BudgetDB

Case Study — Designing a Finance & Workforce Data Warehouse

Mohammed Beheiry · beheiry.dev · github.com/Don-Zz/postgres-finance-workforce-analytics

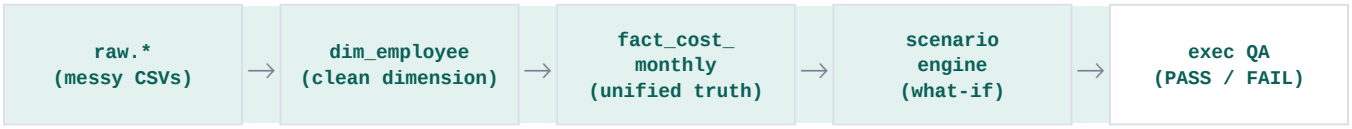
THE QUESTION

Where Is Operational Spend Concentrated, and Can Leadership Trust the Number?

Payroll, vendor spend, sales commissions, and travel & expense each lived in a different export, in a different format, with different naming conventions for the same people. Before any dashboard or budget conversation could happen, those four sources needed to become one trustworthy number leadership could act on — not four spreadsheets someone reconciled by hand every month.

ARCHITECTURE

Raw → Dimensional → Fact → Scenario → QA



Each source lands in `raw` exactly as exported — every column typed TEXT, no assumptions about clean data. A canonical `dim_employee` table then normalizes identity once, so every downstream join has one source of truth for who someone is. Four monthly cost views (payroll, vendors, commission, T&E) roll into a single `fact_cost_monthly` view — the one place dashboards, the scenario engine, and QA checks all read from, so no two reports can ever quietly disagree.

THE HARD PROBLEM

Reconciling Messy Names Without Guessing

The commission export and the payroll table never agreed on how to spell anyone's name: `"Jordan Blake (EE-AE)"` in one file, `"Jordn Blake"` in another. A single fuzzy match would have silently guessed wrong some percentage of the time — unacceptable when the output feeds payroll-adjacent reporting. Instead, matching resolves in three explicit, auditable tiers:

- **Tier 1 — manual override map.** Known typos and aliases get an explicit, reviewed mapping (`employee_commission_name_map`), not an algorithm's best guess.
- **Tier 2 — normalized name key.** Strip punctuation/case/role tags from both sides and match on the cleaned key — catches the formatting noise without guessing at spelling.
- **Tier 3 — flagged, not dropped.** Anything that still doesn't match is tagged `UNMATCHED` and surfaced in an audit view, never silently excluded from a total.

Every commission row carries a `match_method` and `join_status` column, so anyone downstream can see exactly how a number was resolved — and prove that nothing was silently dropped to make a total look cleaner.

THE TRUST LAYER

Nothing Reaches a Dashboard Without Passing QA

`v_exec_qa_checks` independently recomputes each cost category's total straight from its source view and compares it to what the unified fact table reports — a PASS/FAIL column, not a spot-check. If a future change to the join logic ever drops or double-counts a row, this view catches it before a leadership review does.

WHAT-IF MODELING

A Scenario Engine, Not a Copy-Pasted Spreadsheet

A `scenario` table stores named multiplier sets (e.g. "Vendors –10% review") over a date window; a view joins each scenario against the baseline fact table and computes the delta automatically. Testing "what if vendor spend dropped 10% this quarter" is one row insert, not a rebuilt spreadsheet.

31+ SQL views & checks in the warehouse	**98%** Reconciliation pass rate on QA checks	**30%** Cut in manual reporting time	**3-tier** Auditable identity resolution, not guesswork

STACK

`PostgreSQL · SQL (views, window functions, LATERAL joins, generated series) · raw/analytics schema separation · dimensional + unified fact modeling`

Full SQL: github.com/Don-Zz/postgres-finance-workforce-analytics · Mohammed Beheiry - BudgetDB - Full Data Warehouse.sql (companion file, this portfolio folder)