim_doctor dd
7      ON dd.doctor_id = fs.doctor_id
8  JOIN dwh.dim_specialty s
9      ON s.specialty = dd.specialty
10 GROUP BY s.specialty
11 ORDER BY avg_rating DESC;
12
```

Below the query editor, the 'Data Output' tab is active, displaying the results of the query in a table format:

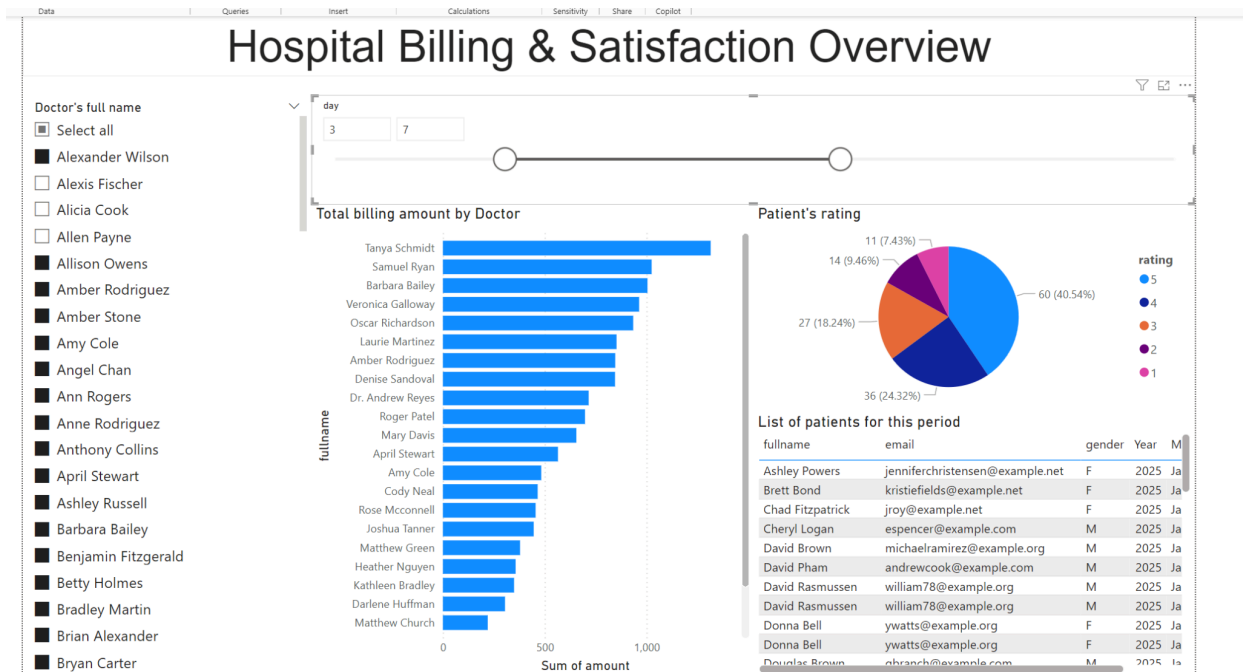
| | specialty character varying (100) | avg_rating numeric |
|---|--------------------------------------|-----------------------|
| 1 | Dermatology | 3.29 |
| 2 | Neurology | 3.22 |
| 3 | Cardiology | 2.92 |
| 4 | Pediatrics | 2.88 |
| 5 | Orthopedics | 2.69 |

Chapter 5. Power BI

It was surprising for me that PostgreSQL integration with powerBI works so well. But at moment when I started to do PowerBI I had a lot of connections to my db, and it wasn't work. But I just removed connections to olap_db and it was fixed. I just imported everything and started to work with visualizations practically immediately.

Open the report that placed: `"/powerbi.pbix"`

You will see something like this:



You can use slicers to check analytical data for doctors:

1. Which doctors earned more money
2. Compare doctors (amount of money billed and also satisfaction of patients)
3. You can see patients for this period (if you add email or other information you can contact them for example and ask more information for the doctor)

Date slicer is shitty because when I was generating the data I didn't check that it generated only for 13 days. But anyway it can be configured easy and changed. I just showed how it can be in mvp :)

Thank you! That's it