CSC540 Project : Inventory Management Final Report

Team Members: JonCarlo Migaly, Gaurav Mehta, Shrawani Gulhane, Paul Nguyen

I. Summary of Design Rationale and Improvements

Final Design Philosophy

The primary objective of the final schema design was to ensure traceability (linking product lots to ingredient lots) and atomic integrity (guaranteeing production transactions succeed entirely or fail safely). This required abandoning the preliminary design's focus on simple lookup tables and embracing a highly normalized structure dependent on composite, traceable keys.

Key Improvements from Preliminary Design:

1. Elimination of Redundant/Non-Normalized Structures: The initial tables for users were consolidated into a single AppUser table linked by Foreign Keys, eliminating transitive dependencies and simplifying access control.
2. Guaranteed Traceability (Atomic Keys): The primary keys for all inventory items (IngredientBatch and ProductBatch) were changed from simple auto-increment IDs to composite, human-readable lot numbers (e.g., ING123-SUP45-B0009), which are automatically generated and enforced by database triggers.
3. Transaction-Level Integrity: Critical rules (Negative Stock, Health Risk) were implemented directly within the DBMS layer to ensure they are non-negotiable and protected by the transactional rollback mechanism, preventing data corruption from application errors.

II. Functional Dependencies and BCNF Normalization

The schema is built to conform to Boyce-Codd Normal Form (BCNF), which is a stricter requirement than 3NF. This design ensures that every determinant in a non-trivial functional dependency is a candidate key.

Analysis of Normalization (BCNF)

Table	Primary Key (PK)	Key Functional Dependencies (FDs)	Normal Form

AppUser	user_id	user_id -> username, role, manufacturer_id	BCNF
Product	product_id	product_id -> name, category_id, standard_batch_size, ...	BCNF
IngredientBatch	lot_number	lot_number -> ingredient_id, supplier_id, quantity_on_hand, ...	BCNF
ProductBatch	lot_number	lot_number -> product_id, total_batch_cost, produced_quantity, ...	BCNF
Formulation	formulation_id	formulation_id -> supplier_id, ingredient_id, unit_price, ...	BCNF
RecipeIngredient	(recipe_id, ingredient_id)	(recipe_id, ingredient_id) -> quantity, unit_of_measure	BCNF
BatchConsumption	(product_lot_number, ingredient_lot_number)	(product_lot_number, ingredient_lot_number) -> quantity_consumed	BCNF

Justification: All tables satisfy BCNF. The chosen design intentionally uses composite Primary Keys and Foreign Keys to eliminate redundant data structures and ensure full dependency on the entire key set.

III. Constraint Implementation and Justification

The design uses the DBMS for integrity checks (non-negotiable rules) and the application layer for polite, complex business policies (UX).

A. Core Integrity Constraints (Implemented in the DBMS)

These constraints are implemented via Stored Procedures and Triggers to ensure Atomicity, Consistency, and Traceability against concurrent user errors or faulty application logic.

Constraint	Implementation	Rationale for DBMS (Why it MUST be here)
Atomic Production	Procedure: Record_Production_Batch (Uses TRANSACTION with ROLLBACK).	Atomicity. Guarantees that the multi-step batch creation is an all-or-nothing operation, protecting financial and inventory integrity.

Health Risk Block	Procedure: Evaluate_Health_Risk	Critical Safety. Enforced inside the transaction to guarantee that no batch containing conflicting ingredients can ever be permanently written to the database.
Negative Inventory	Trigger: trg_validate_consumption (BEFORE INSERT)	Final Guardrail. Prevents race conditions from dropping inventory below zero, ensuring the absolute correctness of the quantity_on_hand attribute.
Recall & Traceability	Procedure: Trace_Recall	Complex Read Logic. Centralizes the multi-table JOIN logic and date arithmetic (20-day window) required to fulfill the recall requirement efficiently.

B. Policy and User Experience Constraints (Implemented in Application Code - Python)

These are "polite" rules enforced by the main.py application to provide good user feedback *before* the database is even touched.

Constraint	Implementation	Justification
FEFO (First-Expired, First-Out)	Python Logic in create_product_batch	Policy Rule. Requires complex iteration, sorting, and calculation logic, which is more readable and flexible in Python code.
Standard Batch Size Multiple	Python Logic in create_product_batch	UI Validation. The check quantity % SBS == 0 is performed in Python to provide the user with immediate, friendly feedback, improving UX.
90-Day Expiration Rule (on Intake)	Python Logic in create_supplier_batch	UI Validation. Checks the date upon input to provide immediate feedback to the supplier, preventing them from filling out a full form only to have the database reject it.