DBMS Project Report
Prepared & Frozen Meal Manufacturing Database — Design

Team Members: Dharani guda(dguda), Ayush Gupta(agupta86), Sasidhar Appalla(pappall), Shanmukha Srinivas Sai Chatadi(schatad)

Design Rationale and Overall Approach:

The goal of this project was to design a relational database that would capture data for the products of a small prepared and/or frozen meals manufacturer, like Steak Dinner, Mac & Cheese, or Chocolate Lava Cake, for example. It was to be a model of how a manufacturer of prepared and/or frozen meals would define the product offerings, recipes, ingredients, manage relationships with suppliers, carry out production in batches, and execute lot-level traceability for ingredients and finished goods. This system is intended as the basis for batch manufacturing, costing, labeling, and recall activities, while keeping the model normalized and auditable.

For this, we designed a relational schema that was fully normalized to the BCNF level. It modeled everything from the creation of products and ingredients to their suppliers, recipes, and batches, including how they are consumed. We documented all the functional and conditional dependencies. The DBMS enforces the basic constraints, while the complex rules fall under the responsibilities of the application. Such a design enforces the correctness of data through checks, FKs, etc., at the DB level. In addition, it will allow an app to implement lot selection logic, such as FEFO, recall tracking, and higher-order business rules. In other words, this schema would support getting costing, labeling, and traceability correct, record history of suppliers, and ensure that the manufacturing process remains compliant with the rules.

Entities and Functional Dependencies:

This database models how a small meal manufacturer keeps track of its products and ingredients received etc. Each Product is part of exactly one Category and is associated with exactly one Recipe, which in turn requires certain ingredients to prepare it through RecipeIngredient. This separation follows the following functional dependencies: product_number → (product_name, category_id) and recipe_id → product_id, in order to avoid the redundancy of evolving the products and recipes independently. Ingredients are represented in the Ingredient table and can be atomic or compound; compound ingredients are decomposed into materials using IngredientComposition, keyed by the pair (compound_id, material_id), so that the quantity of each material depends only on such pairs *(Appendix A)*.

The system also captures how suppliers version their ingredients over time. Each supplier specific formulation is stored in SupplierIngredientFormulation, and its materials are stored in FormulationMaterial. This structure is dictated by the FD formulation_id → (supplier_id, ingredient_id, dates, pack_size, unit_price), which maintains the integrity of past versions without overwriting previous data. Stock levels are managed on incoming ingredient lots through IngredientBatch and on lots of manufactured products through ProductBatch, both uniquely identified by a lot_number with the dependencies lot_number → (ingredient details) and lot_number → (product details). Consumption links are stored in ConsumedIngredientLot that captures which ingredient batches were used to make each product batch.

All together, these relationships establish complete lot-level traceability: IngredientBatch → ConsumedIngredientLot → ProductBatch. This chain of functional dependencies allows for proper costing, labeling, and recall identification. Other conditional dependencies involve properties inherently temporal or dependent upon processes, including one-level ingredient composition, FEFO lot selection, nonoverlapping formulation periods, and integer-multiple batch sizing. These will be implemented in application logic. All functional dependencies, including business-rule FDs not reflected directly in the schema, are documented fully in *Appendix A (Functional Dependencies).*

Normalization and Decomposition:

The functional dependencies, as shown in Appendix A, made it clear we needed normalization. They showed how info depended on a unique ID and where we needed to break things down to cut out the extra stuff. Like at the start, product_number → (product_name, category_id) said loud and clear that product names and categories should only depend on the product key. So, to keep from repeating category names for every product, Category got its own table with its own key, category_id. Product just keeps a link to it. The simple link between Product and Recipe came from recipe_id → product_id and the fact that product_id → recipe_id also worked one way, making sure each product has just one recipe template. The many recipes and ingredients, based on (recipe_id, ingredient_id) → quantity needed a RecipeIngredient table to get rid of partial dependencies and hit BCNF. That way, changing ingredient numbers or an ingredient's name only needs to be done in one place, so no weird update or delete issues. Also, (formulation_id) → (supplier_id, ingredient_id, start_date, end_date, unit_price, pack_size_uom) showed that supplier formulations can change over time independently. This led to splitting things up into SupplierIngredientFormulation (for time and price info) and FormulationMaterial (for what a formulation is made of). Now, adding a new formulation or changing a price doesn't mess with the old info.

The same reasoning was applied subsequently. The dependency lot_number → (product_id, manufacturer_id, quantity, dates) made ProductBatch a table in its own right, keyed by a lot number. Then, lot_number → (ingredient_id, supplier_id, expiry, receipt_date, quantities) defined IngredientBatch as its own thing so you can track recipes of compund ingredients and formulations of suppliers. The link (product_batch_id, ingredient_lot_id) → quantity_used pointed to how batches and ingredient lots are used together, giving ConsumedIngredientLot as the link to help with costing, labeling, and recall queries. On the assembly side, functional dependencies like (compound_id, material_id) → qty_required led to the IngredientComposition table, which keeps maker-defined compound ingredients separate from base ingredient facts and prevents you from sticking a bunch of material rows in the Ingredient table. Because this breaking down eliminates any excess or partial dependencies, each piece of information now depends upon a key and nothing else. Every table reaches at least Third Normal Form, and most reach Boyce-Codd Normal Form, as you will see in *Appendix B*. The result is a stable setup.

Constraints discovered:

While most structural constraints appear in the ER and schema diagrams, several business and operational constraints required explicit documentation and implementation outside the physical schema, as detailed below

1. Application-enforced rules (process, temporal, aggregate)

- Only compound ingredients may appear as parents in IngredientComposition; materials cannot be parents, and cycles are not allowed.
- Supplier formulations must not overlap in time for the same ingredient, and suppliers may create IngredientBatch entries only for ingredients they formulate.
- ProductBatch.quantity_produced must be an integer multiple of the product's standard batch size.
- FEFO lot selection is enforced with no splitting; the earliest unexpired ingredient batch that fully satisfies the needed quantity must be chosen.
- Ingredient batches used in production must not be expired, and total consumption cannot exceed a batch's received quantity.
- Labeling expands compounds into atomic materials using the active formulation and orders them by contribution.
- Recall checks follow IngredientBatch → ConsumedIngredientLot → ProductBatch within a 20-day window.
- Access control ensures users can only act on data tied to their associated manufacturer or supplier, and DoNotCombine prevents incompatible ingredient pairs.

2. DB-level CHECKs (simple row-level invariants)

- Positive and non-negative numeric checks: required quantities, used quantities, on-hand amounts, and unit prices.
- Date-sanity checks such as expiry ≥ production, end ≥ start, and receipt_date ≤ today.
- A 90-day intake rule rejects ingredient batches with short shelf life.
- Lot-number formats are validated in the application, and product lot numbers remain globally unique.

3. Coverage and business-semantic rules

- Each recipe ingredient must map to exactly one consumed ingredient batch (no splitting, no omissions).
- Costing uses a consistent policy based on cost_per_pack / pack_size and the active formulation at production time.

Conclusion:

This report described how a fully normalized and traceable database design was developed for the prepared-meal manufacturing product. Each functional dependency led to a clear decomposition that eliminates redundancy and provides a consistent view of data on products, recipes, suppliers, ingredient batches, and product batches. The schema obtained from here offers reliable recall and access control without sacrificing historical accuracy or operational flexibility. From a design perspective, the solution developed here meets an appropriate balance between theoretical rigor and real-world usability; this enables the developed system to act as a secure and auditable method of laying the groundwork for manufacturing and supply-chain management.

Appendix

Appendix A: Functional Dependencies derived from project description.

ID	Conditional FD / Rule	Purpose (short)
FD-1	compound_id → ingredient_type = 'compound'	Only compound ingredients may appear as parent in IngredientComposition
FD-2	material_id not a parent elsewhere	Enforces one-level composition (no grandchildren)
FD-3	For each (supplier_id, ingredient_id), formulation date ranges must not overlap	Ensures only one active formulation version at any time
FD-4	quantity_produced must be integer multiple of standard batch size	Enforces manufacturing batch size rules
FD-5	Σ ConsumedIngredientLot.quantity_used ≤ IngredientBatch.received_qty	Prevents over-consumption of ingredient lots
FD-6	expiry_date ≥ receipt_date + 90 days	Ingredient intake must have ≥90-day shelf life
FD-7	IngredientBatch.expiry_date ≥ ProductBatch.production_date	Expired ingredient lots cannot be used
FD-8	FEFO selection: choose earliest unexpired lot with enough quantity; no splitting	Implements proper ingredient-lot selection
FD-9	Active formulation for costing/labeling determined by production date	Ensures accurate, time-based cost/label computation
FD-10	Traceability chain: IngredientBatch → ConsumedIngredientLot → ProductBatch	Enables recall within 20-day window
FD-11	on-hand quantity derived from movements	Avoids storing redundant inventory values
FD-12	Supplier must formulate ingredient to create IngredientBatch	Prevents invalid supplier-ingredient combinations

FD-13	Users may only modify data aligned with their manufacturer/supplier role	Enforces access control via User table
FD-14	DoNotCombine prohibits certain ingredient pairs from co-occurring	Prevents allergen/safety issues
FD-15	Label expansion sorted by contribution, tie-broken deterministically	Ensures consistent, compliant labels

Appendix B: Normalization Summary

Table Name	Primary / Candidate Keys	Highest Normal Form	Notes
Category	{category_id}, {category_name}	**BCNF**	Names unique; no transitives
Manufacturer	{manufacturer_id}	**BCNF**	Simple entity
Supplier	{supplier_id}	**BCNF**	Simple entity
Ingredient	{ingredient_id}	**BCNF**	Ingredient attributes fully depend on key
User	{user_id}, {username}	**BCNF**	Username unique; roles separated cleanly
DoNotCombine	{conflict_id}	**BCNF**	Stores disallowed ingredient pairs
Product	{product_number}, {product_name}	**BCNF**	Product names unique; no partial deps
Recipe	{recipe_id}, {product_id}	**BCNF**	1:1 with product enforced
RecipeIngredient	{(recipe_id, ingredient_id)}	**BCNF**	Removes many-to-many redundancy

IngredientComposition	{(compound_id, material_id)}	**BCNF**	One-level manufacturer BOM
SupplierIngredientFormulation	{formulation_id}	**BCNF**	Temporal attributes depend on key
FormulationMaterial	{(formulation_id, materials_id)}	**BCNF**	Ingredient materials for formulation version
IngredientBatch	{lot_number}	**BCNF**	Incoming ingredient lots
ProductBatch	{lot_number}	**BCNF**	Finished product lots
ConsumedIngredientLot	{(product_batch_id, ingredient_lot_id)}	**BCNF**	M:N link between product batches & ingredient lots