

# SQL NPE Claims Analytics (Synthetic)

*SQL-first claims reporting in PostgreSQL with a Power BI presentation layer*

Author: Babak Balouch

Date: 2026-02-12

## Executive summary

This portfolio project demonstrates end-to-end analytics delivery using a SQL-first approach. I designed a relational schema for patient injury compensation claims (synthetic), implemented a reproducible seeding strategy, authored basic and intermediate reporting queries, and built a thin Power BI report that consumes only database views.

The outcome is a clean, interview-safe project that shows how I model data, enforce quality rules, produce KPI-ready reporting layers, and present results in Power BI without shifting business logic out of the database.

## Project purpose and scope

### Purpose:

Show practical SQL competence (basic to intermediate) in a realistic healthcare context, aligned with work typical for statistical and analytics roles (data extraction, KPI definitions, and reporting).

### Scope:

- Database engine: PostgreSQL 16
- Schema design (6 core tables with bridge tables for many-to-many relationships)
- Synthetic data generation (rerun-friendly) with constraint-safe distributions
- Reporting queries (basic + intermediate) and a view-based reporting layer
- Power BI report built on views only; screenshots published to GitHub; PBIX kept local

### Out of scope:

- Use of real NPE case-level data (project uses synthetic data only)
- Advanced statistical modelling or machine learning
- Complex DAX modeling in Power BI (kept intentionally minimal)

## Tech stack and workflow

- PostgreSQL 16 (Homebrew on macOS; standardized PostgreSQL 16 instance on Windows for Power BI work)
- psql for deterministic execution of SQL scripts; DBeaver optionally for GUI inspection
- Git + GitHub for version control and portfolio packaging
- Power BI Desktop (Windows) for report presentation (Import mode)

Repository structure (key files):

Path	Purpose
sql/01_schema.sql	Schema (DDL): tables, constraints, indexes
sql/02_seed_data.sql	Synthetic seed data (rerun-friendly)
sql/03_queries_basic.sql	Basic interview-safe reporting queries
sql/04_queries_intermediate.sql	Intermediate reporting queries + quality checks
sql/05_views.sql	Stable reporting views for Power BI
powerbi/screenshots/	Public Power BI page screenshots
docs/	Session logs and build notes (Power BI notes added separately)

## Data model

The schema is normalized and centered on the claims fact table. Medical codes and injury types are modelled as separate dimensions, linked via bridge tables to support many-to-many relationships.

### Core tables:

- providers: provider attributes (name, type, region)
- claims: one row per claim with dates, status/decision, care level, region, amount, and provider reference
- medical\_codes + claim\_medical\_codes: code dictionary and claim-to-code linkage (Primary/Secondary role)
- injury\_types + claim\_injuries: injury dictionary and claim-to-injury linkage (is\_primary flag)

### Quality rules enforced by constraints include:

- Allowed value whitelists for status, decision, care\_level, patient\_sex, and region
- Decision fields must be NULL unless status is Closed; Closed requires a non-NULL decision
- decision\_date must be NULL or greater than or equal to received\_date
- Non-negative amounts

## Synthetic data generation

The seeding approach is pure SQL and rerun-friendly (TRUNCATE ... RESTART IDENTITY). Data is generated across multiple years with realistic categorical distributions and controlled variation across providers, codes, and injury types.

### Verified dataset sizes (after distribution fix):

- providers: 39
- medical\_codes: 84
- injury\_types: 16
- claims: 1200
- claim\_medical\_codes: 2400 (avg 2.00 codes/claim)
- claim\_injuries: 1800 (avg 1.50 injuries/claim)

A key engineering lesson came from an early uniformity artifact where random sampling behaved like a single 'pick once' result for the whole dataset. I corrected this by using correlated sampling and deterministic per-claim selection rules so distributions remain varied and stable across reruns.

## SQL reporting layer (queries + views)

### Basic reporting queries (sql/03\_queries\_basic.sql) cover:

- Monthly claim intake (received\_date)
- Status and decision distributions
- Amounts by decision
- Claims by region, care level, and patient sex
- Top providers, top medical codes, top injury types
- Average processing time (Closed claims)

### Intermediate queries (sql/04\_queries\_intermediate.sql) add:

- Monthly KPI table with approval rate and total payout
- Approval rate by region and rejection rate by care level
- Processing time stats by care level
- Backlog snapshot (Received + InReview) by region
- Top provider per region (ROW\_NUMBER) and most common code per region (DENSE\_RANK)
- High-payout outlier listing
- Data quality checks (all expected to return 0)

### Reporting views (sql/05\_views.sql) used by Power BI:

- vw\_monthly\_kpi: month-grain KPI table (received, closed, approval/rejection rates, payout, processing days)

- vw\_region\_kpi: region-grain KPI table
- vw\_provider\_summary: provider-grain KPI table

## Business questions and findings (synthetic data)

Because the dataset is synthetic, the findings below are illustrative. The goal is to show how I would answer typical stakeholder questions using a traceable SQL reporting layer.

### Q1. How is claim volume evolving over time?

Answered using vw\_monthly\_kpi (claims\_received by month) and the Executive Overview page trend charts. This supports communication needs (press/internal) by showing volume changes across time.

### Q2. Where are approval and rejection rates highest?

Answered using vw\_region\_kpi and region-level visuals. Rates are computed only on Closed claims and are NULL-safe to prevent misleading results when closed\_claims is zero.

### Q3. Which providers account for the highest workload and payouts?

Answered using vw\_provider\_summary (total\_claims, closed\_claims, approval\_rate\_closed, total\_payout\_nok). A Top-N provider visual highlights workload concentration and supports prioritization of deeper review.

### Q4. What does processing time look like, and does it differ by care level?

Processing time is measured as days between received\_date and decision\_date for Closed claims. In the seeded dataset, overall average processing time was approximately 91 days (basic query validation). Intermediate queries break this down by care\_level to identify potential workflow differences.

### Q5. Are there data quality issues that would undermine reporting?

Data quality checks in sql/04\_queries\_intermediate.sql verify constraint expectations (e.g., Closed with NULL decision, decision\_date earlier than received\_date, rejected claims with positive payouts). After the Part 5 fixes, these checks returned 0.

## Power BI report (views-only, SQL-first)

Power BI is used strictly as a presentation layer. The report imports only the three SQL views in Import mode, keeps the model minimal, and standardizes aggregation rules (Sum for counts/payout; Average for rates/durations).

**Pages:**

- Executive Overview: KPI cards (latest month snapshot), monthly claims trend, approval rate trend, month slicer
- Region Performance: total claims by region, approval rate by region, KPI table, optional scatter for volume vs rate
- Provider Summary: top providers by claim volume, approval rate for the same Top N, provider KPI table with slicers

To keep the public GitHub repo lightweight and readable, I published one screenshot per report page and kept the PBIX file local.

## What I learned

- How to design a normalized schema for a claims domain that supports reporting without over-modelling
- How to enforce reporting integrity with simple, explainable constraints
- How to seed synthetic data in pure SQL while avoiding uniformity artifacts (correlated sampling)
- How to build a stable reporting layer using views (clear grains and NULL-safe KPI definitions)
- How to keep Power BI thin: correct default aggregations, slicer interactions, and minimal DAX

## Limitations

- Synthetic data: results are illustrative and not representative of real NPE outcomes
- The latest month can have 0 received claims depending on date distribution (validated against the view, not a Power BI issue)
- PBIX kept local (public deliverable uses screenshots)

## Appendix: reproducible run sequence

From repo root, run:

- `psql -d npe_claims_demo -f sql/01_schema.sql`
- `psql -d npe_claims_demo -f sql/02_seed_data.sql`
- `psql -d npe_claims_demo -f sql/05_views.sql`
- `psql -d npe_claims_demo -f sql/03_queries_basic.sql`
- `psql -d npe_claims_demo -f sql/04_queries_intermediate.sql`