Design and Normalization Database on SuperMarket Aisle Management

Fadhla Mohamed Mutua - SM3201434

June 19, 2025

Abstract

This report presents the conceptual and logical design of a supermarket aisle management database. The system manages supermarkets, aisles, items, producers, distances, and error logging for items. Explained are the SQL operations to be used (see accompanying code), Entity-Relationship (ER) diagrams, redundancy and normalization steps up to Third Normal Form (3NF).

Contents

1 Introduction **3**

2 Conceptual Schema **5**
2.1 Conceptual Schema Table 5
2.2 Entity-Relationship Diagram 6
2.3 Redundancy Analysis 8

3 Logical Schema **9**

4 Normalization **10**
4.1 First Normal Form (1NF) 10
4.2 Second Normal Form (2NF) 10
4.3 Third Normal Form (3NF) 10

5 Conclusion **11**

1 Introduction

This report is on a database system that supports supermarkets, and, producers by logging errors generated by automated triggers.

Assumption: Multiple supermarkets can have an aisle with the same name as such using AisleID will make it so that no matter which aisle name, the AisleID corresponds to a specific SuperMarket

Note: The relation between Aisle and SuperMarket on the pdf is called Has Aisle while on the accompanying code, it is AisleSuperMarket

2. SQL Operations Overview

This section outlines all the core SQL operations implemented in the database system, explaining their roles and how they support the domain logic of aisle management and compliance.

2.1 Creation of Tables

Operation: Schema Definition and Initialization

The first set of operations defines the relational structure and schema of the database. Tables such as `SuperMarket`, `Aisle`, `Item`, `Producer`, `Distance`, `Contain`, `Manufactured_By`, `ErrorMessages`, and `ItemLogErrors` are created using `CREATE TABLE`. These statements specify primary and foreign keys, data types, and constraints, establishing the foundation for data storage.

2.2 Triggers

Trigger: `trg_check_Item_Aisle_count`

This `BEFORE INSERT` trigger ensures that an item is not placed in multiple aisles within the same supermarket. If violated, it raises an SQL exception. This guarantees one-to-one mapping of item to aisle per supermarket.

Trigger: `trg_log_item_wrong_aisle`

This `AFTER INSERT` trigger validates whether the item is placed in a logically correct aisle using a function. If non-compliant, it logs the violation via an error ID, using standardized messages and timestamping.

Trigger: `ReturnItem`

Monitors inserted items for expiration. If an item is expired, it checks if the distance to the producer is within a threshold or if the item is non-perishable. It then logs whether the item should be returned or thrown away.

2.3 Views

View: `FullItemDetails` — Comprehensive joined result showing each item, its aisle, supermarket, and producer.

View: `ItemWithProducers` — Displays item and associated producer information.

View: `ProducerSuperMarketDistance` — Maps distances between producers and supermarkets.

View: `WhereToStore` — Matches storage types of items with aisle names for placement validation.

View: `ItemErrorDetails` — Provides detailed logs of item errors including messages and discard flags.

2.4 Stored Functions

Function: `fn_validate_aisle_compliance`

Enforces domain rules for aisle placement. Returns a human-readable error if non-compliant, otherwise `NULL`.

Function: `fn_insert_into_error_message`

Inserts a new message into `ErrorMessages` table if it doesn't exist and returns the `ErrorID`.

Function: `fn_suggest_correct_aisle`

Suggests the most appropriate `AisleID` for an item in a specific supermarket based on category and storage.

2.5 Stored Procedures

Procedure: `pr_insert_item_log` — Inserts item error logs with timestamp and error info.

Procedure: `AddItemToAisle` — Adds an item to an aisle, verifying that the aisle belongs to the specified supermarket.

Procedure: `RemoveItemFromSuperMarket` — Deletes an item from all aisles in a supermarket.

Procedure: `LogItemError` — Logs a manual item error using message and item details.

Procedure: `CleanExpiredItems` — Deletes expired and perishable items from the `Contain` table.

Procedure: `CheckItemCompliance` — Checks compliance and optionally suggests correct placement.

Procedure: `sp_check_item_placement` — Iterates through items, validates placement, and logs non-compliance.

Procedure: `sp_expiration_check` — Logs expiration-related issues including producer and discard status.

2.6 Scheduled Events

Event: `ev_daily_item_placement_check` — Daily trigger to run `sp_check_item_placement`.

Event: `ev_daily_expiration_and_cleanup` — Daily cleanup of expired items and logging of expiration status.

Event: `ev_daily_expiration_process` — Wrapper event managing expiration and cleanup automation.

2.7 Queries

Query: Historical Error Analysis

```
SELECT
    em.ErrorMessage,
    i.ItemName, i.ItemStorageType,
    a.AisleID, a.AisleName AS IncorrectAisle,
    le.LogTime,
    le.ToBeThrown
FROM ItemLogErrors le
JOIN Item i ON le.ItemID = i.ItemID
JOIN Aisle a ON le.AisleID = a.AisleID
LEFT JOIN ErrorMessages em ON le.ErrorID = em.ErrorID
ORDER BY le.LogTime DESC;
```

Provides a full log of item placement or expiration violations with contextual information.

2 Conceptual Schema

2.1 Conceptual Schema Table

Entities and Relationship	**Cardinality (Relational)**
Producer — Distance — SuperMarket	1:N : 1:N
SuperMarket — Has_Aisle — Aisle	1:N : 1:1
Aisle — Contains — Item	1:1 : 0:N
Producer — Manufactured_By — Item	1:N : 1:1
ItemLogErrors — Logs_Item — Item	0:N : 0:1
ItemLogErrors — Logs_Aisle — Aisle	0:N : 0:1
ItemLogErrors — Logs_ErrorMessage — ErrorMessage	1:N : 1:1

Table 1: Conceptual Schema Relations with Precise Cardinalities

Relationship	Description
Distance	Connects producers to supermarkets with distance information, useful for calculating transportation and supply chain logistics (In this case if to or not to throw an item.
Has_Aisle	Associates aisles to supermarkets, indicating the layout structure within each store.
Contains	Represents items stored within aisles, linking inventory to location.
Manufactured_By	Links items to their producers, allowing traceability of product origins.
Logs_Item	Logs errors related to specific items, facilitating error tracking and auditing.
Logs_Aisle	Logs errors associated with specific aisles, aiding in pinpointing storage issues.
Logs_ErrorMessage	Associates error logs with standardized error messages, enabling consistent error reporting.

Table 2: Conceptual Schema Relationships and Their Description

2.2 Entity-Relationship Diagram

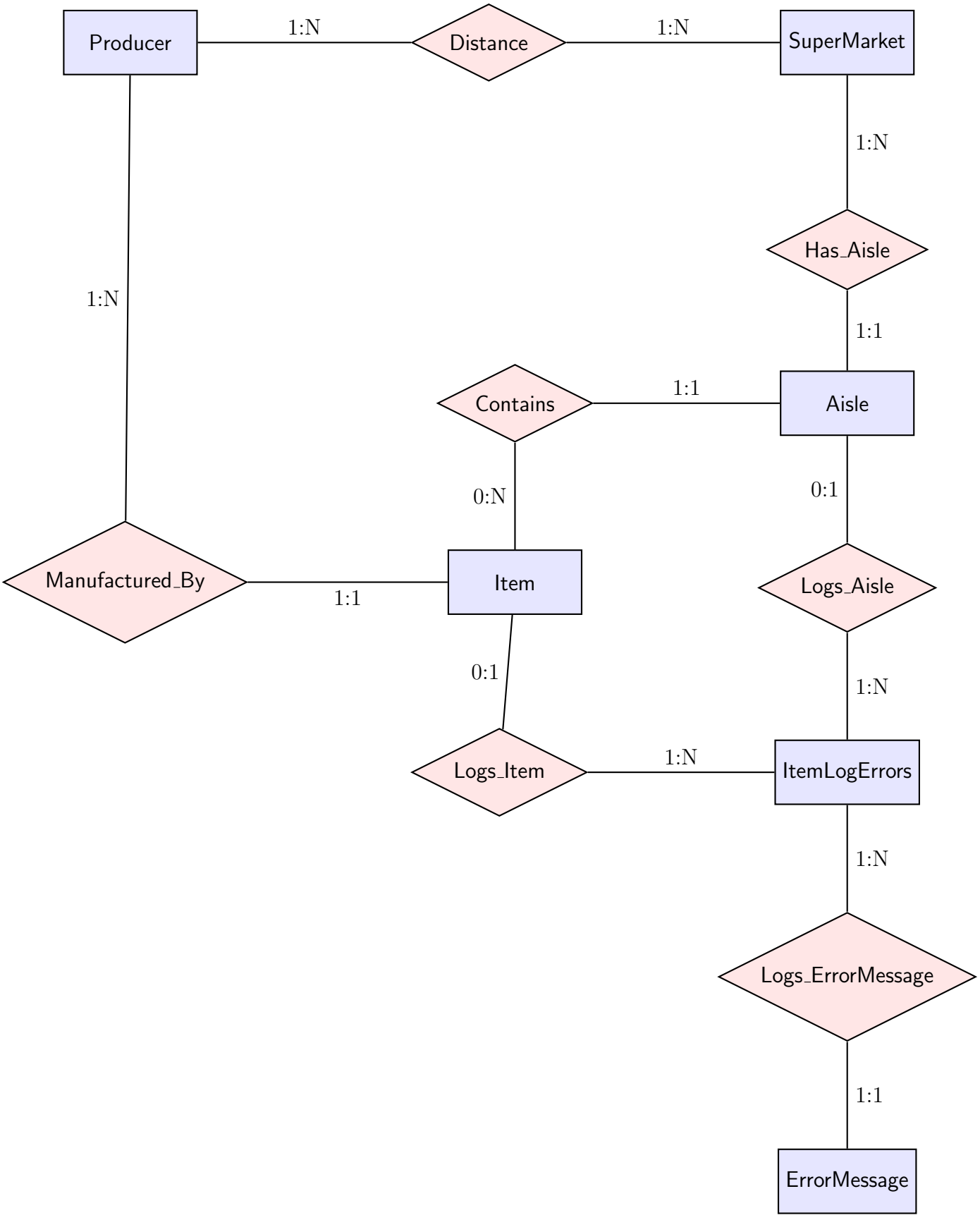

Figure 1: Entity-Relationship Diagram for Aisle Management

2.3 Redundancy Analysis

- **ItemLogErrors** has multiple binary relationships with `Item`, `Aisle`, and `ErrorMessage`.
- This could be replaced by a ternary relationship directly linking `ItemLogErrors` with `Item`, `Aisle`, and `ErrorMessage`.
- Since `ItemLogErrors` is trigger-generated, keeping the separate relationships together eases querying.
- No redundant entities or relationships for `Producer`, `SuperMarket`, `Aisle`, `Item`, or `Distance`.

3 Logical Schema

Table	**Attributes**	**PK**	**FK**
Producer	ProducerID, ProducerName, ProducerLocation	ProducerID	-
SuperMarket	SuperMarketID, SuperMarketName, SuperMarketLocation	SuperMarketID	-
Aisle	AisleID, AisleName	AisleID	-
Item	ItemID, ItemName, ItemCategory, ItemStorageType, ItemPerishable, ItemExpirationDate	ItemID	-
Manufactured_By	ItemID, ProducerID	ItemID	ProducerID
Distance	ProducerID, SuperMarketID, Distance	ProducerID, SuperMarketID	ProducerID, SuperMarketID
Contain	AisleID, ItemID	AisleID, ItemID	AisleID, ItemID
Has_Aisle	AisleID, SuperMarketID,	AisleID, SuperMarketID,	AisleID, SuperMarketID,
ItemLogErrors	ErrorLogID, ItemID, AisleID, ErrorID, LogTime, ToBeThrown	ErrorLogID	ItemID, AisleID, ErrorID
ErrorMessage	ErrorID, ErrorMessage	ErrorID	-

Table 3: Logical Schema Tables with Underlined Primary Keys

4 Normalization

4.1 First Normal Form (1NF)

- All attributes are atomic and indivisible.
- No repeating arrays exist as ErrorMessage message is a String.

4.2 Second Normal Form (2NF)

- No partial dependency exists on composite keys.

4.3 Third Normal Form (3NF)

- No transitive dependencies are present.
- All non-key attributes depend only on the primary key.

Entities and Relationship	**Cardinality (can/(or) must : quantity : can/(or) must : quantity)**
Producer — Distance — SuperMarket	1:N : 1:N
SuperMarket — Has_Aisle — Aisle	1:N : 1:1
Aisle — Contains — Item	1:1 : 0:N
Producer — Manufactured_By — Item	1:N : 1:1
ItemLogErrors — Logs_Item — Item	0:N : 0:1
ItemLogErrors — Logs_Aisle — Aisle	0:N : 0:1
ItemLogErrors — Logs_ErrorMessage — ErrorMessage	1:N : 1:1

Table 4: Normalized Conceptual Schema with precise cardinalities

Table	Attributes	PK	FK
Producer	ProducerID, ProducerName, ProducerLocation	ProducerID	-
SuperMarket	SuperMarketID, SuperMarketName, SuperMarketLocation	SuperMarketID	-
Aisle	AisleID, AisleName	AisleID	-
Item	ItemID, ItemName, ItemCategory, ItemStorageType, ItemPerishable, ItemExpirationDate	ItemID	-
Manufactured_By	ItemID, ProducerID	ItemID	ProducerID
Distance	ProducerID, SuperMarketID, Distance	ProducerID, SuperMarketID	ProducerID, SuperMarketID
Contain	AisleID, ItemID	AisleID, ItemID	AisleID, ItemID
Has_Aisle	AisleID, SuperMarketID,	AisleID, SuperMarketID,	AisleID, SuperMarketID,
ItemLogErrors	ErrorLogID, ItemID, AisleID, ErrorID, LogTime, ToBeThrown	ErrorLogID	ItemID, AisleID, ErrorID
ErrorMessage	ErrorID, ErrorMessage	ErrorID	-

Table 5: Normalized Logical Schema

5 Conclusion

The normalization process confirms the schema adheres to 3NF, supporting automated error tracking without introducing redundancy.