Simulating Transaction Flow and Reconciliation in a Fintech System

Tools: PostgreSQL (pgAdmin), SQL (CTEs, Joins), PL/pgSQL

Author: Sylvia Imisi

Project Objective

The core aim of this project was to simulate how a fintech company tracks transactions across multiple channels and performs automated reconciliation with merchants.

In a real world system, data flows from the frontend app (user-facing) to the backend database via APIs. However, because this project lacked a frontend, I simulated those flows using SQL triggers and functions to represent what would happen if users were actually performing transactions in real time.

I'm fascinated by the question: "Where does data go? Where does it live after a user performs an action?" This project was designed to answer that by creating a working backend that mimics a live system in finance.

Database Design

I created a normalized schema with realistic financial and operational tables, including:

- transactions
- wallet_transactions
- expected_settlement
- actual_settlement
- failed_attempts
- withdrawals
- merchants

Each table is logically connected, forming a fully normalized backend structure that reflects how a real fintech company tracks money movement, failed attempts, and financial summaries.

Backend Logic: Core Functions and Triggers

Instead of relying on a frontend/API, I built three key PostgreSQL functions that simulate how real systems handle live transactions and withdrawals:

1. block_failed_transaction

Purpose: Prevents failed transactions from entering the main transaction flow.

- If a transaction status is 'failed', it does not get inserted into transactions.
- Instead, it gets recorded in the failed_attempts table.
- This ensures the transactions table remains clean and reliable.

🔹 2. process successful transaction

Purpose: Handles all updates required for a successful transaction.

- Inserts the transaction into the transactions table.
- Updates the merchant's wallet balance in wallets.
- Inserts into expected settlement — logging what the merchant expects to receive.
- Updates the merchants table with current balance or stats (if needed).

🔹 3. process withdrawal

Purpose: Handles a merchant withdrawal event.

- Checks if the merchant has sufficient wallet balance.
- Deduct the withdrawal amount from the wallets table.
- Inserts a record into the withdrawals table.
- Logs the action in wallet transactions for full traceability.

Simulating a Live System (Without a Frontend)

Since there was no user interface or frontend:
- All 'user actions' were triggered through functions (as if APIs had called them).
- Triggers and logic flows ensured that each event affected multiple tables, mimicking a full backend workflow.
- Data was either inserted manually or via logic automation.

This made it possible to simulate how live data enters and moves within a fintech system — wallet top-ups, payouts, failed payments, reconciliations, etc.

Channel Insights and Query Analysis

After populating the system with transactions across channels (wallet, bank_transfer, and card), I ran several analytical SQL queries to uncover business insights.

Channel Distribution by Transaction Count:

- Wallet: 52.73%
- Bank Transfer: 29.96%
- Card: 17.32%

Channel Distribution by Failure Rate:

- Wallet: 42.59%
- Card: 35.19%
- Bank Transfer: 22.22%

Top 5 Merchants by Volume %(Amount)
- Sylvia Restaurant – 44.76%
- Adjokey Amala Joint – 16.40%

- Hope Fashion – 15.78%
- Agent POS – 11.90%
- Alpha Fig Store – 11.16%

Top 5 Merchants by Number of Transactions
- Hope Fashion – 26 transactions
- CY Delight Kitchen – 24 transactions
- Alpha Fig Store – 22 transactions
- Sylvia Restaurant – 22 transactions
- Grace Supermarket – 21 transactions

Reconciliation Engine (Highlight of the Project)

Reconciliation was the heart of this project just as it is in any real fintech company.

To practice this:
- I intentionally mismatched entries in the actual_settlement tables:
  - Some settlements were missing
  - Others were delayed, overpaid, or underpaid

Then, using CTEs and LEFT JOINS, I wrote SQL queries to generate a Reconciliation Report, identifying:
- Missing settlements
- Delayed settlements
- Overpayments
- Underpayments

This mirrors what fintech ops teams do daily, comparing what should've been paid out vs. what actually was.

Key Skills Demonstrated

- Deep understanding of how financial data flows through a backend system.
- Writing automated logic using PostgreSQL functions and triggers.
- Designing a clean, normalized schema that reflects real-world flows.
- Using SQL joins, CTEs, and filters to extract performance and reconciliation reports.
- Thinking like a fintech operations analyst — focusing not just on transactions, but also failures, exceptions, and data mismatches.

Summary

This project helped me simulate what goes on under the hood in a fintech platform. Every time a user transacts, data flows across systems and this backend simulation shows exactly how that happens, even without a frontend.

The highlight was building a real reconciliation flow that surfaces financial discrepancies just like a real settlement or ops team would.

Deliverables

- SQL scripts for all functions and triggers
- Full database schema and relationships
- Reconciliation query using CTEs and LEFT JOINS
- Summary KPIs from data exploration
- Project documentation/report