

## Part 1 — Dimension Model + Approach

### Dimensional Model

- **Dim\_Patient** (Patient\_Key, First\_Name, Last\_Name, Date\_of\_Birth)

Key = natural patient\_id.

- **Dim\_Doctor** (Doctor\_Key, Doctor\_Name, Specialty)

Key = natural doctor\_id.

- **Dim\_Date** (Date\_Key, Full\_Date, Year, Month, Day)

Key = YYYYMMDD.

- **Dim\_Diagnosis** (Diagnosis\_Key, Diagnosis\_Code, Diagnosis\_Description)

Key = natural DiagnosisCode (one row/code).

- **Fact\_Visit** (Visit\_Key, Patient\_Key, Doctor\_Key, Date\_Key, Amount)

Keys are natural (visit\_id → Visit\_Key).

- **Bridge\_Visit\_Diagnosis** (Visit\_Key, Diagnosis\_Key, Diagnosis\_Sequence, Is\_Primary)

Resolves a many-to-many between visits and diagnoses; preserves order and primary flag.

### Business requirements

1. “For a given patient, on a given date, for a doctor, list all diagnoses”

Join Fact\_Visit → Bridge\_Visit\_Diagnosis → Dim\_Diagnosis filtered by Patient/Date/Doctor.

The **bridge** allows multiple diagnoses per visit in sequence with a primary marker.

2. “How many patients were diagnosed with a given diagnosis\_code?”

Count distinct Patient\_Key by Diagnosis\_Key using Bridge\_Visit\_Diagnosis → Fact\_Visit.

## Part 2 – Delta Live Tables Pipeline & ETL Implementation

### Hospital Visit Analysis using Databricks DLT

#### Dataset Used

We used the **Synthea Synthetic Electronic Health Records (EHR)** dataset, which contains realistic patient, provider, and encounter data.

The following CSV files were uploaded into a Databricks **Volume**:

- patients.csv – basic patient demographics
- providers.csv – doctor information (name, specialty)
- encounters.csv – visit records with encounter type and cost
- conditions.csv – diagnosis codes and descriptions

All files were stored under  
/Volumes/hospital\_cg/visit-check/raw.

#### Overall Architecture

The project was implemented as a three-layer Delta Live Tables ( DLT ) pipeline plus a Gold SQL layer for analytics.

Layer	Purpose	Storage
<b>Bronze</b>	Ingest raw CSV data using Auto Loader and DLT streaming	/Volumes/hospital_cg/visit-check/raw
<b>Silver</b>	Create cleaned and standardized views with proper data types	SQL Views (silver_*_v)
<b>Gold</b>	Build analytical tables (Dimensions + Fact + Bridge)	SQL Tables (dim_*, fact_visit, bridge_visit_diagnosis)

### 3. Implementation Steps

#### Step 1 – Bronze Layer Creation

- Created a **DLT Python notebook** (DLT\_Hospital\_Pipeline) that uses dlt.table() to load all CSV files as streaming Delta tables.
- Each table (e.g., bronze\_patients, bronze\_providers, bronze\_encounters, bronze\_conditions) was ingested from the volume path using Auto Loader.

## Step 2 – Silver Layer (Views)

- Created SQL views to clean and standardize the Bronze data.
- Converted IDs to string for consistent joins.
- Extracted key columns for each entity:
  - **Patients:** Id, First, Last, Birthdate
  - **Doctors:** Id, Name, Specialty
  - **Encounters:** Id, Patient, Provider, Start, Cost, Code, Description
  - **Visit-Diagnosis:** Linked each visit with diagnosis codes and generated sequence + primary flag.

## Step 3 – Gold Layer (Dimensional Model)

Built six analytical tables using CREATE OR REPLACE TABLE:

1. **dim\_date** – dates extracted from encounter and diagnosis timestamps
2. **dim\_patient** – patient dimension using natural key patient\_id
3. **dim\_doctor** – doctor dimension using natural key doctor\_id
4. **dim\_diagnosis** – unique diagnosis codes with names and descriptions
5. **fact\_visit** – main fact table recording each visit, doctor, patient, date, and amount
6. **bridge\_visit\_diagnosis** – many-to-many bridge between visits and diagnoses with diagnosis\_sequence and is\_primary

This follows a **Star Schema + Bridge Table** design.

## Step 4 – Pipeline Job Setup

Created a Databricks **Job with 2 tasks:**

1. **Task 1 – Bronze Ingestion:** Runs the DLT notebook.
2. **Task 2 – Gold Build:** Runs the SQL notebook containing Gold table definitions.

**Dependency:** Task 2 depends on Task 1.

So, each time the Job runs,

- Bronze is updated.
- Silver views always query the current data.
- But Gold tables must be rebuilt periodically.

## Conclusion

- This project demonstrated a complete ETL workflow in Databricks using Delta Live Tables.
- The Bronze layer handles continuous ingestion.
- Silver views standardize and clean data.
- Gold tables implement a star schema with a bridge for multi-diagnosis visits.
- The pipeline is fully automated and supports incremental updates whenever new records are inserted into the Bronze tables.