DBMS PROJECT FINAL REPORT

1. Introduction

This database was designed for a mid-sized prepared and frozen meals manufacturer. It supports three roles (Manufacturer, Supplier, Viewer), ingredient and recipe management, FEFO-based inventory usage, detailed recall traceability, and simple reporting. The schema in 01_schema_and_logic.sql is used by a Python application whose behaviour is documented in the role function files (manufacturer_actions.py, supplier_actions.py, viewer_actions.py).

The design goals were: (i) derive tables from clear functional dependencies, (ii) normalize all base tables to at least Third Normal Form (3NF), and preferably Boyce–Codd Normal Form (BCNF), and (iii) push as many business constraints as practical into the DBMS via keys, foreign keys, CHECK constraints, triggers, procedures and views, leaving only UI- and role-specific rules to application code.

2. Functional Dependencies that Shaped the Design

2.1 Identifiers and core entities

Core entity tables have straightforward dependencies. For manufacturer(manufacturer_id, name, created_at) we have manufacturer_id → (name, created_at). supplier(supplier_id, supplier_code, name, created_at) has two candidate keys: supplier_id → (supplier_code, name, created_at) and supplier_code → (supplier_id, name, created_at). category(category_id, name) similarly treats both category_id and name as keys. user_account(user_id, username, password_hash, role, manufacturer_id, supplier_id, created_at) has user_id → all other attributes and username → all other attributes. These dependencies motivated surrogate primary keys plus UNIQUE constraints on natural identifiers such as supplier_code, category name and username.

2.2 Ingredients, composition and supplier pricing

For ingredients, ingredient(ingredient_id, name, is_compound, created_at) is governed by ingredient_id → (name, is_compound, created_at). Compound ingredients are built from materials, so ingredient_material(parent_ingredient_id, material_ingredient_id, qty_oz) uses the composite key (parent_ingredient_id, material_ingredient_id) → qty_oz. This decomposition avoided storing variable-length ingredient lists in ingredient, which would violate full functional dependency.

Supplier capability and pricing required a more deliberate decomposition. Conceptually, for each (supplier_id, ingredient_id) there may be multiple pack sizes and prices that change over time. Instead of putting pack_size_oz and unit_price directly on supplier_ingredient, the schema splits this into supplier_ingredient(supplier_id, ingredient_id) with key (supplier_id, ingredient_id) and supplier_formulation(formulation_id, supplier_id, ingredient_id, pack_size_oz, unit_price, effective_from, effective_to) with formulation_id → all other attributes and, at the business level, (supplier_id, ingredient_id, effective_from) → (pack_size_oz, unit_price, effective_to). This decomposition removes anomalies caused by time-varying prices and enables a trigger to prevent overlapping date ranges.

A similar pattern is used for supplier_formulation_material(formulation_id, material_ingredient_id, qty_oz), where (formulation_id, material_ingredient_id) → qty_oz. Functional dependencies here directly motivated the use of separate intersection tables rather than denormalised repeating groups.

2.3 Recipes, inventory and production

For finished products, product_type(product_type_id, manufacturer_id, category_id, product_code, name, standard_batch_units, created_at) has product_type_id → all other attributes and the natural-key dependency (manufacturer_id, product_code) → (name, category_id, standard_batch_units). The DDL therefore declares product_type_id as the primary key and enforces a UNIQUE constraint on (manufacturer_id, product_code).

Recipe versioning is handled by recipe_plan(recipe_plan_id, product_type_id, version_number, created_at, notes) with recipe_plan_id → (product_type_id, version_number, created_at, notes), and by recipe_plan_item(recipe_plan_item_id, recipe_plan_id, ingredient_id, qty_oz_per_unit, step_number) with recipe_plan_item_id → (recipe_plan_id, ingredient_id, qty_oz_per_unit, step_number) and the logical dependency (recipe_plan_id, step_number) → (ingredient_id, qty_oz_per_unit). This separation keeps each row tied to exactly one product version and one ingredient step.

Inventory tables follow the same pattern. ingredient_batch(ingredient_batch_id, ingredient_id, supplier_id, supplier_batch_id, lot_number, quantity_oz, on_hand_oz, unit_cost, expiration_date, received_at) has ingredient_batch_id → all other attributes and lot_number → all other attributes, justifying a surrogate primary key plus a UNIQUE lot_number. product_batch(product_batch_id, product_type_id, manufacturer_id, product_lot_number, produced_units, batch_cost, unit_cost, expiration_date, created_at) uses product_batch_id and product_lot_number as keys, and product_batch_consumption(product_batch_consumption_id, product_batch_id, ingredient_batch_id, qty_oz) has product_batch_consumption_id → (product_batch_id, ingredient_batch_id, qty_oz). staging_consumption is a transient table used during batch creation; its surrogate key staging_id avoids composite-key anomalies.

3. Normalization Summary

The Normalization Table was built directly from the implemented schema. It records, for each base table, the highest normal form justified by the functional dependencies and declared keys. All tables are at least in Third Normal Form, and in fact every listed base table is in Boyce–Codd Normal Form (BCNF).

For example, supplier has two candidate keys (supplier_id and supplier_code) and all non-trivial dependencies have a key on the left, so it is in BCNF. product_type has a surrogate key product_type_id and an alternate key (manufacturer_id, product_code); again, every non-trivial dependency is from a key, so product_type is in BCNF. Intersection tables such as ingredient_material, supplier_ingredient, supplier_formulation_material and product_batch_consumption have composite primary keys and no additional descriptive attributes, so every attribute is fully dependent on the whole key. The summary table in Appendix A lists each table and its normal form; none are below 3NF, satisfying the project requirement.

4. Constraints Beyond Basic Table Definitions

4.1 Constraints implemented inside the DBMS

The DDL already uses primary keys, foreign keys, UNIQUE constraints, NOT NULL and CHECK constraints, as described in CONSTRAINTS_DOCUMENTATION.md (for example, positive quantities, non-negative costs, and required identifiers). Several important business rules, however, required more than column-level constraints and were implemented using triggers, stored procedures and views.

Triggers (TRIGGERS_DOCUMENTATION.md) enforce cross-row and temporal rules. trg_ingredient_batch_before_insert enforces the 90-day minimum shelf-life rule and initialises on_hand_oz, while also generating a unique lot_number. trg_supplier_formulation_before_insert prevents overlapping effective date ranges for each (supplier_id, ingredient_id) pair. trg_ingredient_material_before_insert ensures one-level composition (no grandchildren and no self-references), and trg_product_batch_consumption_* triggers validate that consumed lots exist, are not expired and never drive on_hand_oz negative.

Stored procedures (PROCEDURES_DOCUMENTATION.md) encapsulate multi-step invariants. sp_record_product_batch creates a product batch in a single transaction, checks that requested production is a positive multiple of standard_batch_units, verifies that all lots in staging_consumption have sufficient on_hand_oz and are not expired, checks do_not_combine for incompatible ingredient pairs, computes batch_cost and unit_cost, and finally inserts product_batch and product_batch_consumption rows. sp_trace_recall and sp_compare_products_incompatibility implement traceability and incompatibility analysis using consistent SQL logic. Views such as v_report_onhand, v_nearly_out_of_stock, v_almost_expired and v_active_formulations summarise inventory and pricing state so that both the application and ad-hoc users see the same business logic.

4.2 Constraints implemented only in application code

Some rules were intentionally left to the Python application because they are about user interaction or cannot be conveniently expressed in SQL. The role system is the most important example. user_account stores role, manufacturer_id and supplier_id, but the rule “each user holds exactly one role” is enforced in the login and user-creation flows rather than through conditional CHECK constraints. Similarly, the Viewer role is implemented as a read-only menu in viewer_actions.py; all three roles connect to the database with the same MySQL account, so the DBMS cannot distinguish them at privilege level.

Manufacturer and Supplier functions (MANUFACTURER_FUNCTIONS.md, SUPPLIER_FUNCTIONS.md) also perform additional input validation beyond what the DB enforces. For example, create_product_type() rejects empty product names, trims whitespace, and asks for confirmation before deleting a product_type that has no history, even though referential integrity would allow deletion. Supplier-side functions preview the impact of removing ingredients or formulations and require an explicit “yes” confirmation. FEFO lot selection is implemented in auto_select_lots_fefo() in the application: it chooses candidate ingredient_batch rows ordered by expiration_date and populates staging_consumption, while the DB-level triggers and procedures enforce safety (no expired or overdrawn lots) regardless of the selection strategy.

Finally, some representational rules are handled in code for simplicity. For the symmetric do_not_combine relation, the application always inserts pairs with ingredient_a < ingredient_b before calling INSERT, relying on the database only for uniqueness of the pair. Likewise, deletion policies for higher-level entities (such as refusing to delete a product_type if it has any recipe or batch history) are implemented in the manufacturer menus, even where the foreign-key configuration would technically allow cascading deletes. This division keeps intrinsic data validity in the DBMS and UI or policy decisions in the application.

5. Conclusion

In summary, the final schema is driven by explicit functional dependencies, normalized to BCNF as confirmed in Normalization_Table.docx, and supported by a layered constraint strategy. Keys, foreign keys, CHECK constraints, triggers, stored procedures and views allow the DBMS to enforce core business rules such as FEFO usage, non-overlapping supplier pricing, safe consumption of lots and clean composition hierarchies. Application code is used mainly for role management, richer input validation and user-facing safety checks. This balance meets the project goal of using the DBMS aggressively for constraint enforcement while still enabling a practical, role-based command-line application.

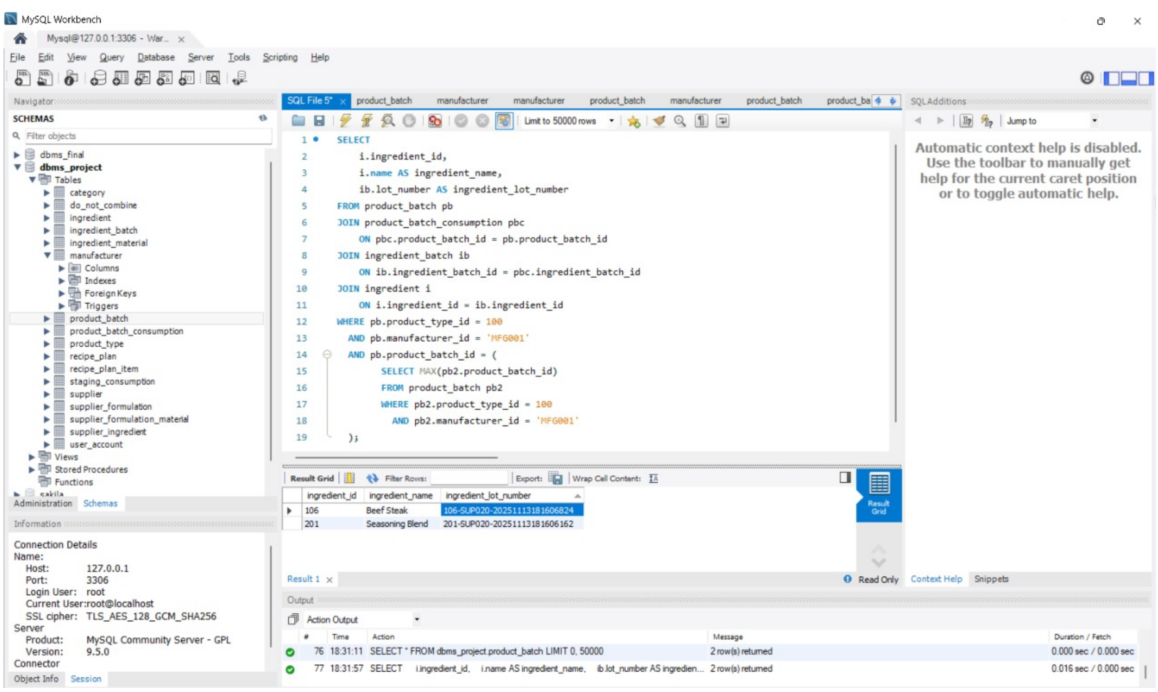
Appendix A – Normalization Summary

Table	Normal Form	Notes
manufacturer	BCNF	Simple entity; all attributes depend only on the PK.
supplier	BCNF	Entity with alternate key; all dependencies from superkeys.
category	BCNF	PK plus unique name; no transitive dependencies.
user_account	BCNF	PK with unique username; role depends on key.
ingredient	BCNF	PK; name unique; all attributes depend on ingredient_id.
ingredient_material	BCNF	Intersection table; composite PK; qty depends on whole key.
supplier_ingredient	BCNF	Intersection; composite key defines full dependency.
supplier_formulation	BCNF	PK + logical natural key; all non-key attributes depend on formulation.
supplier_formulation_material	BCNF	Composite PK; qty depends only on entire key.
do_not_combine	BCNF	Composite key-only table; no non-key attributes.
product_type	BCNF	PK + alternate natural key; all fields depend on product type.
recipe_plan	BCNF	PK; version_number forms logical alternate key.
recipe_plan_item	BCNF	PK; (recipe_plan_id, step_number) logical

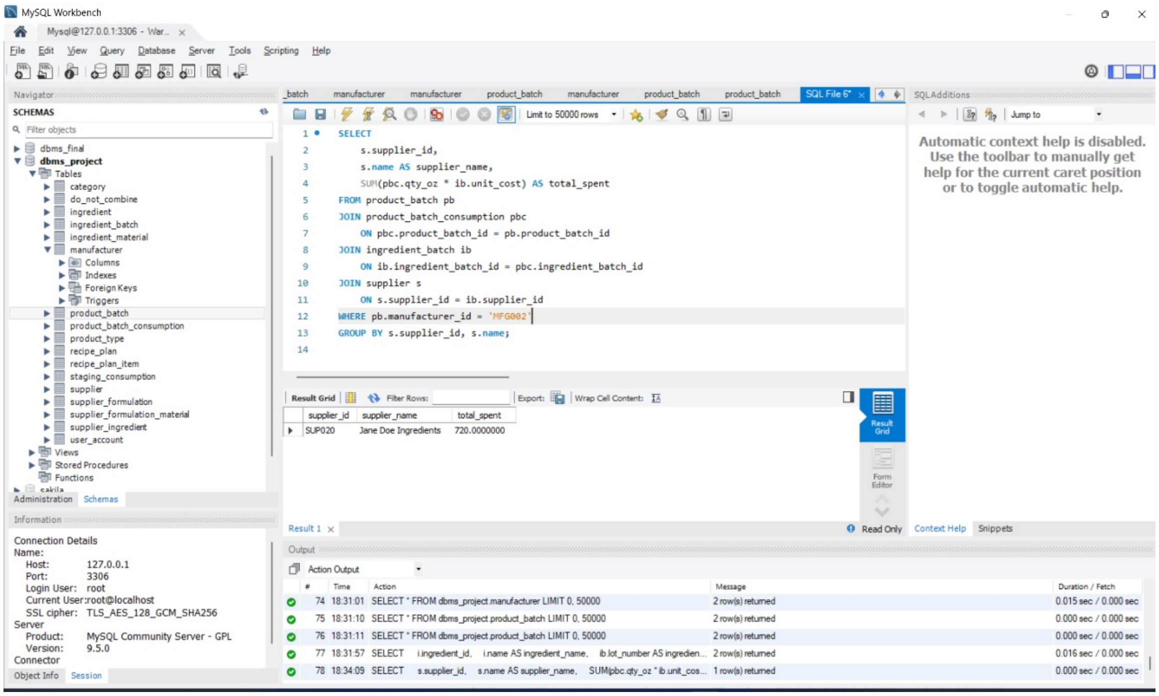
		candidate key.
ingredient_batch	BCNF	PK; lot_number alternate key; no transitive dependencies.
staging_consumption	BCNF	Surrogate PK staging_id; transient staging table.
product_batch	BCNF	PK; product_lot_number alternate key; attributes depend on batch.
product_batch_consumption	BCNF	PK; qty depends only on row key.

Appendix B – Screenshots of working queries

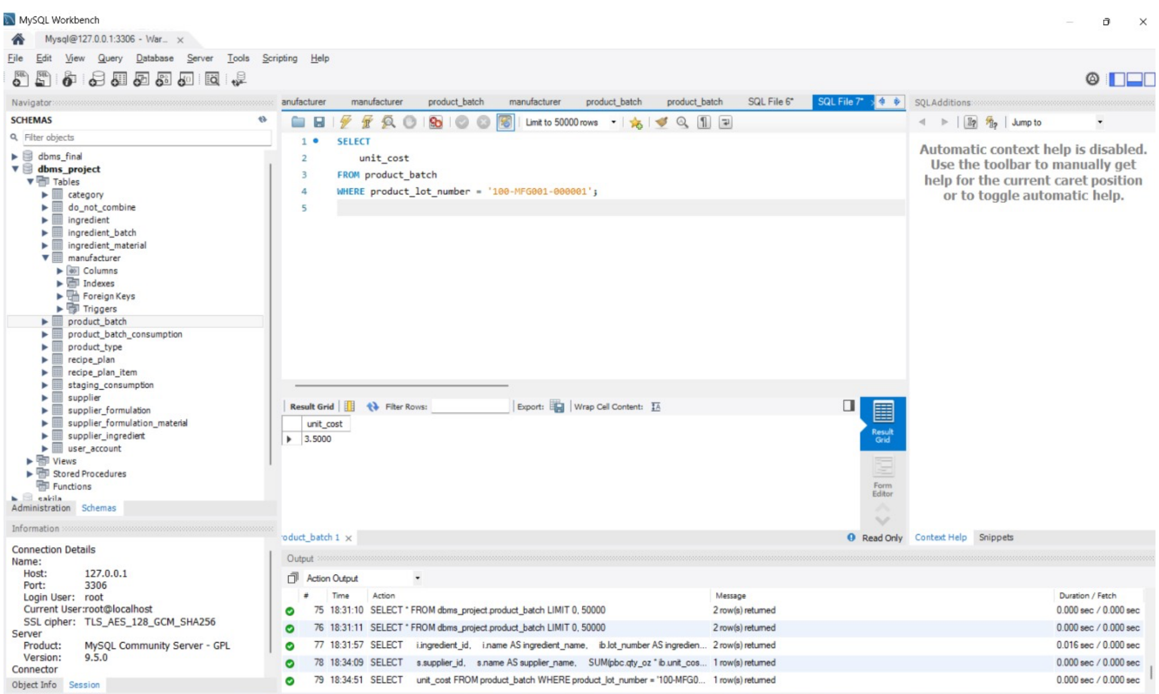
Query 1: List the ingredients and the lot number of the last batch of product type Steak Dinner (100) made by manufacturer MFG001.



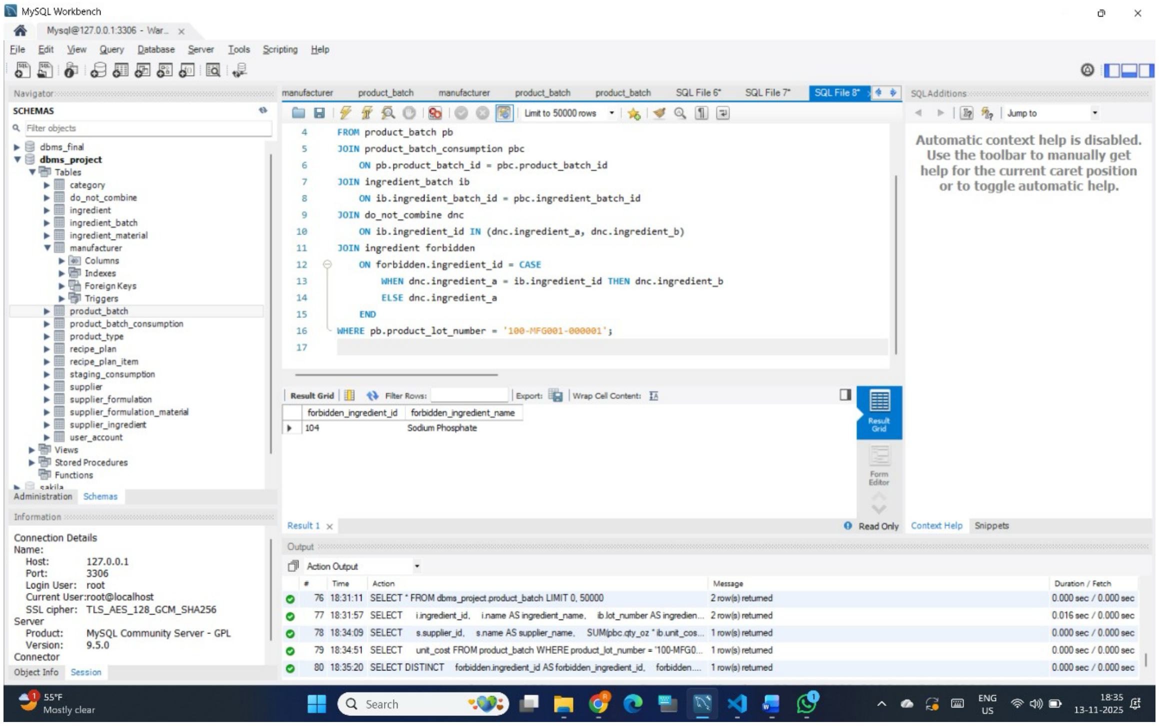
Query 2: For manufacturer MFG002, list all the suppliers that they have purchased from and the total amount of money they have spent with that supplier.



Query 3: For product with lot number 100-MFG001-000001, find the unit cost for that product.



Query 4: Based on the ingredients currently in product lot number 100-MFG001-000001, what are all ingredients that cannot be included (i.e. that are in conflict with the current ingredient list)



Query 5: Which manufacturers have supplier James Miller (21) NOT supplied to?