Data Restructuring Legacy POS System

Migration from File-Based Storage
to PostgreSQL Relational Database

Version 1.0 2025-11-28

Status: **Ready for Implementation**

Contents

1 Overview

This document outlines the data restructuring phase for migrating the legacy POS system from fragile file-based storage (`*.txt` files) to a robust, normalized relational database using **PostgreSQL**. The new design ensures referential integrity, ACID compliance, concurrency safety, and significantly improved data quality.

2 Current Data Storage Issues

2.1 File-Based Storage Problems

1. **No Normalization** – Denormalized records (e.g., phone + all rentals in one line in `userDatabase.txt`)
2. **No Referential Integrity** – Orphaned records possible
3. **No Transaction Support** – Partial writes can corrupt files
4. **No Concurrency Control** – Race conditions and overwrites
5. **Inconsistent Formats** – Mixed space/comma delimiters
6. **No Data Validation** – Invalid quantities, dates, formats accepted
7. **No Audit Trail Integrity** – Append-only logs without checksums

3 Target Database Schema

3.1 Technology Choice: PostgreSQL

Rationale:

- Mature, ACID-compliant, industry-standard
- Excellent Spring Boot / JPA integration
- Rich feature set (JSONB, full-text search, advanced indexing)
- Open source – no licensing costs
- High concurrency performance

3.2 Entity Relationship Diagram

4 Database Schema DDL

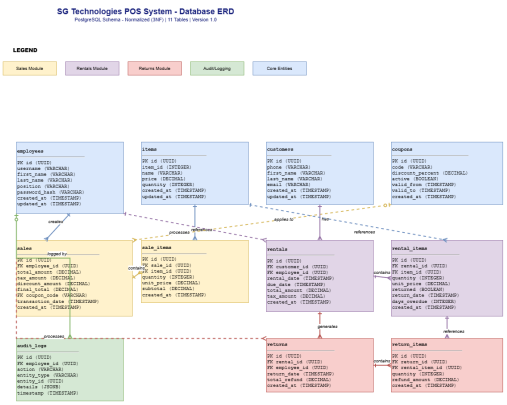

Figure 1: Entity Relationship Diagram – Normalized PostgreSQL Schema for the Legacy POS System

```sql
-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Employees table
CREATE TABLE employees (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    username VARCHAR(50) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    position VARCHAR(20) NOT NULL CHECK (position IN ('Admin', 'Cashier')),
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Items table
CREATE TABLE items (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    item_id INTEGER UNIQUE NOT NULL,
    name VARCHAR(200) NOT NULL,
    price DECIMAL(10, 2) NOT NULL CHECK (price >= 0),
    quantity INTEGER NOT NULL CHECK (quantity >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Customers table
CREATE TABLE customers (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    phone VARCHAR(20) UNIQUE NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Coupons table
CREATE TABLE coupons (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    code VARCHAR(50) UNIQUE NOT NULL,
    discount_percent DECIMAL(5, 2) NOT NULL CHECK (discount_percent >= 0
       AND discount_percent <= 100),
    active BOOLEAN DEFAULT TRUE,
    valid_from TIMESTAMP,
    valid_to TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Sales table
CREATE TABLE sales (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    employee_id UUID NOT NULL REFERENCES employees(id),
    total_amount DECIMAL(10, 2) NOT NULL CHECK (total_amount >= 0),
    tax_amount DECIMAL(10, 2) NOT NULL CHECK (tax_amount >= 0),
    discount_amount DECIMAL(10, 2) DEFAULT 0 CHECK (discount_amount >= 0),
    final_total DECIMAL(10, 2) NOT NULL CHECK (final_total >= 0),
```

```
    coupon_code VARCHAR(50) REFERENCES coupons(code),
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Sale items table
CREATE TABLE sale_items (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    sale_id UUID NOT NULL REFERENCES sales(id) ON DELETE CASCADE,
    item_id UUID NOT NULL REFERENCES items(id),
    quantity INTEGER NOT NULL CHECK (quantity > 0),
    unit_price DECIMAL(10, 2) NOT NULL CHECK (unit_price >= 0),
    subtotal DECIMAL(10, 2) NOT NULL CHECK (subtotal >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Rentals table
CREATE TABLE rentals (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    customer_id UUID NOT NULL REFERENCES customers(id),
    employee_id UUID NOT NULL REFERENCES employees(id),
    rental_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    due_date TIMESTAMP NOT NULL,
    total_amount DECIMAL(10, 2) NOT NULL CHECK (total_amount >= 0),
    tax_amount DECIMAL(10, 2) NOT NULL CHECK (tax_amount >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Rental items table
CREATE TABLE rental_items (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    rental_id UUID NOT NULL REFERENCES rentals(id) ON DELETE CASCADE,
    item_id UUID NOT NULL REFERENCES items(id),
    quantity INTEGER NOT NULL CHECK (quantity > 0),
    unit_price DECIMAL(10, 2) NOT NULL CHECK (unit_price >= 0),
    returned BOOLEAN DEFAULT FALSE,
    return_date TIMESTAMP,
    days_overdue INTEGER DEFAULT 0 CHECK (days_overdue >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Returns table
CREATE TABLE returns (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    rental_id UUID NOT NULL REFERENCES rentals(id),
    employee_id UUID NOT NULL REFERENCES employees(id),
    return_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_refund DECIMAL(10, 2) NOT NULL CHECK (total_refund >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Return items table
CREATE TABLE return_items (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    return_id UUID NOT NULL REFERENCES returns(id) ON DELETE CASCADE,
    rental_item_id UUID NOT NULL REFERENCES rental_items(id),
    quantity INTEGER NOT NULL CHECK (quantity > 0),
    refund_amount DECIMAL(10, 2) NOT NULL CHECK (refund_amount >= 0),
```

```
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Audit logs table
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    employee_id UUID REFERENCES employees(id),
    action VARCHAR(50) NOT NULL,
    entity_type VARCHAR(50),
    entity_id UUID,
    details JSONB,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_employees_username ON employees(username);
CREATE INDEX idx_items_item_id ON items(item_id);
CREATE INDEX idx_customers_phone ON customers(phone);
CREATE INDEX idx_sales_employee ON sales(employee_id);
CREATE INDEX idx_sales_date ON sales(transaction_date);
CREATE INDEX idx_sale_items_sale ON sale_items(sale_id);
CREATE INDEX idx_rentals_customer ON rentals(customer_id);
CREATE INDEX idx_rental_items_rental ON rental_items(rental_id);
CREATE INDEX idx_rental_items_returned ON rental_items(returned);
CREATE INDEX idx_audit_logs_employee ON audit_logs(employee_id);
CREATE INDEX idx_audit_logs_timestamp ON audit_logs(timestamp);
```

Listing 1: PostgreSQL Schema Creation Script

5 Data Migration Strategy

5.1 Migration Steps

1. Parse all legacy `*.txt` files
2. Data cleaning (deduplication, phone validation, date normalization, etc.)
3. Transform into normalized structures
4. Bulk insert with proper foreign key resolution
5. Post-migration validation and integrity checks

5.2 Migration Script Structure (Java Pseudo-code)

```
1. Read employeeDatabase.txt      employees table
2. Read itemDatabase.txt          items table
3. Read userDatabase.txt          customers + rentals + rental_items
4. Read saleInvoiceRecord.txt     sales + sale_items
5. Read returnSale.txt            returns + return_items
6. Read couponNumber.txt          coupons table
7. Read employeeLogfile.txt       audit_logs table
```

Listing 2: High-level Migration Flow

6 Schema Improvements

6.1 Normalization Benefits

- **1NF**: Atomic values, no repeating groups
- **2NF**: No partial dependencies
- **3NF**: No transitive dependencies

6.2 Data Quality Improvements

- CHECK constraints on ranges
- Foreign keys with ON DELETE CASCADE where appropriate
- UNIQUE constraints on natural keys
- NOT NULL + sensible DEFAULTs

6.3 Security Improvements

- Passwords stored as bcrypt hashes only
- Comprehensive audit trail in `audit_logs`
- `Database-level validation prevents injection of bad data`

7 Migration Plan

Phase 1	Schema Creation – tables, constraints, indexes, UUID extension
Phase 2	Data Migration – parsing, cleaning, loading, validation
Phase 3	Application Integration – JPA repositories, @Transactional services
Phase 4	Verification & Go-Live – data comparison, load testing, rollback plan

8 Repository Pattern Implementation (Spring Data JPA)

```java
// EmployeeRepository
public interface EmployeeRepository extends JpaRepository<Employee, UUID> {
    Optional<Employee> findByUsername(String username);
    List<Employee> findByPosition(String position);
}

// ItemRepository
public interface ItemRepository extends JpaRepository<Item, UUID> {
    Optional<Item> findByItemId(Integer itemId);
    List<Item> findByQuantityGreaterThan(int minQuantity);
```

```
}

// CustomerRepository
public interface CustomerRepository extends JpaRepository<Customer, UUID> {
    Optional<Customer> findByPhone(String phone);
}

// SaleRepository
public interface SaleRepository extends JpaRepository<Sale, UUID> {
    List<Sale> findByEmployeeIdAndTransactionDateBetween(
        UUID employeeId, LocalDateTime start, LocalDateTime end);
}

// RentalRepository
public interface RentalRepository extends JpaRepository<Rental, UUID> {
    List<Rental> findByCustomerId(UUID customerId);
    List<Rental> findByDueDateBeforeAndReturnedFalse(LocalDate date);
}
```

Listing 3: High-level Migration Flow

9 Benefits of New Schema

- **Scalability** – concurrent transactions handled safely
- **Data Integrity** – impossible to create orphaned records
- **Query Performance** – optimized indexes for common reports
- **Maintainability** – clean, normalized structure
- **Extensibility** – easy to add loyalty, inventory alerts, analytics
- **Security** – proper hashing and audit trail
- **Reporting** – complex analytics now trivial with SQL

Document Version: 1.0
Date: 2025-11-28
Status: Ready for Implementation