

|  |
| --- |
| 01000741752 |
| Smart Phone |
| Omar123915@hotmail.com |
| Envelope |
| Omar Hossam Attia |
| User |

[GITHUB REPOSITORY](https://github.com/OmarAttia95/FHIR_ETL_Project/blob/main/airflow_dag_workflow.py)

**SQL Schema for Structured Tables and Business & Technical Metadata**

Dear,

This SQL schema is designed to manage healthcare-related data, focusing on patients, their encounters, medications, conditions, diagnostic reports, and claims. The schema is designed in 3NF (Third Normal Form) for optimal storage, data integrity, and to avoid redundancy. It is also structured as a Data Warehouse (DWH), meaning it is optimized for reporting and analysis.

**Technical Metadata**:

**1. Database Normalization**

This database is designed to be in **3NF** (Third Normal Form) to ensure the following:

**Elimination of Redundant Data**: Redundant data is minimized by separating the data into smaller, related tables.

**Data Integrity**: Ensures minimal data duplication and avoids inconsistency by splitting tables and using foreign keys.

**Data Update** Efficiency: By normalizing, updates are easier and faster because changes are made in one place, not across multiple rows.

**Ease of Querying**: The 3NF design allows for more efficient data retrieval, especially for complex queries where multiple relationships are involved.

**2. Primary Key and Foreign Key Constraints**

**Primary Keys**: Each table has a unique identifier, known as the Primary Key. This ensures that every row in a table can be uniquely identified.

**Example:** PATIENT\_ID in the patients table.

**Foreign Keys**: These are used to establish relationships between tables. A foreign key in one table links to the primary key of another table. This ensures referential integrity.

**Example**: The PATIENT\_ID in the medications, claims, encounters, and other tables acts as a foreign key that links back to the patients table.

3. **Data Types**

VARCHAR (Variable Character Length): Used for textual data such as names, IDs, and address details. Ensures storage efficiency by only allocating the space needed for the actual data.

Example: PATIENT\_ID VARCHAR(255), FIRST\_NAME VARCHAR(255).

DATE & TIMESTAMP: These types store date and time information, essential for tracking events such as BIRTH\_DATE, ENCOUNTER\_DATE, and CLAIM\_START\_DATE.

Example: BIRTH\_DATE DATE, START\_DATETIME TIMESTAMP.

DECIMAL: Used for financial figures such as costs and prices. This ensures that calculations with high precision, such as medical costs or claim costs, are handled correctly.

Example: COST DECIMAL.

TEXT: Used for larger strings such as PATIENT\_ADDRESS or LOCATION where the length is uncertain or potentially very large.

**4. Indexing**

Indexes improve query performance by allowing the database to quickly find and retrieve data without scanning entire tables. Indexes are created on foreign keys and other frequently queried columns.

Example Indexes:

CREATE INDEX idx\_patient\_id ON patients(PATIENT\_ID);

CREATE INDEX idx\_claims\_patient\_id ON claims(PATIENT\_ID);

These indexes speed up common queries like retrieving all claims for a specific patient, improving performance especially in large datasets.

**5. Fact and Dimension Tables**

Fact Tables: These tables hold measurable, quantitative data and often include keys from dimension tables.

Example: fact\_patient\_metrics table stores aggregated metrics (like TOTAL\_CLAIM\_COST and TOTAL\_ENCOUNTER\_COST) for each patient, with references to the PATIENT\_ID.

Dimension Tables: These tables hold descriptive attributes about the business entities. In this case, the patients, medications, claims, encounters, and others are dimension tables that describe patients, medications, claims, etc.

**6. Surrogate Keys**

The fact\_patient\_metrics table uses surrogate keys (i.e., METRIC\_ID as an auto-incremented primary key) to uniquely identify rows, decoupling business identifiers from database keys. This allows for easier management and performance improvement.

7. **PATIENT\_ID as a Natural Key**

In this database schema, the PATIENT\_ID serves as a natural key for the patients table and other related tables. A **natural key** is a unique, real-world identifier that is used to uniquely identify records in a table without requiring a separate artificially created key. Here's why **PATIENT\_ID** qualifies as a natural key:

**Uniqueness:**

The PATIENT\_ID is unique for each patient in the database, making it a natural candidate for serving as a unique identifier for patient-related data across multiple tables**.**

**Real-world meaning:**

PATIENT\_ID is meaningful in the context of the healthcare domain. It corresponds to a unique identifier for patients in the real world (for example, a government-issued health ID or a system-generated identifier used within a hospital's record-keeping system).

**Referential Integrity:**

The PATIENT\_ID is used as a foreign key in other tables like medications, claims, encounters, and diagnostic\_reports. By using a natural key, we preserve the referential integrity of the relationships between tables, ensuring that each reference points to a valid patient record.

**Simplification:**

Using PATIENT\_ID directly avoids the need to create a surrogate key (like an auto-incrementing ID) for the patients table. This simplifies data management and querying, as the PATIENT\_ID is both unique and human-readable.

**Efficient Join Operations:**

As a natural key, PATIENT\_ID can efficiently join tables across the database, such as linking patients with claims, medications, encounters, and more, without needing to reference an additional surrogate key.

**Business Metadata**:

* **Patients Table**: Captures essential demographic and contact information about patients. It is used for identifying patients in all other records across the system.
* **Medications Table**: Tracks prescriptions, which is crucial for monitoring treatment and medication adherence.
* **Claims Table**: Tracks insurance claims and is important for the financial aspects of patient care. It allows healthcare providers and insurers to track the cost of treatments and claims.
* **Encounters Table**: Provides details on patient visits to healthcare facilities, essential for both clinical care and billing purposes.
* **Diagnostic Reports and Conditions Tables**: These are critical for diagnosing conditions, determining treatment plans, and tracking medical conditions over time.
* **Patient History Table**: Used for auditing and maintaining patient records over time, capturing changes in patient details.
* **Fact Patient Metrics Table**: Aggregates key metrics about patient care, helping stakeholders analyze treatment costs and outcomes.

**Data Warehouse (DWH) Considerations:**

* **The fact table** (fact\_patient\_metrics) and dimension tables (e.g., patients, medications, claims, etc.) follow a star schema design, which is common in data warehousing.
* **This schema** allows for efficient querying of patient-related metrics, such as total claim costs, total medications, and encounter counts, which are aggregated in the fact table.
* **This design** is optimized for analytical queries, allowing users to quickly obtain key insights into patient care and associated costs.

Here are some sample SQL queries that you can use to verify and interact with the database.

**Verify Patient Information**

**“””**SELECT \* FROM patients WHERE PATIENT\_ID = '5d45e219-8b8b-5298-3646-a50b0e3743b3';”””

**Retrieve All Claims for a Patient**

**“””**SELECT \* FROM claims WHERE PATIENT\_ID = '5d45e219-8b8b-5298-3646-a50b0e3743b3';”””

**Get Metrics for a Patient**

**“””**SELECT \* FROM fact\_patient\_metrics WHERE PATIENT\_ID ='5d45e219-8b8b-5298-3646-a50b0e3743b3';”””

**Total Claim Costs for a Patient**

“””SELECT SUM(COST) AS total\_claim\_cost FROM claims WHERE PATIENT\_ID ='5d45e219-8b8b-5298-3646-a50b0e3743b3';”””

**List All Conditions Diagnosed in the Last 30 Days**

**“””**SELECT \* FROM conditions WHERE CONDITION\_ONSET > CURRENT\_DATE - INTERVAL '30' DAY;”””

**Get Total Number of Encounters and Claims for Each Patient**

**“””**SELECT p.PATIENT\_ID, COUNT(e.EVENT\_ID) AS total\_encounters, COUNT(c.CLAIM\_ID) AS total\_claims FROM patients p LEFT JOIN encounters e ON p.PATIENT\_ID = e.PATIENT\_ID LEFT JOIN claims c ON p.PATIENT\_ID = c.PATIENT\_ID GROUP BY p.PATIENT\_ID;”””

**Count the Number of Active Patients**

“””SELECT COUNT(DISTINCT p.PATIENT\_ID) AS active\_patients FROM patients p JOIN encounters e ON p.PATIENT\_ID = e.PATIENT\_ID WHERE e.END\_DATETIME > CURRENT\_DATE - INTERVAL '1' YEAR;”””

Omar Hossam Attia

Analytics engineer