CSC 540: Project Report

Shraawani Lattoo, Jazmine Green, Ava Collier, Gabe Milburn

November 2025

1 Introduction

This project focuses on the operation of Inventory Management for a small manufacturer that produces prepared and frozen meals. Our system supports the management of products, ingredients, recipes, formulation, and prices, and roles for the manufacturer, supplier, and viewer. We want to ensure the tracking of production and materials, and to do so, we implemented an database system.

Our final report will describe the implementation decisions for our database system, the ER diagram accumulated, explanation of the functional dependencies, normalization process, and constraints for each attribute.

2 ER Model

The Entity-Relation model represents the whole structural diagram for use in database design. We conduct an ER diagram for database design, debugging, creation and patching, and requirements gathering. The following describes our diagram and how each entity represents the objects: Product, Ingredient, Manufacturer, or Batch.

Following our original ER Diagram and schema, we made changes to improve the normalization, functional dependencies, and key relationships across multiple entities. The updates are made the ensure the tables are Boyce-Codd Normal Form, which reduces redundancy and makes the diagram with dependency integrity. This included defining new primary keys and removing unnecessary attributes used. These changes were found during normalization and to ensure each attribute depends only on the primary keys.

All tables were updated to ensure full compliance with BCNF by removing attributes that caused partial, transitive, or non-key-based dependencies. The User, Supplier, Manufacturer, Ingredient, and ProductCategory tables required no structural changes since their attributes already acted as keys or depended fully on their primary keys.

The Formulations table was refined by removing the isActive attribute, which introduced dependency issues, and adding VersionNumber to correctly track formulation revisions. In the FormulationIngredientList table, the ValidityDate attribute was removed because it depended only on part of the composite key (FormulationID, MaterialID). The IngredientBatch table was simplified to keep LotID as the sole key, removing attributes such as IngredientID, ValidityDate, SupplierID, and BatchID; additionally, BatchID no longer auto-increments and is now uniquely identified in combination with SupplierID. The DoNotCombineList table now uses the composite key (Ingredient1ID, Ingredient2ID) after removing SupplierID to eliminate unrelated dependencies. The Product table was adjusted by defining ProductID as the primary key and removing the previous unique constraint on (ManufacturerID, ProductName) to avoid partial dependencies, while the Recipe table was simplified by removing ManufacturerID so that RecipeID is the only determinant. In the RecipeBOM table, the composite key (RecipeID, IngredientID) was retained, and attributes such as the duplicate IngredientID field and CreationDate were removed to eliminate dependency violations. Finally, the ProductBatch table now uses LotID as the only key after removing RecipeID, Costs, BatchID, and ManufacturerID, ensuring that every remaining attribute depends solely on the key.

3 Functional Dependencies

The functional dependencies were driven by each of our table's primary keys. Different attributes were removed across tables because of BCNF violations. All of our relations in the final schema satisfy BCNF,

with each primary or composite key cleanly determining all remaining attributes.

- **User**

$$\text{UserID} \rightarrow \text{Username}$$

The primary key UserID determines all other attributes. Username is UNIQUE.

- **Supplier**

$$\text{All attributes are keys}$$

Relations consist of key attributes.

- **Manufacturer**

$$\text{All attributes are keys}$$

All attributes determine each other, meaning no non-key dependencies exist.

- **Ingredient**

$$\text{IngredientID} \rightarrow \text{(all other attributes)}$$

IngredientID is the main key and determines IngredientName and isCompound.

- **Formulations**

$$\text{FormulationID} \rightarrow \text{(all other attributes)}$$

FormulationID determines the formulation details; attributes like isActive were removed because they were not fully dependent on the key.

- **FormulationIngredientList**

$$\text{(FormulationID, MaterialID)} \rightarrow \text{(all other attributes)}$$

(Composite key; fully determines all other attributes.) ValidityDate was removed because it did not depend on the entire key.

- **IngredientBatch**

$$\text{LotID} \rightarrow \text{(all remaining attributes)}$$

LotID is the primary key(Removed IngredientID, ValidityDate, SupplierID, BatchID).The BatchID portion of LotID is now unique only when paired with SupplierID.)

- **DoNotCombineList**

$$\text{(Ingredient1ID, Ingredient2ID)} \rightarrow \text{(all other attributes)}$$

This is a Composite key. SupplierID was removed because it did not depend on this composite key.

- **ProductCategory**

$$\text{All attributes are keys}$$

Each attribute uniquely determines the others.

- **Product**

$$\text{ProductID} \rightarrow \text{(ManufacturerID, ProductName)}$$

ProductID determines all product information; a unique constraint on (ManufacturerID, ProductName) was removed.

- **Recipe**

$$\text{RecipeID} \rightarrow \text{(all other attributes)}$$

RecipeID determines recipe properties; ManufacturerID was removed because it did not depend on the key.

- **RecipeBOM**

$$\text{(RecipeID, IngredientID)} \rightarrow \text{(all remaining attributes)}$$

The composite key identifies each ingredient in a recipe (Removed redundant IngredientID and CreationDate attributes).

- **ProductBatch**

$$\text{LotID} \rightarrow \text{(all remaining attributes)}$$

LotID uniquely determines batch information (Removed RecipeID, Costs, BatchID, ManID).

4 Normalization

Table	Primary Key	Normal Form	Comments
User	UserID	BCNF	UserID determines all other attributes; Username is UNIQUE.
Supplier	SupplierID	BCNF	SupplierID is PK; UserID is unique. All attributes are keys or unique.
Manufacturer	ManufacturerID	BCNF	Same as Supplier; all attributes are keys or unique.
Ingredient	IngredientID	BCNF	IngredientID determines IngredientName and Is-Compound.
Formulation	FormulationID	BCNF	FormulationID is PK; fully determines IngredientID, SupplierID, PackSize, UnitPrice, VersionNumber, Effective dates.
FormulationIngredientList	(FormulationID, MaterialID)	BCNF	Composite PK; Quantity depends on the full key.
IngredientBatch	LotID	BCNF	LotID determines FormulationID, ManufacturerID, Quantity, and ExpirationDate.
DoNotCombineList	(Ingredient1ID, Ingredient2ID)	BCNF	Composite key determines no other attributes; satisfies BCNF.
ProductCategory	CategoryID	BCNF	PK determines CategoryName; no other dependencies.
Product	ProductID	BCNF	PK determines CategoryID, ManufacturerID, ProductName, DefaultBatchSize.
Recipe	RecipeID	BCNF	PK determines ProductID, CreationDate.
RecipeBOM	(RecipeID, IngredientID)	BCNF	Composite PK; Quantity fully depends on both keys.
ProductBatch	LotID	BCNF	LotID determines RecipeID, ProductionDate, ExpirationDate, BatchQuantity.
ProductBatchIngredient Batch	(ProductLotID, Ingredient-LotID)	BCNF	Composite PK; QuantityUsed depends on full key.

5 Constraints

Database Constraints

- **Primary Keys:** Uniquely identify records (e.g., `User(UserID)`, `Ingredient(IngredientID)`).

- **Foreign Keys:** Enforce referential integrity (e.g., `Supplier(UserID) → User(UserID)`).

- **Unique constraints:** Prevent duplicates (`User(Username)`,`Ingredient(IngredientName)`, `Product(ManufacturerID, ProductName)`).

- **NOT NULL / CHECK:** Ensure valid values (positive quantities on any quantity or dollar value, valid user roles, start date < end date, and that there are no repeat do-not-combine combinations).

- **Ordering rules:** `DoNotCombineList(Ingredient1ID < Ingredient2ID)` prevents duplicate ingredient pairs.

- **Prevent Expired Consumption:** Trigger ensures that batches cannot be made with expired ingredients.

- **Atomic Do-Not-Combine:** Trigger ensures that do-not-combine pairs only have atomic ingredients.

- **Custom Lot ID generation:** For `IngredientBatch` and `ProductBatch`.

- **Evaluate Health Risk Violations:** Make sure ingredients that shouldn't be combined are warned about in formulations, ingredient batches, and recipes, and blocked in product batches.

- **Manage Ingredient Dates:** Ensure that ingredient versions do not have overlapping effective dates

- **Formulation Rules:** Atomic and Compound ingredients must use the same ingredient ID for the same ingredient name, but can have different quantities and prices in different formulations.

- **Valid IDs:** Ensures that various IDs exist before use. For example, ensure that an ingredient ID exists before using that ingredient in a recipe, or that a recipe ID exists before using that recipe in a product batch.

- **Valid Dates:** Ensure any dates entered are valid calendar year dates that exist.

Application Constraints

- **Valid Database Connection:** Ensures that the database is connected properly.

- **Valid User:** Ensures that the UserID used to login actually exists in the database.

- **Ensure Clean Transactions:** Ensure any remaining output from past queries is caught before running a new query.

- **Numeric/Date Input:** There are places where the application re-checks for valid numeric input, dates, and positive values, where required, to allow for appropriate error messages, loop functionality, and the option to re-enter an invalid value for ease of use. However, the database will always ensure the appropriate numeric type is provided before any values are stored, regardless of any error-handling logic on the application side (e.g., positive, float, integer, check constraints).

6   Summary

Most of the rules for keeping the data accurate and consistent are used on the database. The more abstract rules, such as ensuring that the database can be connected and used appropriately, are left for the application to handle. Additionally, some repeat checks are used in the application to allow the system to handle user error smoothly, but all data-centric constraints are validated by the database, so that no invalid data is ever entered.