Table of Contents

Introduction

This project involves the **design** and **implementation** of a Distribution Center Inventory Relational Database Management System (**RDBMS**). The primary objective of the system is to effectively manage the storage and movement of products within a distribution network. It tracks inventory levels at the distribution center, records vendor transactions, and supports the delivery of goods to affiliated stores, warehouses, and end customers. The system is built using a normalized schema, adhering to ACID principles to ensure consistency across vendor transactions, inventory updates, and delivery records. Key entities include **Products**, **Vendors**, **Deliveries**, **Locations**, **Stores**, **Warehouses**, and **Customers**.

Project Overview

This project focuses on the development of a Distribution Center Inventory Relational Database Management System (RDBMS). The system is designed to manage and track the flow of products from a central distribution center to multiple delivery endpoints, including stores, warehouses, and customers. The primary goal is to provide a reliable and structured way to monitor product inventory, vendor transactions, and scheduled deliveries, as well as ensuring that stock levels are accurate and traceable at all times. The system supports the addition of vendors and products, records the quantity and timing of incoming stock, and updates inventory levels as items are delivered or distributed to various destinations. Key components of the database include entities such as **Products**, **Vendors**, **Deliveries**, **Customers**, **Stores**, **Warehouses**, and **Locations**. A shared Location model is used to efficiently handle address data across multiple entity types. The system also ensures data consistency and integrity through normalized relationships and transactional operations. This RDBMS provides a foundation for inventory control, helping distribution centers operate more efficiently and make informed logistical decisions based on real-time data.

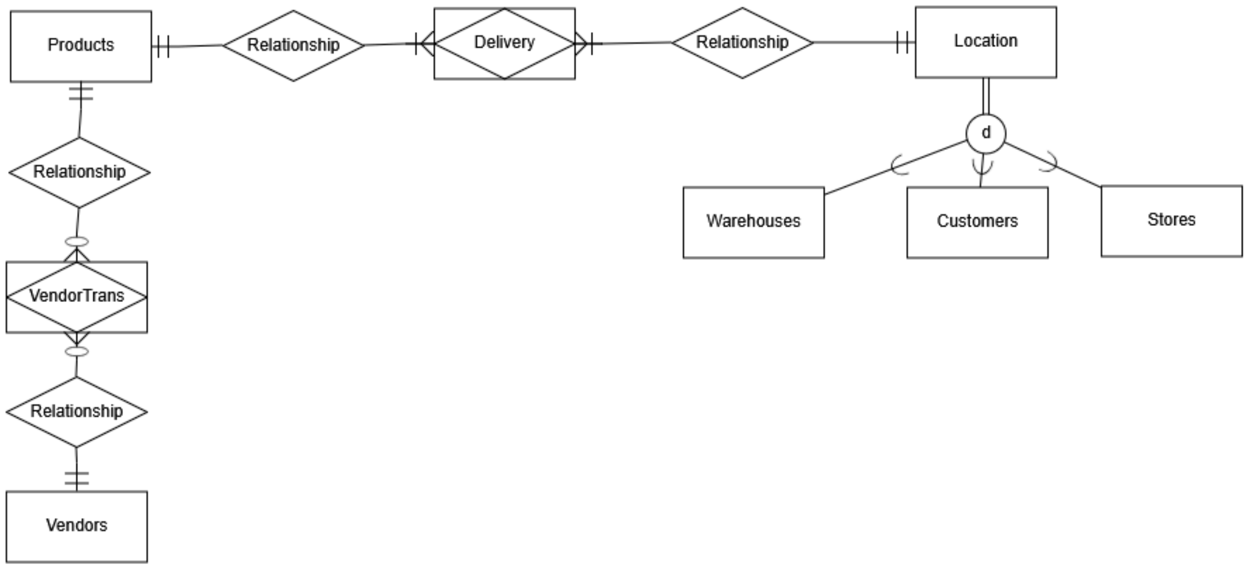
Implementation Details

The implementation of the Distribution Center Inventory Management System was done using a relational database. The system was built using MSSQL, with tables created by first understanding what information I needed to store, and then understanding their functional requirements.

When creating a distribution center, I had to understand what their purpose was:

*"**Distribution centers** are more like all-in-one logistics operations that store, pick, pack and ship products to fulfill customer orders — either to retail locations or directly to individual consumers."*

With this definition in mind, I was able to design my core entities, **Vendors**, **Products**, **Warehouses**, **Stores**, **Customers**. After establishing my core entities, I needed to distinguish the relationships between them. To do that, I've created a conceptual model to visualize this relationship.



ERD Explanation

The Entity Relationship Diagram for the Distribution Center Inventory Management System represents the core data structure used to track the movement and storage of products. It consists of 8 main entities: **Products**, **Vendors**, **VendorTransactions**, **Deliveries**, **Customers**, **Stores**, **Warehouses**, and **Locations**.
Products are supplied by **Vendor**.
The relationship between **Products** and **Vendors** is captured by the **VendorTransaction** table, which logs transactions made between Vendors and the Distribution Center.

The **Deliveries** table is used to track the distribution of products from the distribution center to external locations. Each delivery includes the product being shipped, the number of units, the source and destination locations, delivery status and delivery date. This table helps manage inventory levels and ensures accurate delivery tracking.

The system uses the **locations** table as a supertype, with **Warehouses**, **Stores**, and **Customers** as the subtype. This helps to avoid redundancy in storing address information for **Warehouses**, **Stores**, and **Customers**. Each of these three tables reference the locations “locationID”, which links to the street address, zipcode, state, and location type (customer/store/warehouse). This design makes it easier to manage and update address data across all location based entities.

Warehouses represent physical storage facilities.

Stores are retail establishments that receive products from the distribution center.

Customers represent end users or members who receive direct deliveries.

Products - ProductID, Name, SKU, Price, Quantity, SKU, UPC_Number, Category, ProductDescription, VendorID

Vendor - VendorID, VendorName, Number, Email

VendorTransaction - TransactionID, ProductID, VendorID, UnitsSold, TransactionDate

Delivery - DeliveryID, SourceLocation, DestinationLocation, ProductID, Units, DeliveryStatus, DateDelivered, TrackingCode

Location - LocationID, Address, Zipcode, State, LocationType

Warehouses - WarehouseID, LocationID

Customers - CustomerID, LocationID, FirstName, LastName, Gender, Email, Number

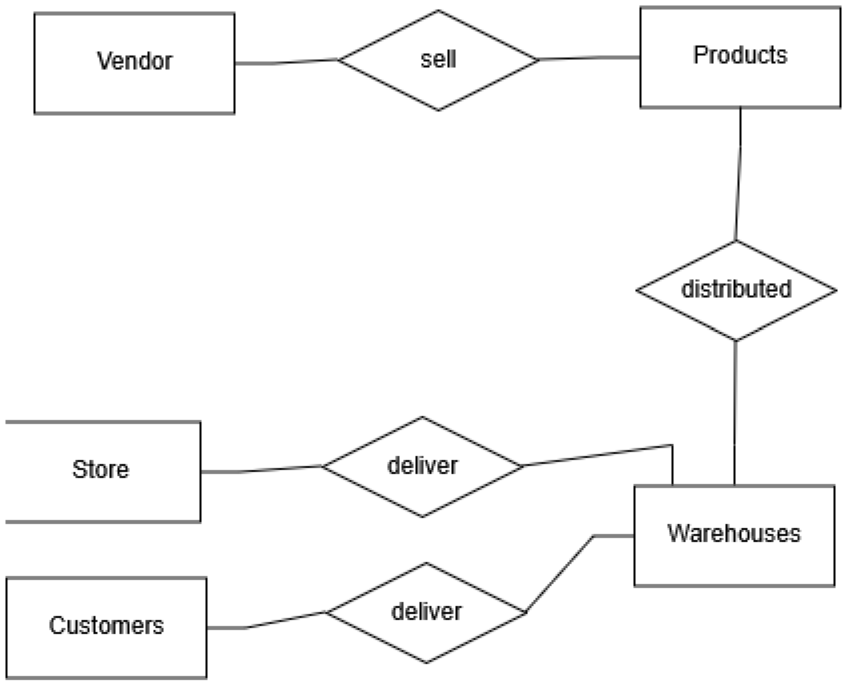
Stores - StoreId, LocationID, ManagerName

Challenges and Solutions

- My first challenge was, “How do I distinguish the destination of the delivery between the choices of store, warehouse, or customer?”
    - This was complicated due to my initial database design. It didn’t allow for the delivery of product unless it was made to or from a warehouse.
    -

deliveryID	ProductID	warehouseID	Quantity
0001	256	20	300

    - Product with the ID 256 is being delivered to a warehouse with the ID 20, delivering 300 units of said product.
    -



- However, this made me realize the flaw in the design, “how would I represent deliveries to a store or customer?”
DeliveryID	ProductID	WarehouseID	StoreID	CustomerID	Quantity
0002	232	Null	Null	200	50
0010	40	Null	32	Null	100
    - While this solves the issue of delivering to any location, it creates several problems:
        - WarehouseID, StoreID, and CustomerID are all trying to represent a single delivery destination.
            - This means for every row, i’ll have at least 2 Null Values
                - This makes queries complex as I’ll always need *IS NOT NULL* within the logic
                - Also makes joins more complex and harder readability
        - Lack of future-proofing, if there’s a need to add more destination types, I’d have to continue adding more columns and check for NULLS. No Scalability.
    - To solve this, I've created a **location** table to reduce this redundancy. Making location a supertype, with **warehouses**, **stores**, and **customers** as subtypes.
        - This reduces redundancy
        - Keeps my schema flexible
        - Makes joins cleaner
        - Makes the database scalable
DeliveryID	ProductID	LocationID	Quantity	,Status
0002	232	10	100	Delivered
0010	40	20	50	En route

Conclusion

This project focused on designing a Relational Database Management System for a distribution center to track products from vendors to stores, warehouses, and customers. The database was built using clear relationships between key entities like Products, Vendors, Deliveries, and Locations.

By organizing data into separate, related tables and avoiding repetition, the system ensures accuracy and is easy to maintain.

Overall, this database provides a solid structure for managing inventory and deliveries efficiently. It can be expanded in the future to support features like real-time tracking by working with the distribution centers inhouse software or automated restocking.