Text

Description automatically generatedA picture containing logo

Description automatically generated

**PUSL2019 Information Management Retrieval**

**POS Systems’s Report 2024/25**

Group No: BC

Group Members: 10

|  |  |  |  |
| --- | --- | --- | --- |
| **Teams** | **ID** | **Name** | **Task** |
| TEAM SQL | 10952358 | I B Y D Irugalbandara [AI Lead] | * Database Tables * Sample Data * Database Diagram * Data Validation & Constraints * C# App - Tables & Joints * C# App – In App Data Validation |
| 10953783 | R J Ganegame [Backend Lead] | * Procedures * Triggers * C# App - UI * C# App - Backend * C# App - Input Data Validation |
| 10953476 | R P W D G Rajapaksha | * Functions * Views |
| TEAM DOC | 10953336 | R Y M M P Rajakaruna | * Normalization * Finalized Report |
| 10953524 | H G L D K Hathnapitiya | * Data Security * Risk Analysis * Complete Database Code |
| 10953572 | H H D J V Herath | * Critical analysis * References and TOC |
| TEAM  DESING | 10953233 | G Kishalan | * ER/EER Diagram * Presentation Slide Design |
| 10953377 | G Sathushan | * Relational Mapping |
| 10953332 | A R Ihsaan | * Data Dictionary * Database Backup |
| 10953249 | M V I Senadheera | * Introduction to the scenario * Back cover * Workload Matri |

TABLE OF CONTENT

[1. Inroduction 3](#_Toc186967605)

[2. ER/EER DIAGRAM. 5](#_Toc186967606)

[2.1 Assumptions for the ER/EER Diagram (Microsoft, 2024) 6](#_Toc186967607)

[3. relational mapping 7](#_Toc186967608)

[4. normalization. 8](#_Toc186967609)

[5. data dictionary. 15](#_Toc186967610)

[6. DATABASE DESIGN 20](#_Toc186967611)

[6.1 custommer management 20](#_Toc186967612)

[6.2 employee management 22](#_Toc186967613)

[6.3 “report” generation 27](#_Toc186967614)

[6.4 product and stock management 28](#_Toc186967615)

[6.5 discount management 39](#_Toc186967616)

[6.6 sales management 41](#_Toc186967617)

[6.7 database diagrame 46](#_Toc186967618)

[7. procedures & triggers 46](#_Toc186967619)

[7.1 Stored Procedures 46](#_Toc186967620)

[7.2 Triggers 46](#_Toc186967621)

[8. functions and views 47](#_Toc186967622)

[8.1 calculate total sales 47](#_Toc186967623)

[8.2 calculate product sales 48](#_Toc186967624)

[8.3 get product availability 49](#_Toc186967625)

[8.4 get discount price 50](#_Toc186967626)

[8.5 customer sales summery 51](#_Toc186967627)

[8.6 product sales overview 52](#_Toc186967628)

[8.7 employee role summery 53](#_Toc186967629)

[8.8 sales transaction details 54](#_Toc186967630)

[9. data security measures 55](#_Toc186967631)

[10. critical analysis 58](#_Toc186967632)

[10.1 Highlight the 'key features' of our solution [strengths] 58](#_Toc186967633)

[10.2 Identify and explain the 'areas that need improvements' [weaknesses] 60](#_Toc186967634)

[10.3 Future implementation 63](#_Toc186967635)

[11. appendices 66](#_Toc186967636)

[11.1 creat tables 66](#_Toc186967637)

[11.2 Sample Data 72](#_Toc186967638)

[11.3 Function And Views 90](#_Toc186967639)

[11.4 Procedures & Triggers 96](#_Toc186967640)

[11.5 project links 103](#_Toc186967641)

[12. pos system screenshots 104](#_Toc186967642)

[12.1 applications 104](#_Toc186967643)

[12.2 project management 114](#_Toc186967644)

[References 116](#_Toc186967645)

# Inroduction

Supermarkets are complex systems that manage various operational aspects, from inventory control to customer engagement. With rising customer expectations and an increasingly competitive retail landscape, it has become essential to adopt advanced technological solutions. This project focuses on developing a customized Point of Sale (POS) system tailored to the needs of a supermarket.

The proposed system is not just a cash register replacement but an all-encompassing solution designed to handle sales, stock management, discounts, and reporting. By automating key processes and providing real-time insights, the system aims to improve operational efficiency, reduce errors, and empower management to make informed decisions for business growth.

### overview of the super market’s pos system requirements.

The supermarket imagines a smart and reliable **POS system** (EDB, 2024) that brings together several essential features to meet its unique needs and streamline operations.

1. Customer Management

The POS system will store and manage customer information, enabling personalized interactions. Key functionalities include,

* Recording details such as names, contact numbers, and purchase histories.
* Supporting loyalty programs where customers can earn and redeem points.
* Enabling targeted promotions based on purchase trends, such as offering discounts to high-spending customers.

2. Stock Management

Inventory control is a critical aspect of retail operations. The system will,

* Maintain real-time inventory levels to avoid stockouts or overstocking.
* Provide notifications for low-stock or expiring items to prevent wastage.
* Allow categorization of products by attributes like brand, category, and expiry date for efficient tracking.

3. Discount and Promotions Management

To attract and retain customers, discounts and promotions must be managed seamlessly by,

* Automatically apply discounts based on predefined rules, such as percentage discounts or "Buy 1 Get 1 Free" offers.
* Allow staff to update discount policies quickly during promotional events.
* Ensure discounts are accurately reflected in the final billing.

Sales Management.

Sales processing is at the heart of the POS system. Its features will include,

* Recording each transaction with details like item, quantity, price, and timestamp.
* Supporting multiple payment (Wikipedia, 2024) methods, including cash, cards, and mobile payments.
* Ensuring receipts are generated and printed quickly to reduce checkout delays.

5. Reporting and Analytics

Management needs actionable data to make decisions. The reporting module will:

* Generate daily, weekly, and monthly reports summarizing sales, revenue, and inventory usage.
* Highlight trends such as best-selling items or peak shopping times.
* Present data visually through charts and graphs for better understanding.

### Identified business needs and challenges.

**Business Needs**

* **Automation**: Automating tasks like inventory updates, sales recording, and discount application will reduce the manual workload and errors.
* **Real-Time Data Access**: Managers and staff require instant updates on inventory and sales to make timely decisions.
* **Customer Engagement**: Personalized interactions through loyalty programs and promotions will improve customer satisfaction and retention.
* **Data-Driven Decisions**: Reports on sales trends and stock levels are essential for effective planning and forecasting.

**Challenges**

* **Scalability**: The system must be flexible to accommodate future expansions, such as new branches or online sales.
* **Security**: Protecting sensitive customer and business data from unauthorized access is critical.
* **User Adoption**: Designing a user-friendly interface will be necessary to ensure that staff can use the system efficiently with minimal training.
* **Data Accuracy**: Maintaining accurate and consistent data across all modules of the system is a major technical challenge.

### objectives of the database AND application DEVELOPMENT.

**Primary Objectives**

1. **Streamline Operations**: Automate repetitive tasks to save time and reduce human errors.
2. **Ensure Data Integrity**: Use database constraints and validation rules to maintain data consistency.
3. **Empower Management**: Provide detailed reports and analytics to assist in decision-making.
4. **Enhance User Experience**: Create a simple and intuitive interface for all system users.

**Secondary Objectives**

* Allow easy updates to discount and stock information.
* Provide secure access to sensitive data with role-based permissions.
* Offer scalability to support future business growth.

### assumptions and constraints.

**Additional Assumptions**

* The supermarket primarily operates offline, and this system will be deployed locally with optional cloud backup.
* Staff using the system have basic knowledge of computers and will receive training for the application.
* Discount rules and stock categories will be predefined by the management during the setup phase.
* The supermarket’s internet connectivity is reliable for any optional online inetegrations in the future.

**Constraints on Data Handling and Software Features**

* **Data Accuracy**: Data validation (CFI, 2024) will be enforced through primary keys, foreign keys, and constraints. For instance, a sales transaction cannot occur without a corresponding product in stock.
* **Performance**: The system must handle peak transaction loads efficiently, especially during busy hours.
* **Security**: Sensitive information, such as customer details and payment records, must be encrypted and accessible only to authorized users.
* **Compliance**: The system must adhere to relevant data protection laws and industry standards.

# A diagram of a company Description automatically generatedER/EER DIAGRAM.

## Assumptions for the ER/EER Diagram (Microsoft, 2024)

1. **Customer**:

* Each customer has a unique CustomerID.
* Customers can register in the system, and their details such as name, email, and phone numbers are stored.
* A customer can make multiple transactions (1:M relationship with Sales).

1. **Product**:

* Each product has a unique ProductID.
* Products belong to specific categories.
* ReorderLevel indicates the minimum quantity of a product before it needs restocking.
* **Product - Stock:** Each product can have different stock levels across various warehouses.(1:M)

1. **Stock**:

* Each Stock is uniquely identified by a Stock\_ID.
* Stock shows us the last Restocked date.
* Stocks are stored in a WareHouse.

1. **Discount**:

* Discounts are applied to Sales, creating an M:N relationship between products and Sales.
* Each discount is uniquely identified by a DiscountID and includes details like Discount Details, start date, and end date.
* Discounts can apply to multiple Sales.

1. **Sales**:

* Each sales transaction is uniquely identified by a TransactionID.
* Sales include details of the customer making the purchase, the Discount method, and the Employees.
* Discounts applied during sales are linked to the transaction.

1. **Employee** :

* Employees are uniquely identified by EmployeeID.
* Each employee have Employee Id, contact information, First and Last Name .
* Employees perform actions logged in the AuditLog.

1. **Audit Logging**:

* Actions performed by employees (e.g., adding, modifying, or deleting data) are recorded in the AuditLog with details of the action and a time.

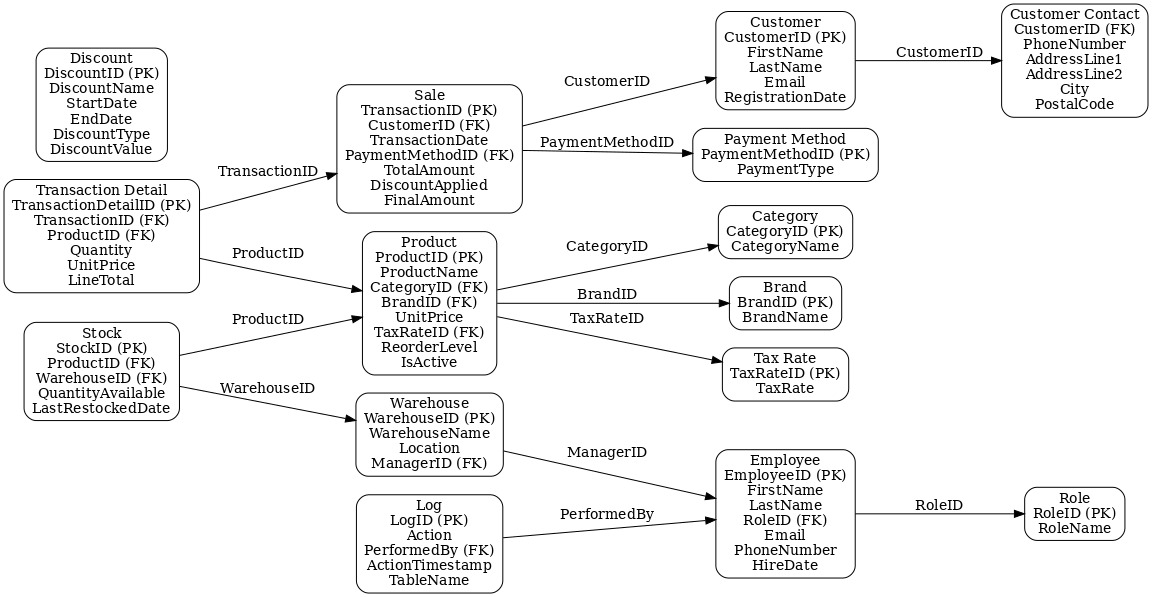
1. **Report Generation**:

* Reports are uniquely identified by a ReportID and include details such as the report type, generation date, and generated by (an employee).
* Reports are generated daily or monthly to help the owner make informed decisions.
* Sales and other operational data contribute to report generation (1:M relationship between Reports and Sales).

1. **Relationships and Cardinality**:

* A customer can have multiple transactions (1:M between Customer and Sales).
* A Sales can belong to multiple discounts, and discounts can apply to multiple Sales (N:N between Sales and Discount).
* Employees can generate multiple reports and perform multiple actions logged in the audit (1:N relationships)

# relational mapping



# normalization.

### 1NF (First Normal Form)

In **1NF** (Wikipedia, 2024), all data must be atomic (no multi-valued attributes), and there must be a unique identifier for each row.

**Customer Management**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **LastName** | **Email** | **PhoneNumber** | **AddressLine1** | **AddressLine2** | **City** | **PostalCode** | **RegistrationDate** |
| C123 | John | Doe | [john@email.com](mailto:john@email.com) | 1234567890 | Line1 | Line2 | New York | 10001 | 2024-01-01 |
| C124 | Jane | Smith | [jane@email.com](mailto:jane@email.com) | 0987654321 | Line3 | Line4 | Chicago | 60606 | 2024-02-01 |

**Product Management**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **Category** | **Brand** | **UnitPrice** | **TaxRate** | **ReorderLevel** | **IsActive** |
| P101 | TV | Electronics | Sony | 1000.00 | 10% | 5 | Yes |
| P102 | Sofa | Furniture | Ikea | 500.00 | 5% | 2 | Yes |

**Stock Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StockID** | **ProductID** | **Warehouse** | **QuantityAvailable** | **LastRestockedDate** |
| S001 | P101 | Warehouse1 | 10 | 2024-01-10 |
| S002 | P102 | Warehouse2 | 5 | 2024-01-15 |

**Discount Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DiscountID** | **DiscountName** | **StartDate** | **EndDate** | **DiscountDetails** |
| D001 | Summer Sale | 2024-06-01 | 2024-06-30 | 15%, Percentage |

**Sales Management**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **TransactionID** | **CustomerID** | **TransactionDate** | **PaymentMethod** | **TotalAmount** | **DiscountApplied** | **FinalAmount** | **ProductDetails** |
| T001 | C123 | 2024-01-05 | Credit Card | 2000.00 | 15% | 1700.00 | P101: 2 units, $1000/unit |

**Employee Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **FirstName** | **LastName** | **Role** | **ContactInfo** | **HireDate** |
| E001 | Alice | Brown | Manager | [alice@company.com](mailto:alice@company.com), 1234567 | 2020-01-01 |

### 2NF (Second Normal Form).

To achieve **2NF**, all partial dependencies (fields depending only on part of a composite key) are removed. This step involves creating separate tables for attributes that do not directly depend on the primary key.

**Customer Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **LastName** | **Email** | **RegistrationDate** |
| C123 | John | Doe | [john@email.com](mailto:john@email.com) | 2024-01-01 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerID** | **PhoneNumber** | **AddressLine1** | **AddressLine2** | **City** | **PostalCode** |
| C123 | 1234567890 | Line1 | Line2 | New York | 10001 |

**Product Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **UnitPrice** | **ReorderLevel** | **IsActive** |
| P101 | TV | 1000.00 | 5 | Yes |

|  |  |
| --- | --- |
| **ProductID** | **Category** |
| P101 | Electronics |

|  |  |
| --- | --- |
| **ProductID** | **Brand** |
| P101 | Sony |

|  |  |
| --- | --- |
| **ProductID** | **TaxRate** |
| P101 | 10% |

**Stock Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StockID** | **ProductID** | **Warehouse** | **QuantityAvailable** | **LastRestockedDate** |
| S001 | P101 | Warehouse1 | 10 | 2024-01-10 |

**Discount Management**

|  |  |  |  |
| --- | --- | --- | --- |
| **DiscountID** | **DiscountName** | **StartDate** | **EndDate** |
| D001 | Summer Sale | 2024-06-01 | 2024-06-30 |

|  |  |  |
| --- | --- | --- |
| **DiscountID** | **DiscountType** | **DiscountValue** |
| D001 | Percentage | 15% |

**Sales Management**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TransactionID** | **CustomerID** | **TransactionDate** | **PaymentMethod** | **TotalAmount** | **DiscountApplied** | **FinalAmount** |
| T001 | C123 | 2024-01-05 | Credit Card | 2000.00 | 15% | 1700.00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TransactionID** | **ProductID** | **Quantity** | **UnitPrice** | **LineTotal** |
| T001 | P101 | 2 | 1000.00 | 2000.00 |

**Employee Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EmployeeID** | **FirstName** | **LastName** | **RoleID** | **HireDate** |
| E001 | Alice | Brown | R001 | 2020-01-01 |

|  |  |
| --- | --- |
| **RoleID** | **RoleName** |
| R001 | Manager |

|  |  |
| --- | --- |
| **EmployeeID** | **ContactInfo** |
| E001 | [alice@company.com](mailto:alice@company.com), 1234567 |

### 3NF (third Normal Form).

To achieve **3NF**, all transitive dependencies (attributes depending on non-primary key attributes) are removed. The following tables demonstrate the fully normalized structure:

**Customer Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **LastName** | **Email** | **RegistrationDate** |
| C123 | John | Doe | [john@email.com](mailto:john@email.com) | 2024-01-01 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerID** | **PhoneNumber** | **AddressLine1** | **AddressLine2** | **City** | **PostalCode** |
| C123 | 1234567890 | Line1 | Line2 | New York | 10001 |

**Product Management**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **CategoryID** | **BrandID** | **UnitPrice** | **TaxRateID** | **ReorderLevel** | **IsActive** |
| P101 | TV | CAT01 | BR01 | 1000.00 | TAX01 | 5 | Yes |

|  |  |
| --- | --- |
| **CategoryID** | **CategoryName** |
| CAT01 | Electronics |

|  |  |
| --- | --- |
| **BrandID** | **BrandName** |
| BR01 | Sony |

|  |  |
| --- | --- |
| **TaxRateID** | **TaxRate** |
| TAX01 | 10% |

**Stock Management**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StockID** | **ProductID** | **WarehouseID** | **QuantityAvailable** | **LastRestockedDate** |
| S001 | P101 | WH01 | 10 | 2024-01-10 |

|  |  |  |  |
| --- | --- | --- | --- |
| **WarehouseID** | **WarehouseName** | **Location** | **ManagerID** |
| WH01 | Warehouse1 | New York City | E001 |

**Discount Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DiscountID** | **DiscountName** | **StartDate** | **EndDate** | **DiscountType** | **DiscountValue** |
| D001 | Summer Sale | 2024-06-01 | 2024-06-30 | Percentage | 15% |

**Sales Management**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TransactionID** | **CustomerID** | **TransactionDate** | **PaymentMethodID** | **TotalAmount** | **DiscountApplied** | **FinalAmount** |
| T001 | C123 | 2024-01-05 | PM01 | 2000.00 | 15% | 1700.00 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TransactionDetailID** | **TransactionID** | **ProductID** | **Quantity** | **UnitPrice** | **LineTotal** |
| TD001 | T001 | P101 | 2 | 1000.00 | 2000.00 |

|  |  |
| --- | --- |
| **PaymentMethodID** | **PaymentType** |
| PM01 | Credit Card |

**Employee Management**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **FirstName** | **LastName** | **RoleID** | **Email** | **PhoneNumber** | **HireDate** |
| E001 | Alice | Brown | R001 | [alice@company.com](mailto:alice@company.com) | 1234567890 | 2020-01-01 |

|  |  |
| --- | --- |
| **RoleID** | **RoleName** |
| R001 | Manager |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LogID** | **Action** | **PerformedBy** | **ActionTimestamp** | **TableName** |
| LOG001 | Update Stock | E001 | 2024-01-10 10:30:00 | Stock |

**Report Generation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ReportID** | **ReportName** | **GeneratedBy** | **GeneratedDate** | **ReportType** | **ReportData** |
| REP001 | Daily Sales Report | E001 | 2024-01-11 | Sales | JSON Data |

### Process of normalization

**Normalization Process: 1NF, 2NF, and 3NF**

Normalization (Geeks for Geeks, 2024) is a systematic approach to organizing data in a database to reduce redundancy and improve data integrity. This document explains normalization through **1NF**, **2NF**, and **3NF**, using examples to demonstrate how redundant data is removed at each stage.

**1NF (First Normal Form)**

**Definition**

1NF ensures that:

* All columns in a table contain atomic (indivisible) values.
* Each record is unique, and there are no duplicate rows.
* There are no repeating groups or arrays within the table.

**Example: Data in Unnormalized Form (0NF)**

Consider a CustomerOrders table that stores customer orders.

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CustomerName** | **Orders** |
| C001 | John Doe | TV: 2 units, $1000/unit; Sofa: 1 unit, $500 |
| C002 | Jane Smith | TV: 1 unit, $1000/unit |

**Transition to 1NF**

To achieve 1NF:

* Split multi-valued attributes (e.g., Orders) into atomic values.
* Create a separate row for each unique order.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ProductName** | **Quantity** | **UnitPrice** |
| C001 | John Doe | TV | 2 | 1000 |
| C001 | John Doe | Sofa | 1 | 500 |
| C002 | Jane Smith | TV | 1 | 1000 |

**2NF (Second Normal Form)**

**Definition**

2NF eliminates partial dependencies, where non-primary key attributes depend only on part of a composite primary key. To achieve 2NF:

* Move attributes that are dependent on only part of the primary key to a separate table.
* Ensure all attributes depend on the entire primary key.

**Example: Data in 1NF**

Continuing with the CustomerOrders table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ProductName** | **Quantity** | **UnitPrice** |
| C001 | John Doe | TV | 2 | 1000 |
| C001 | John Doe | Sofa | 1 | 500 |
| C002 | Jane Smith | TV | 1 | 1000 |

**Identifying Partial Dependencies**

* CustomerName depends only on CustomerID.
* UnitPrice depends only on ProductName.

**Transition to 2NF**

Split the table into two separate tables:

**Customer Table**

|  |  |
| --- | --- |
| **CustomerID** | **CustomerName** |
| C001 | John Doe |
| C002 | Jane Smith |

**Product Table**

|  |  |
| --- | --- |
| **ProductName** | **UnitPrice** |
| TV | 1000 |
| Sofa | 500 |

**CustomerOrders Table**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **ProductName** | **Quantity** |
| C001 | TV | 2 |
| C001 | Sofa | 1 |
| C002 | TV | 1 |

By eliminating partial dependencies, redundancy is reduced. For example, the price of a product is stored only once in the Product table, rather than being repeated for every order.

**3NF (Third Normal Form)**

**Definition**

3NF eliminates transitive dependencies, where a non-primary key attribute depends on another non-primary key attribute. To achieve 3NF:

* Move attributes that do not directly depend on the primary key to a separate table.

**Example: Data in 2NF**

Consider the Customer table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **City** | **PostalCode** |
| C001 | John Doe | New York | 10001 |
| C002 | Jane Smith | Chicago | 60606 |

**Identifying Transitive Dependencies**

* City depends on PostalCode.
* CustomerID is the primary key.

**Transition to 3NF**

Split the table into two separate tables:

**Customer Table**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CustomerName** | **PostalCode** |
| C001 | John Doe | 10001 |
| C002 | Jane Smith | 60606 |

**Location Table**

|  |  |
| --- | --- |
| **PostalCode** | **City** |
| 10001 | New York |
| 60606 | Chicago |

By removing transitive dependencies, redundancy is further reduced. For instance, the city corresponding to a postal code is stored only once in the Location table.

**Summary of Normalization Steps**

|  |  |  |
| --- | --- | --- |
| **Normalization Form** | **Key Action** | **Result** |
| 1NF | Remove multi-valued attributes | Atomic values and unique rows |
| 2NF | Eliminate partial dependencies | Attributes depend on the entire primary key |
| 3NF | Eliminate transitive dependencies | Attributes depend only on the primary key |

By applying normalization, we ensure the database is efficient, consistent, and free of unnecessary redundancy. This improves data integrity and minimizes the risk of anomalies during data manipulation.

# data dictionary.

**Customer Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Customer | CustomerID | INT | - | Primary Key, Not Null, Auto Increment | Primary Key |
|  | FirstName | VARCHAR | 100 | Not Null | Customer's first name |
|  | LastName | VARCHAR | 100 | Not Null | Customer's last name |
|  | Email | VARCHAR | 255 | Not Null, Unique | Customer's email address |
|  | PhoneNumber | VARCHAR | 11 | Not Null | Customer's phone number |
|  | AddressLine1 | VARCHAR | 255 | Not Null | Primary address line |
|  | AddressLine2 | VARCHAR | 255 | Null | Secondary address line |
|  | City | VARCHAR | 100 | Not Null | City of residence |
|  | PostalCode | VARCHAR | 20 | Not Null | Postal code |
|  | RegistrationDate | DATE | - | Not Null | Date of registration |

**Product and Stock Management**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table Name** | | **Field Name** | | **Data Type** | | **Length** | **Constraints** | **Description** |
| Product | | ProductID | | INT | |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | | ProductName | | VARCHAR | | 255 | Not Null | Name of the product |
|  | | CategoryID | | INT | |  | Foreign Key, Not Null | Foreign Key to Category |
|  | | BrandID | | INT | |  | Foreign Key, Not Null | Foreign Key to Brand |
|  | | UnitPrice | | DECIMAL | | 10,2 | Not Null | Price per unit |
|  | | TaxRateID | | INT | |  | Foreign Key, Not Null | Foreign Key to TaxRate |
|  | | ReorderLevel | | INT | |  | Not Null | Stock level to trigger reorder |
|  | | IsActive | | BOOLEAN | |  | Not Null | Indicates if the product is active |
| **Table Name** | **Field Name** | | **Data Type** | | **Length** | | **Constraints** | **Description** |
| Category | CategoryID | | INT | |  | | Primary Key, Not Null, Auto Increment | Primary Key |
|  | CategoryName | | |  |  | | --- | --- | | VARCHAR | | |  | | | 255 | | Not Null | Name of the category |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Brand | BrandID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | BrandName | |  |  | | --- | --- | | VARCHAR | | |  | | 255 | Not Null | Name of the brand |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| TaxRate | TaxRateID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | TaxRate | |  |  | | --- | --- | | DECIMAL | | |  | | 5,2 | Not Null | Tax percentage applied |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Stock | StockID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | ProductID | |  |  | | --- | --- | | INT | | |  | | 5,2 | Foreign Key, Not Null | Foreign Key to Product |
|  | WarehouseID | INT |  | Foreign Key, Not Null | Foreign Key to Warehouse |
|  | QuantityAvailable | INT |  | Not Null | Quantity available in stock |
|  | LastRestockedDate | DATE |  | Not Null | Date when stock was last restocked |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Warehouse | WarehouseID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | WarehouseName | |  |  | | --- | --- | | VARCHAR | | |  | | 255 | Not Null | Name of the warehouse |
|  | Location | VARCHAR | 255 | Not Null | Location of the warehouse |
|  | ManagerID | INT |  | Foreign Key, Not Null | Foreign Key to Employee |

**Discount Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Discount | DiscountID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | DiscountName | |  |  | | --- | --- | | VARCHAR | | |  | | 5,2 | Not Null | Name of the discount |
|  | StartDate | DATE |  | Not Null | Start date of the discount |
|  | EndDate | DATE |  | Not Null | End date of the discount |
|  | DiscountPercentage | |  | | --- | |  |  |  | | --- | | DECIMAL | | 5,2 | Not Null | Percentage of discount |
|  | DiscountType | |  | | --- | | VARCHAR |  |  | | --- | |  | | 50 | Not Null | Type of discount (e.g., Percentage, Flat) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| ProductDiscount | ProductDiscountID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | ProductID | |  |  | | --- | --- | | INT | | |  | |  | Foreign Key, Not Null | Foreign Key to Product |
|  | DiscountID | INT |  | Foreign Key, Not Null | Foreign Key to Discount |

**Sales Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| SalesTransaction | TransactionID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | CustomerID | |  |  | | --- | --- | | INT | | |  | |  | Foreign Key (Nullable for Guest Customers) | Foreign Key to Customer, Nullable for Guest Customers |
|  | TransactionDate | DATE |  | Not Null | Date of transaction |
|  | PaymentMethodID | INT |  | Foreign Key, Not Null | Foreign Key to PaymentMethod |
|  | TotalAmount | DECIMAL | 10,2 | Not Null | Total amount before discounts |
|  | DiscountApplied | DECIMAL | 5,2 | Not Null | Discount applied to the transaction |
|  | FinalAmount | DECIMAL | 10,2 | Not Null | Final amount after discounts |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| SalesTransactionDetail | TransactionDetailID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | TransactionID | |  |  | | --- | --- | | INT | | |  | |  | Foreign Key, Not Null | Foreign Key to SalesTransaction |
|  | ProductID | INT |  | Foreign Key, Not Null | Foreign Key to Product |
|  | Quantity | INT |  | |  | | --- | | Not Null | | Quantity of product sold |
|  | UnitPrice | DECIMAL | 10,2 | Not Null | Price per unit |
|  | LineTotal | DECIMAL | 10,2 | Not Null | Total price for the line item |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| PaymentMethod | PaymentMethodID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | PaymentType | |  |  | | --- | --- | | VARCHAR | | |  | | 50 | Not Null | Type of payment (e.g., Cash, Credit Card, Mobile Payment) |

**Employee Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Employee | EmployeeID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | FirstName | |  |  | | --- | --- | | VARCHAR | | |  | | 100 | Not Null | First name of the employee |
|  | LastName | VARCHAR | 100 | Not Null | Last name of the employee |
|  | RoleID | INT |  | Foreign Key, Not Null | Foreign Key to Role |
|  | Email | VARCHAR | 255 | Not Null | Employee's email address |
|  | PhoneNumber | VARCHAR | 11 | Not Null | Employee's phone number |
|  | HireDate | DATE |  | Not Null | Date of hire |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Role | RoleID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | RoleName | |  |  | | --- | --- | | VARCHAR | | |  | | 100 | Not Null | Name of the role (e.g., Admin, Cashier, Manager) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| AuditLog | LogID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | Action | |  |  | | --- | --- | | TEXT | | |  | |  | Not Null | Description of the action |
|  | PerformedBy | INT |  | Foreign Key to Employee, Not Null | Foreign Key to Employee |
|  | ActionTimestamp | TIMESTAMP |  | Not Null | Timestamp of the action |
|  | TableName | VARCHAR | 100 | Not Null | Name of the table affected |

**Report Generation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table Name** | **Field Name** | **Data Type** | **Length** | **Constraints** | **Description** |
| Report | ReportID | INT |  | Primary Key, Not Null, Auto Increment | Primary Key |
|  | ReportName | |  |  | | --- | --- | | VARCHAR | | |  | | 255 | Not Null | Name of the report |
|  | GeneratedBy | INT |  | Foreign Key to Employee | Foreign Key to Employee |
|  | GeneratedDate | DATE |  | Not Null | Date the report was generated |
|  | ReportType | VARCHAR | 50 | Not Null | Type of report (e.g., Daily, Monthly, Sales Summary) |
|  | ReportData | BLOB/JSON |  | Not Null | Data in the report (Blob or JSON) |

A screenshot of a computer screen

Description automatically generatedRelationship

# DATABASE DESIGN

## custommer management

### A screenshot of a computer Description automatically generated6.1.1. “customer” table

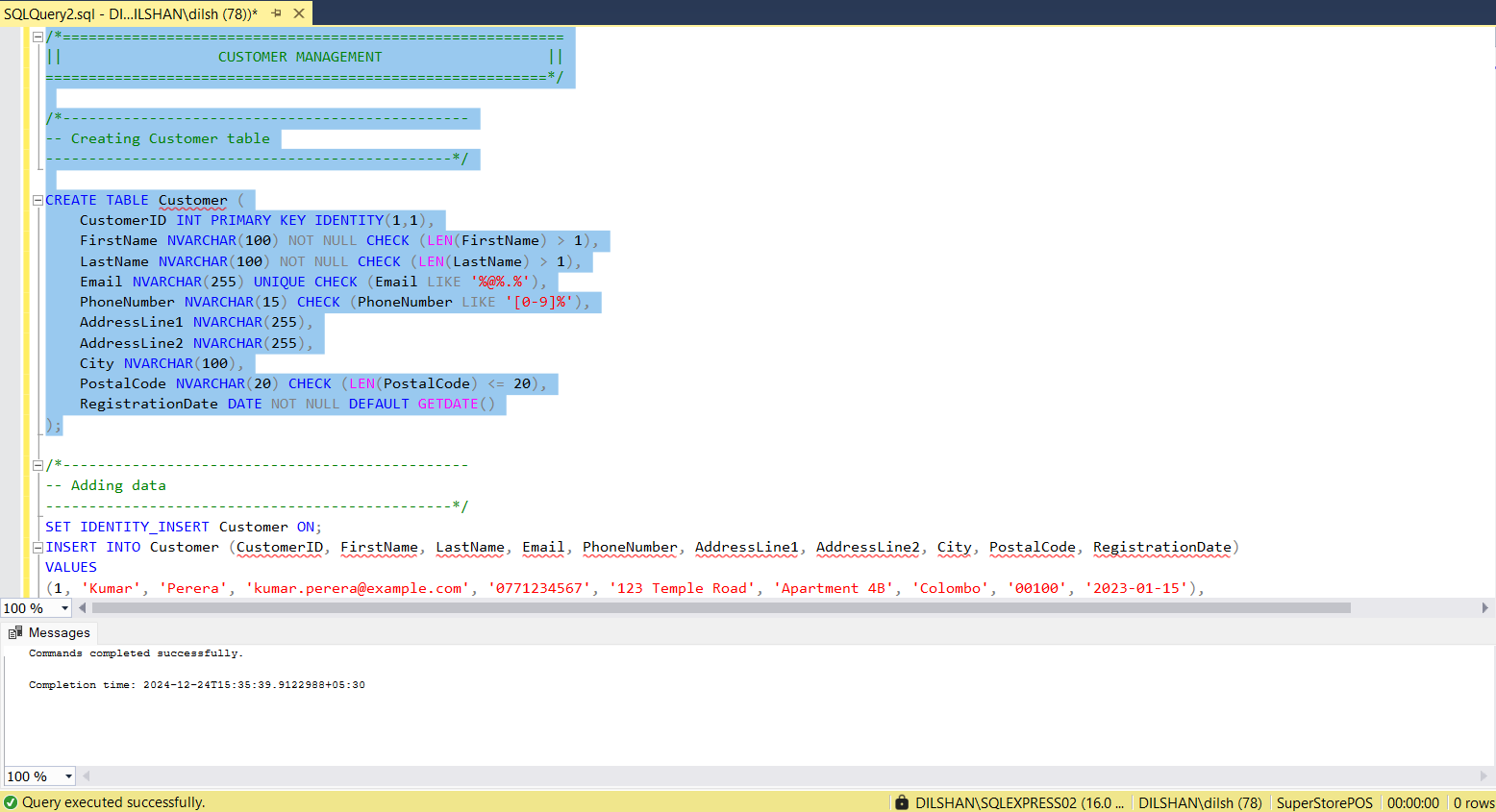
### 6.1.2. purpose

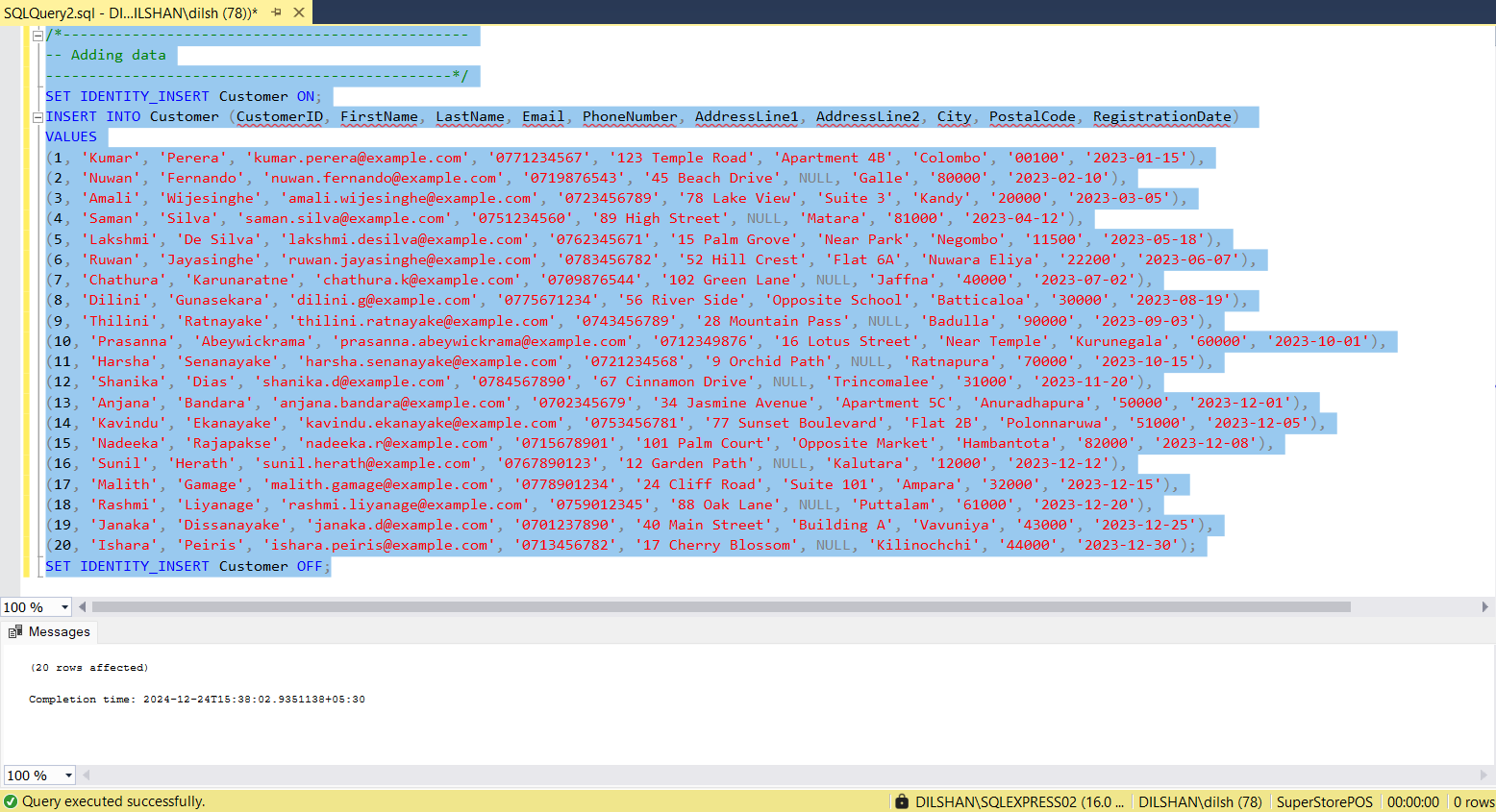
Manages customer information like personal details, contact information, and registration date. Essential for identifying and contacting customers (IBM, 2024).

### 6.1.3. data validation

* **Primary Key Constraint:** (CustomerID) Ensures unique identification of each customer.
* **NOT NULL Constraints:** Prevent missing essential details like (FirstName) and (LastName).
* **CHECK Constraints:**
* Validates (FirstName) and (LastName) length.
* Ensures (Email) follows a valid format and is unique.
* Validates (PhoneNumber) starts with digits.
* Limits (PostalCode) length.

### 6.1.3. code & output results.





## employee management

### 6.2.1.1. “employee” table

A screenshot of a computer

Description automatically generated

### 6.2.1.2. porpose

Maintains employee information (Resorce for Employee, 2024), linking employees to roles. Tracks hire dates and contact details for the organization.

### 6.2.1.3. data validation

* **Primary Key Constraint:** (EmployeeID) Uniquely identifies employees.
* **Foreign Key Constraint:** (RoleID) Links employees to roles.
* **NOT NULL Constraints:** Prevent missing essential details like (FirstName), (LastName), (Email) and (HireDate)
* **CHECK Constraints:**
* Validates (FirstName), (LastName), and (Email) format.
* Ensures (HireDate) is not in the future.

### 6.2.1.4. code & output results

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### 6.2.2.1. “role” table

A screenshot of a computer

Description automatically generated

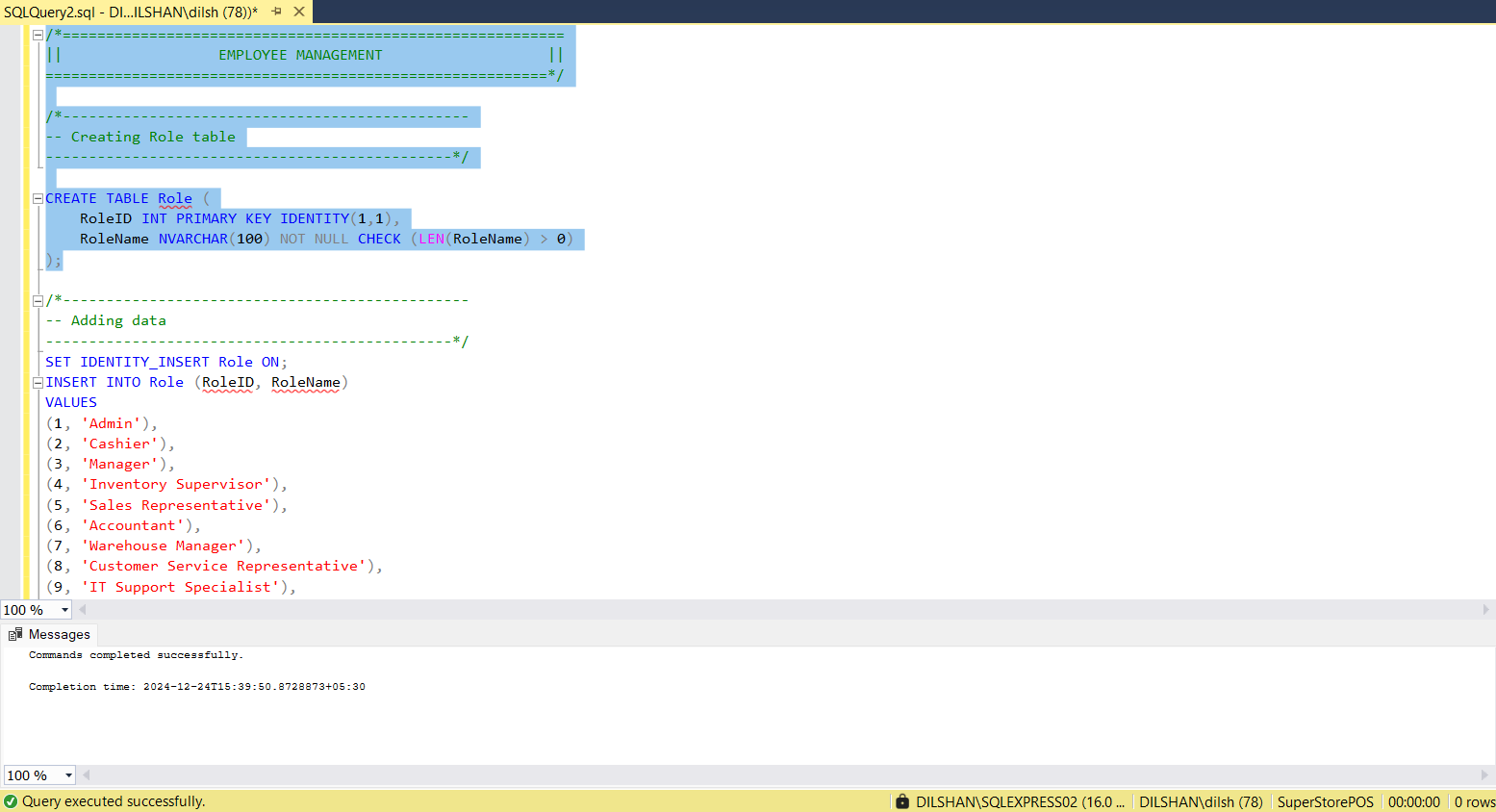
### 6.2.2.2. porpose

Defines roles (e.g. Admin, Manager) for employees, helping assign and manage responsibilities.

### 6.2.2.3. data validation

* **Primary Key Constraint:** (RoleID) Ensures unique role identification.
* **NOT NULL Constraints:** Ensure (RoleName) is not empty.

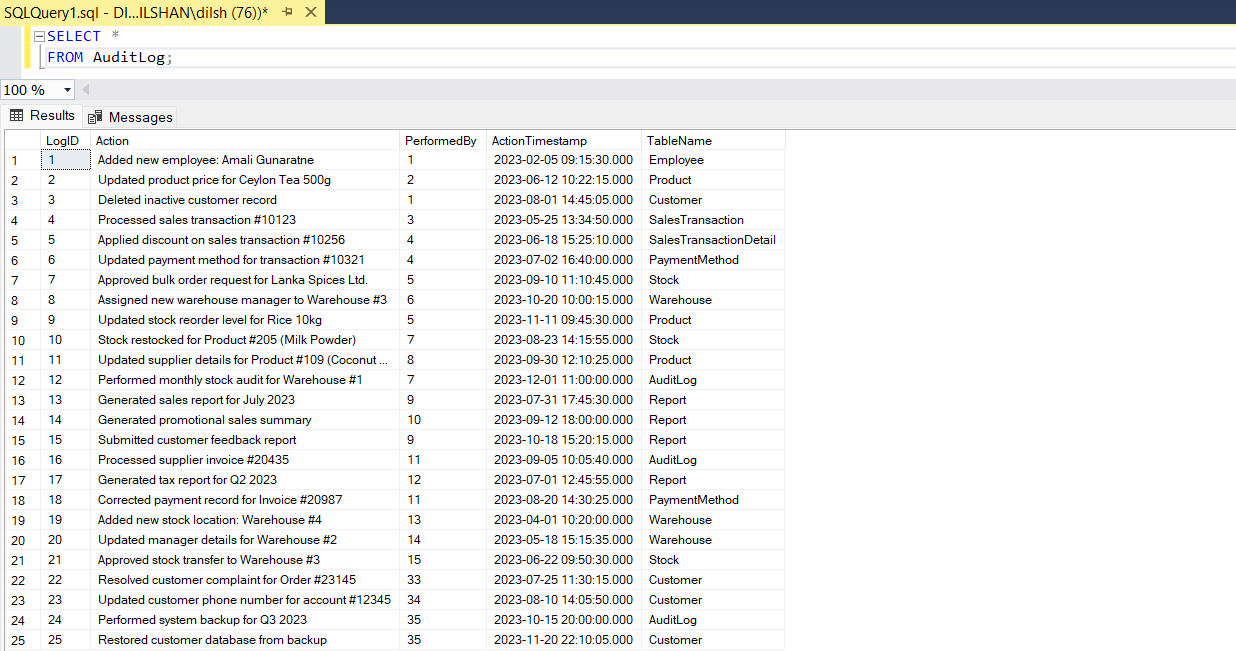
### 6.2.2.3. code & output results



A screenshot of a computer

Description automatically generated

### 6.2.3.1. “auditlog” table



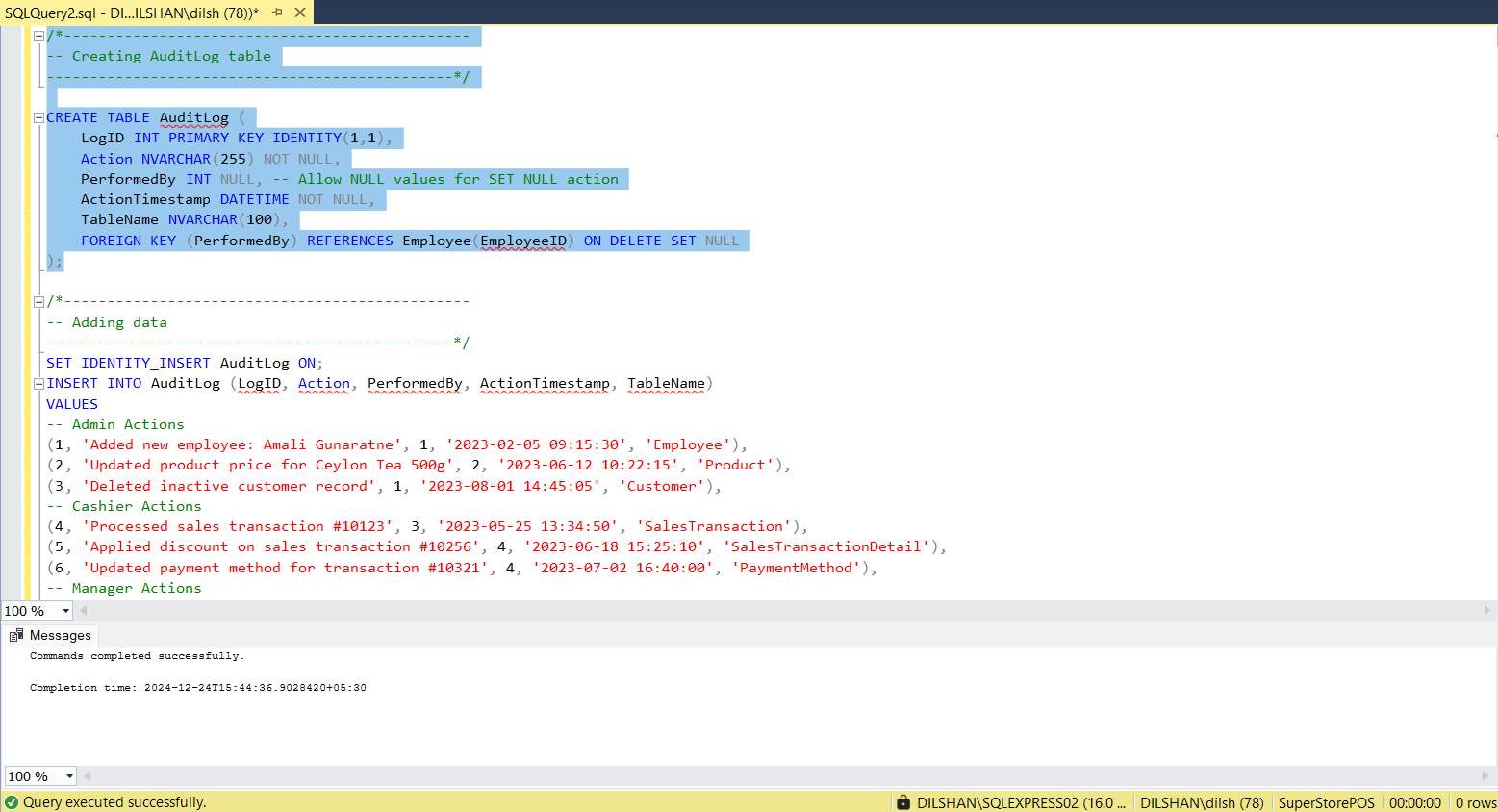
### 6.2.3.2. purpose

Logs administrative actions performed by employees, ensuring traceability and accountability.

### 6.2.3.3. data validation

* **Primary Key Constraint:** (LogID) Unique identifier for log entries.
* **Foreign Key Constraint:** (PerformedBy) Links actions to employees, allowing NULL for unspecified actions.
* **NOT NULL Constraints:** Ensure (Action) and (ActionTimestamp) are not empty.

### 6.2.3.4. code & output results



## “report” generation

### 6.3.1. “report” table

### 6.3.2. purpose

Stores generated reports, linking them to the responsible employee. Useful for record-keeping and analysis.

### 6.3.3. data validation

* **Primary Key Constraint:** (ReportID) Uniquely identifies reports.
* **Foreign Key Constraint:** (GeneratedBy) Links reports to generating employees.
* **NOT NULL Constraints:** Prevent missing essential details (GeneratedDate) and (ReportType).
* **CHECK Constraints:**
* Ensures (ReportName) has at least 1 character

### 6.3.4. code & output results

A screenshot of a computer

Description automatically generated

## product and stock management

### 6.4.1.1. “category” table

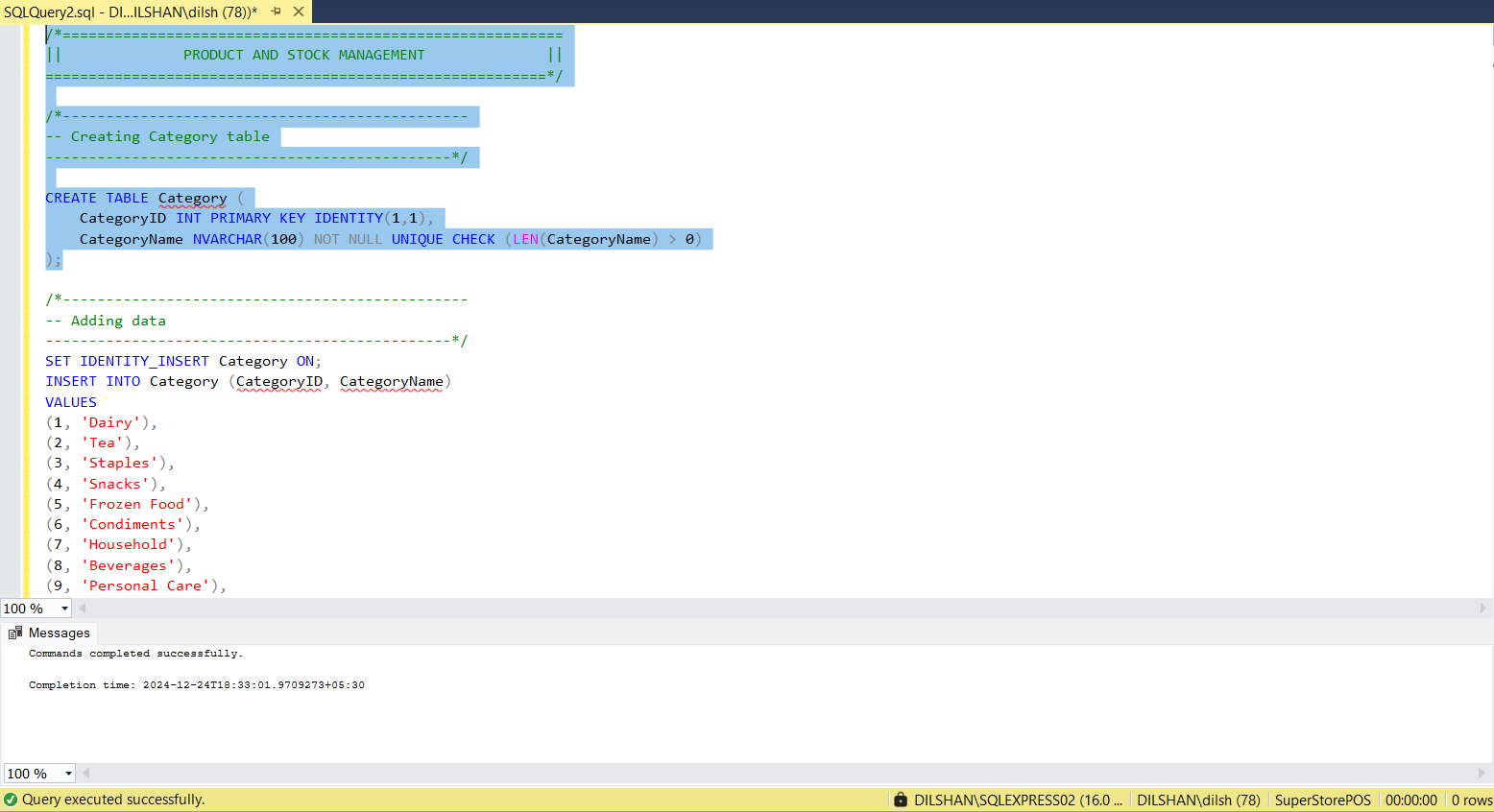
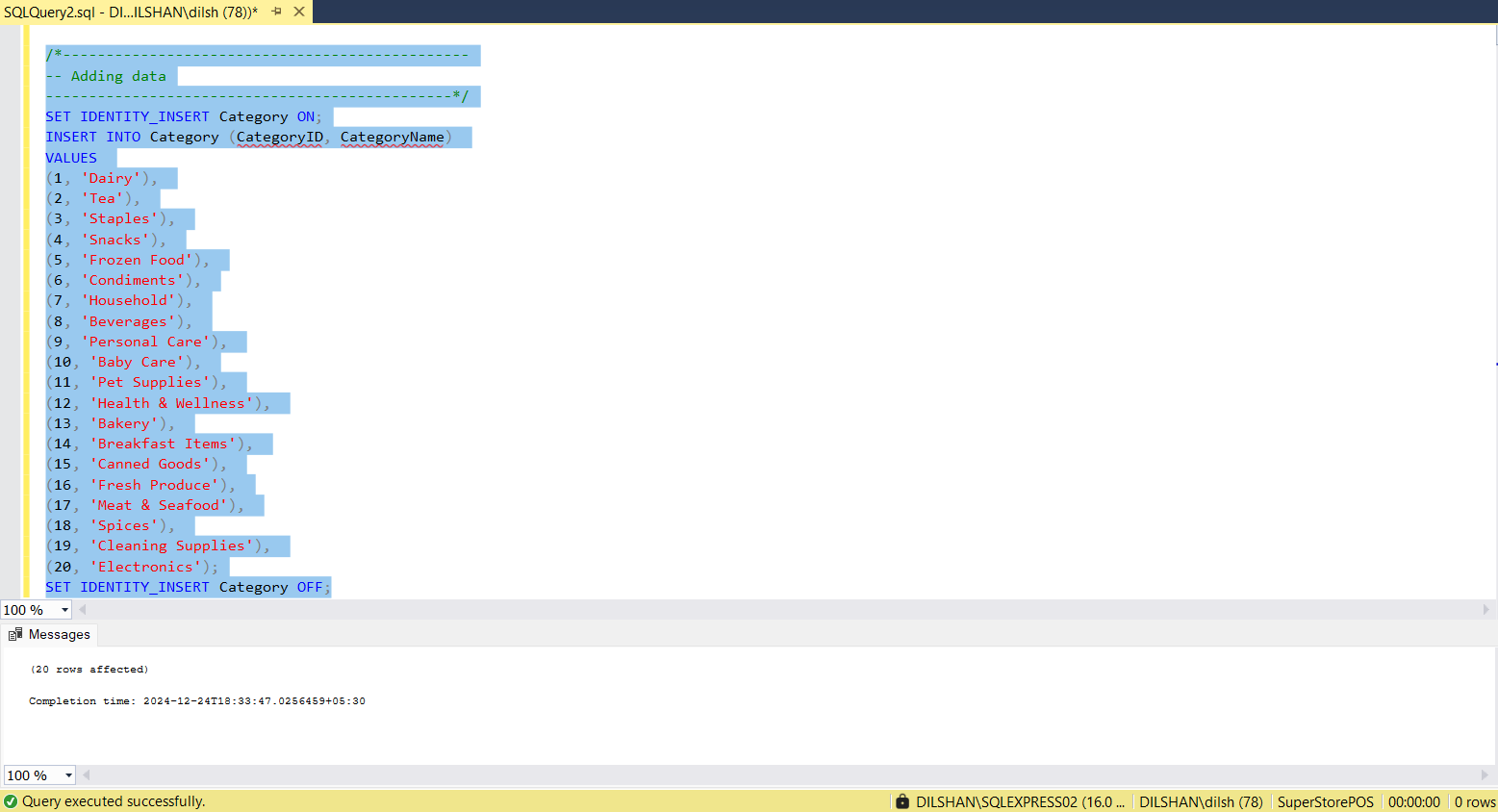
### 6.4.1.2. purpose

Categorizes products for better organization and retrieval.

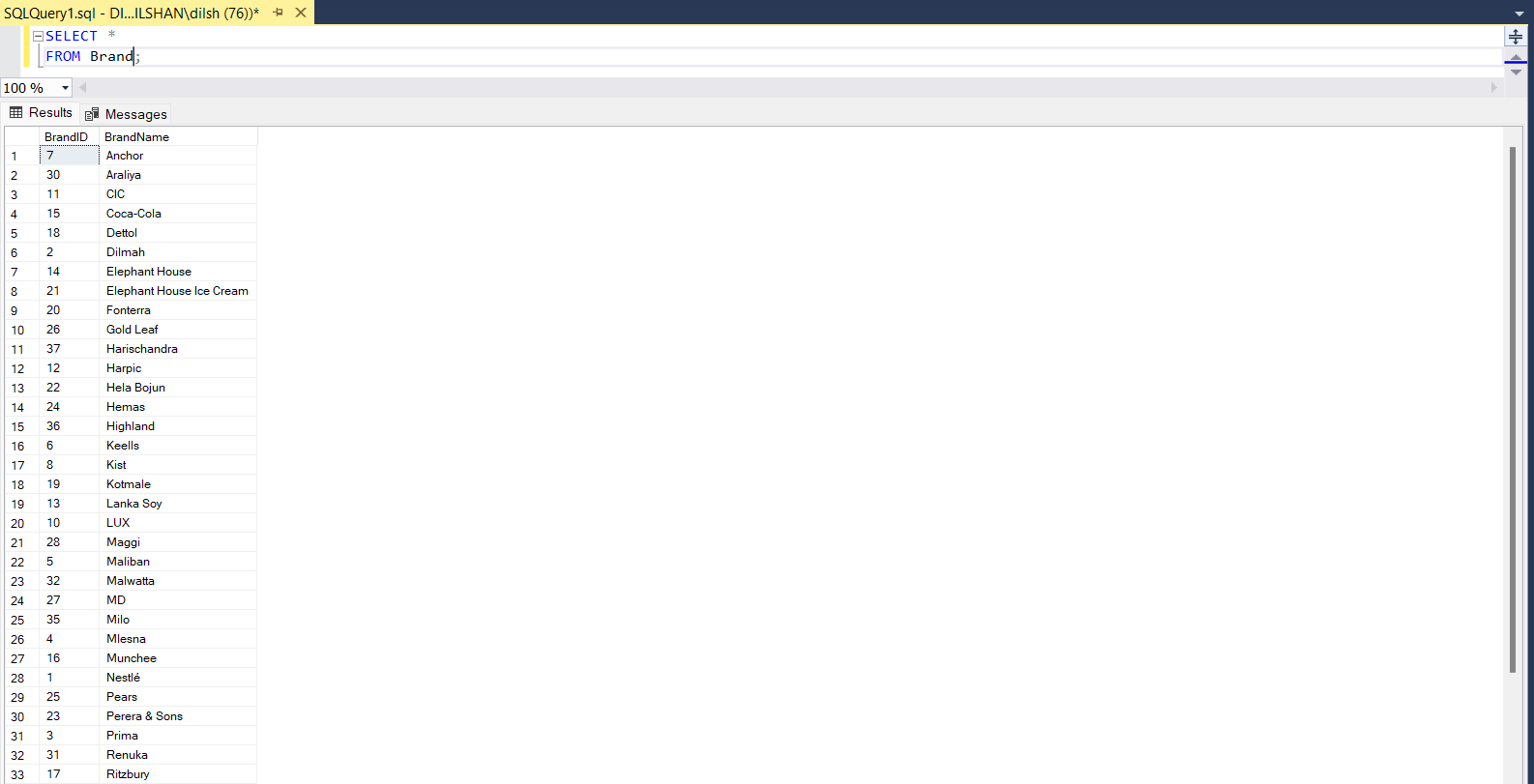
### 6.4.1.3. data validation

* **Primary Key Constraint:** (CategoryID) Ensures unique category identification.
* **NOT NULL Constraints:** Ensure (CategoryName) is not empty.
* **CHECK Constraints:**
* Ensures (CategoryName) is unique and valid.

### 6.4.1.4. code & output results



### 6.4.2.1. “brand” table



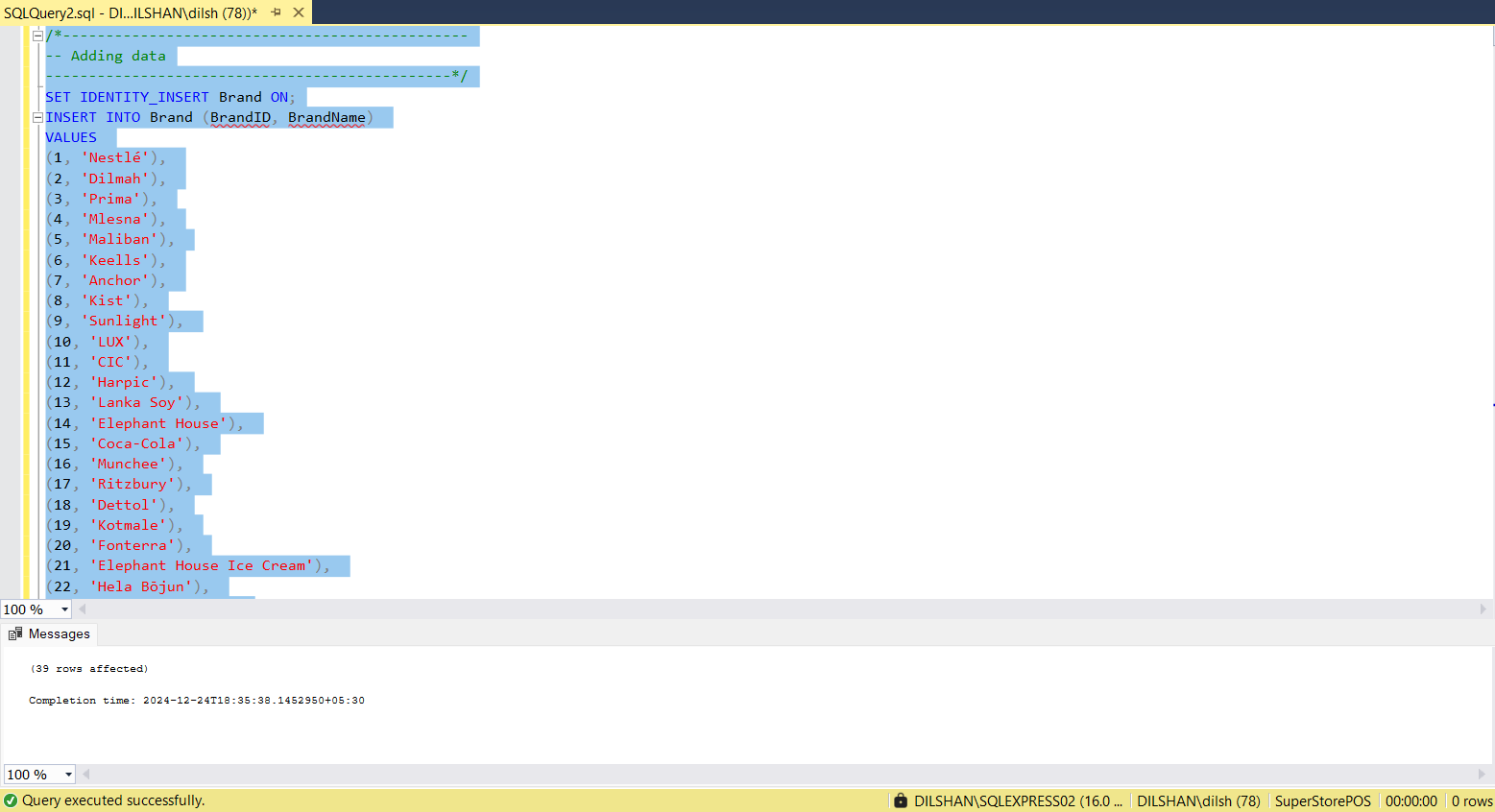
### 6.4.2.2. purpose

Manages information about product brands, and helps in brand-based product searches.

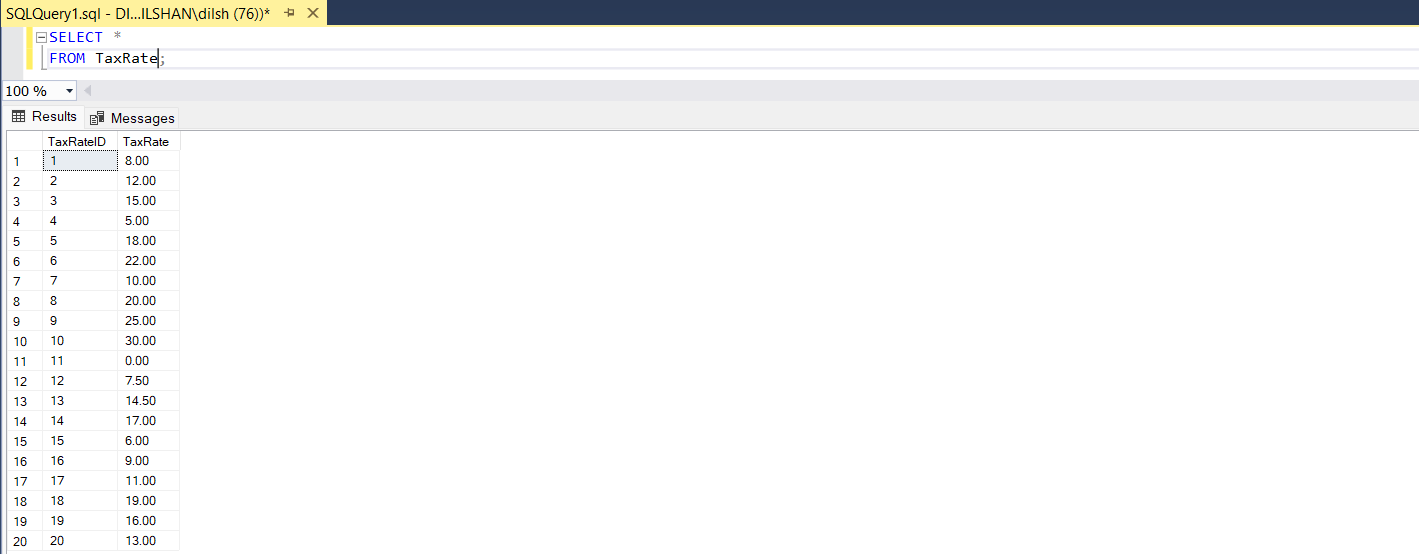
### 6.4.2.3. data validation

* **Primary Key Constraint:** (BrandID) Unique identifier for brands.
* **NOT NULL Constraints:** Ensure (BrandName) is not empty.
* **CHECK Constraints:**
* Ensures (BrandName) is unique and valid.

### 6.4.2.4. code & output results



### 6.4.3.1. “taxrate” table



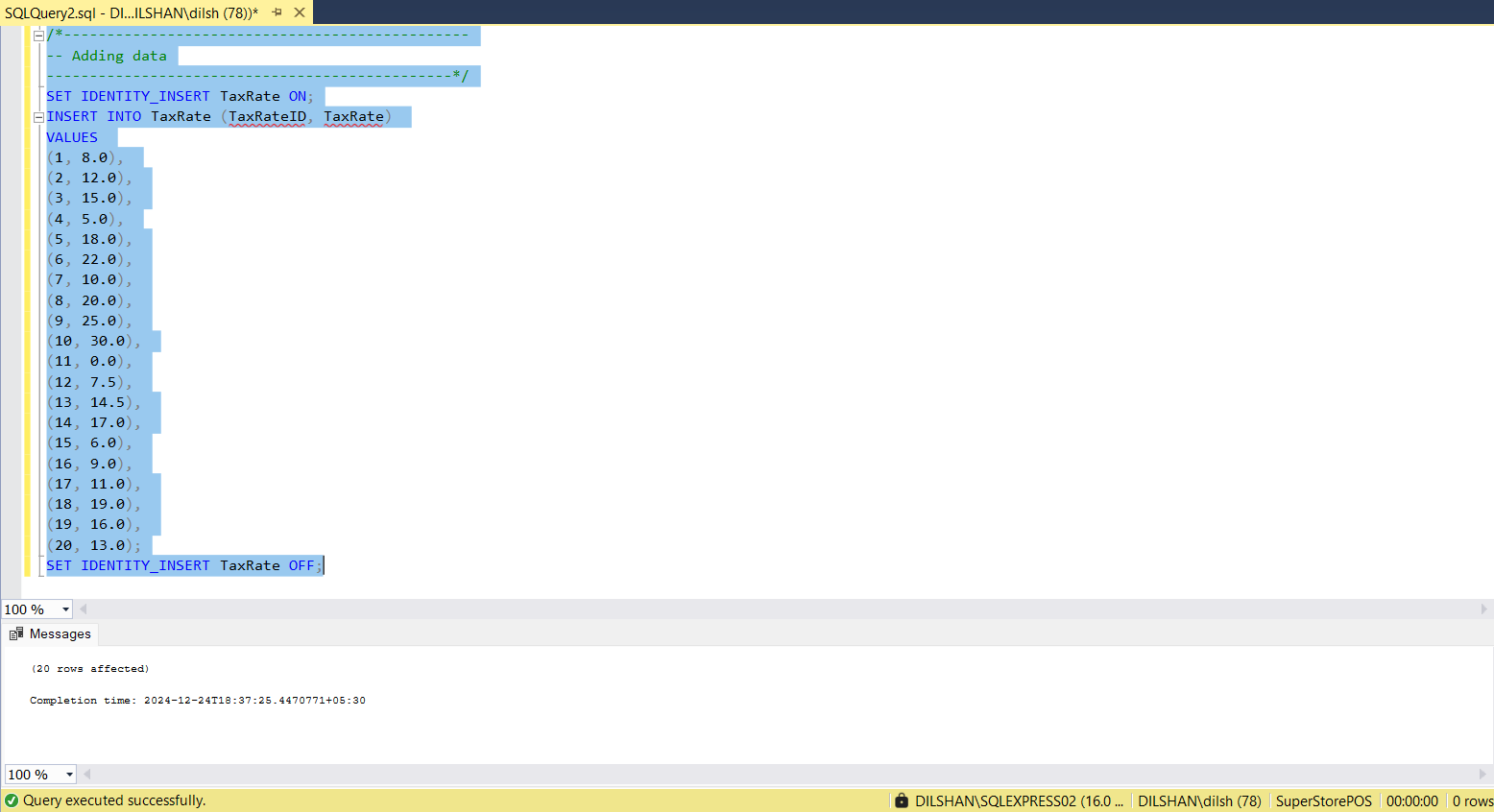
### 6.4.3.2. purpose

Stores applicable tax rates for products, simplifying tax calculations.

### 6.4.3.3. data validation

* **Primary Key Constraint:** (TaxRateID) Unique identifier for tax rates.
* **NOT NULL Constraints:** Ensure (TaxRate) is not empty.
* **CHECK Constraints:**
* Restricts (TaxRate) to a valid percentage (0-100).

### 6.4.3.4. code & output results



### 6.4.4.1. “product” table

### 6.4.4.2. purpose

Manages product information, including category, brand, pricing, and tax rates. Ensures active product status.

### 6.4.4.3. data validation

* **Primary Key Constraint:** (ProductID) Ensures unique identification of products.
* **Foreign Key Constraints:** (CategoryID), (BrandID), (TaxRateID) Link products table to Category, Brand, and TaxRate tables.
* **NOT NULL Constraints:** Prevent missing essential details like (ProductName), (UnitPrice), (ReorderLevel) and (IsActive) status.
* **CHECK Constraints:**
* Ensure (ProductName) is valid.
* Validate (UnitPrice) and (ReorderLevel) are non-negative.

### 6.4.4.4. code & output results

### 6.4.5.1. “wherehose” table

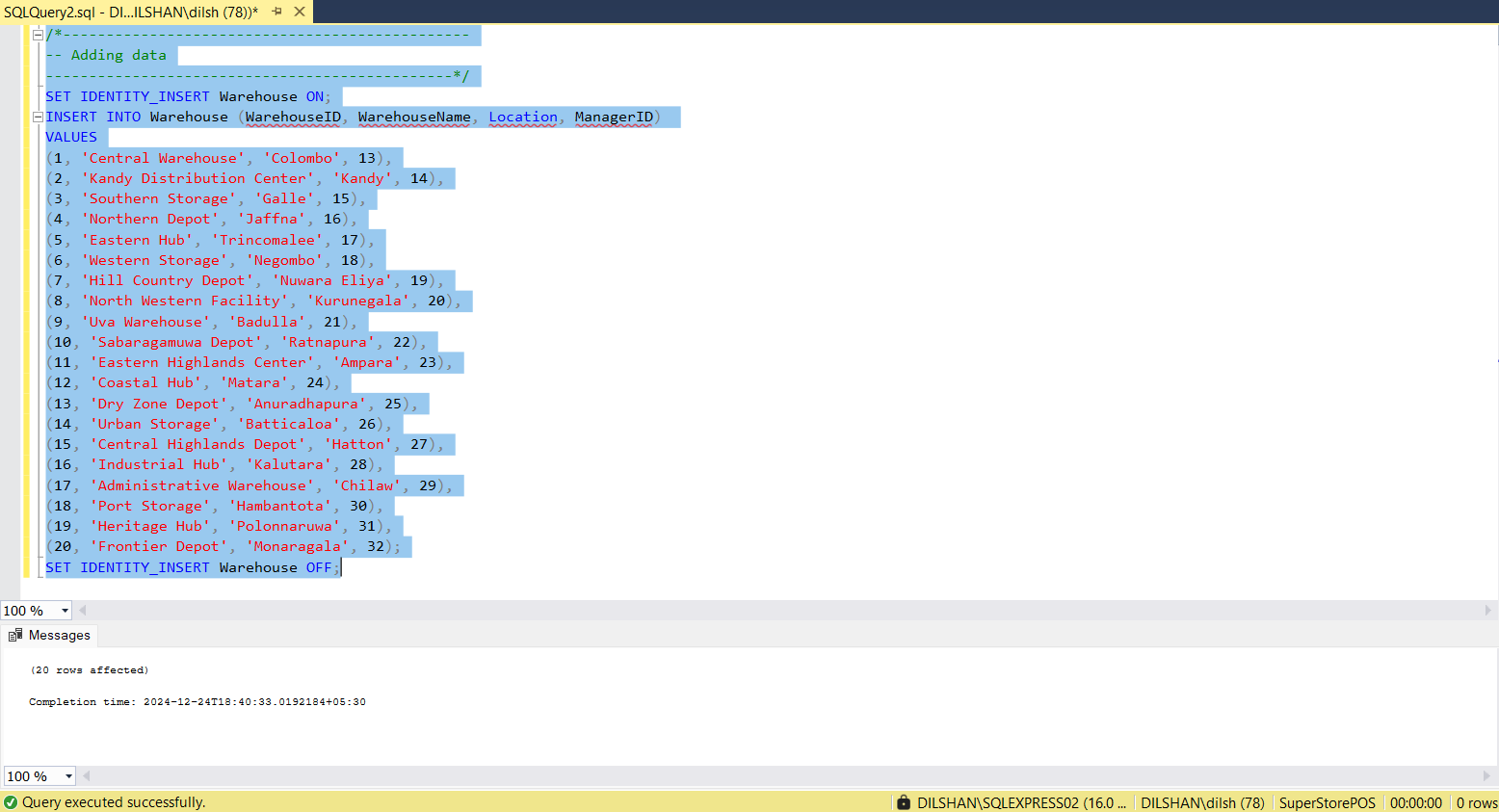
### 6.4.5.2. purpose

Tracks warehouse details, including location and manager assignment.

### 6.4.5.3. data validation

* **Primary Key Constraint:** (WarehouseID) Unique identifier for warehouses.
* **Foreign Key Constraint:** (ManagerID) Links managers to warehouses, allowing NULL.
* **NOT NULL Constraints:** Ensure (WarehouseName) is not empty.
* **CHECK Constraints:**
* Check (WarehouseName) is valid.

### 6.4.5.4. code & output results



### 4.4.6.1. “stock” table

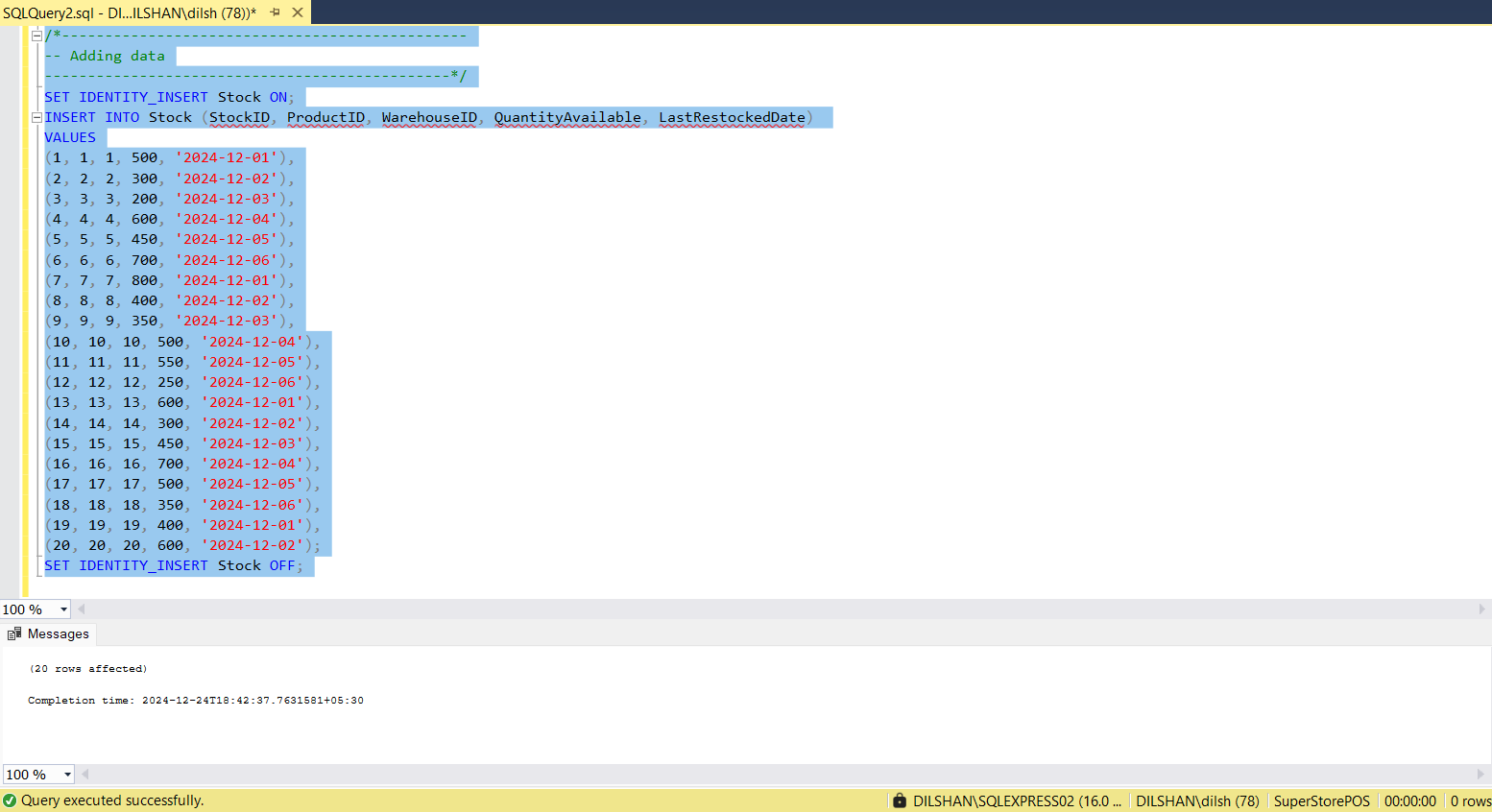
### 4.4.6.2. purpose

Tracks product quantities available in warehouses and the last restocking date.

### 4.4.6.3. data validation

* **Primary Key Constraint:** (StockID) Unique identifier for stock records.
* **Foreign Key Constraints:** (ProductID), (WarehouseID) Link stocks to Product and Warehouse.
* **NOT NULL Constraints:** Ensure (QuantityAvailable) is not empty.
* **CHECK Constraints:**
* Ensure (QuantityAvailable) is non-negative.
* Validate (LastRestockedDate) is not in the future.

### 4.6.6.4. code & output results



## discount management

### 6.5.1.1. “discount” table

### 6.5.1.2. purpose

Manages discounts, ensuring valid date ranges and percentage constraints.

### 6.5.1.3. data validation

* **Primary Key Constraint:** (DiscountID) Unique identifier for discounts.
* **NOT NULL Constraints:** Prevent missing essential details like (DiscountName), (DiscountType), (StartDate) and (EndDate).
* **CHECK Constraints:**
* Ensures (EndDate) is not before (StartDate).

### 6.5.1.4. code & output results

### 6.5.2.1. “productdiscount” table

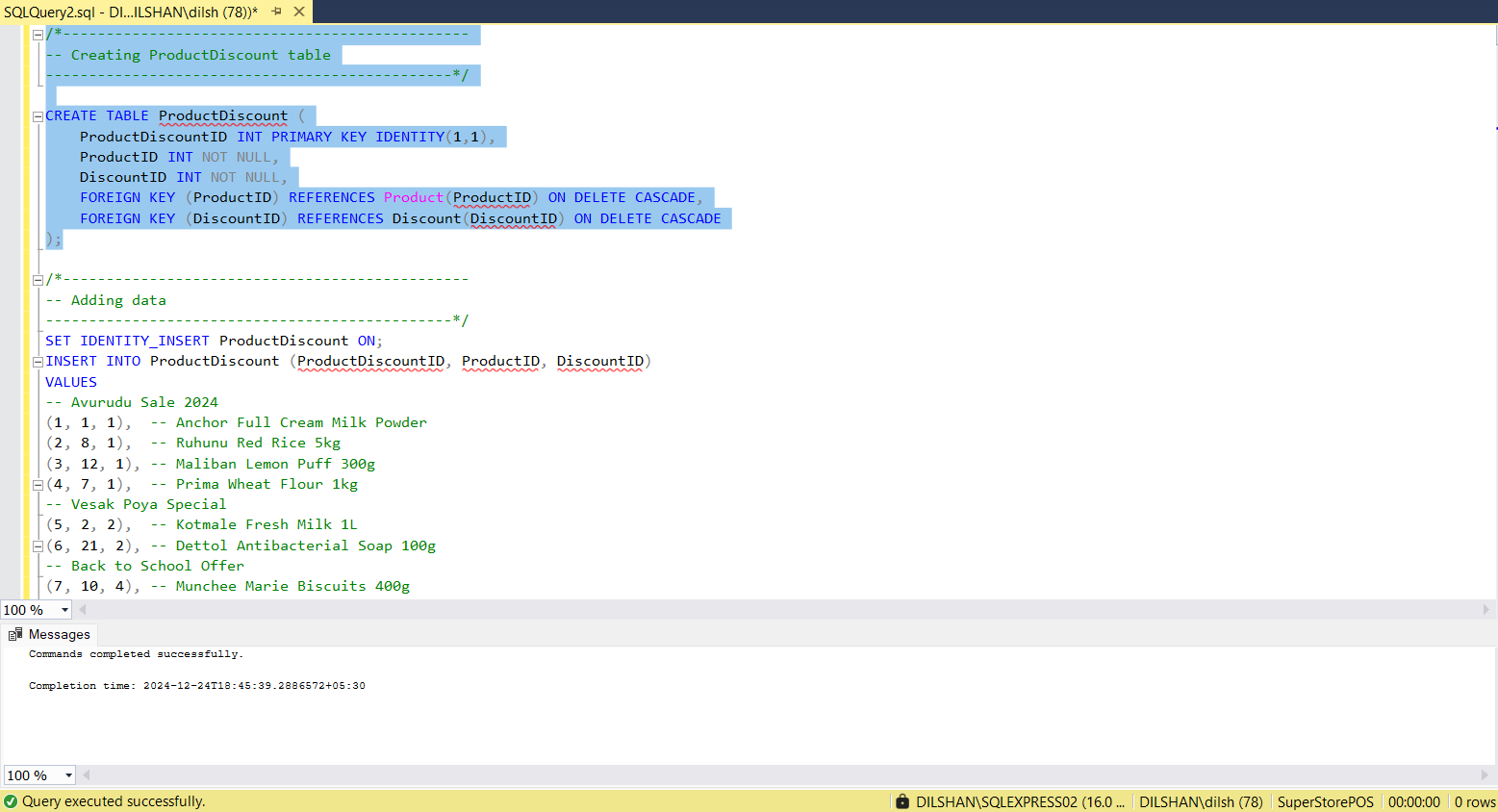
### 6.5.2.2. purpose

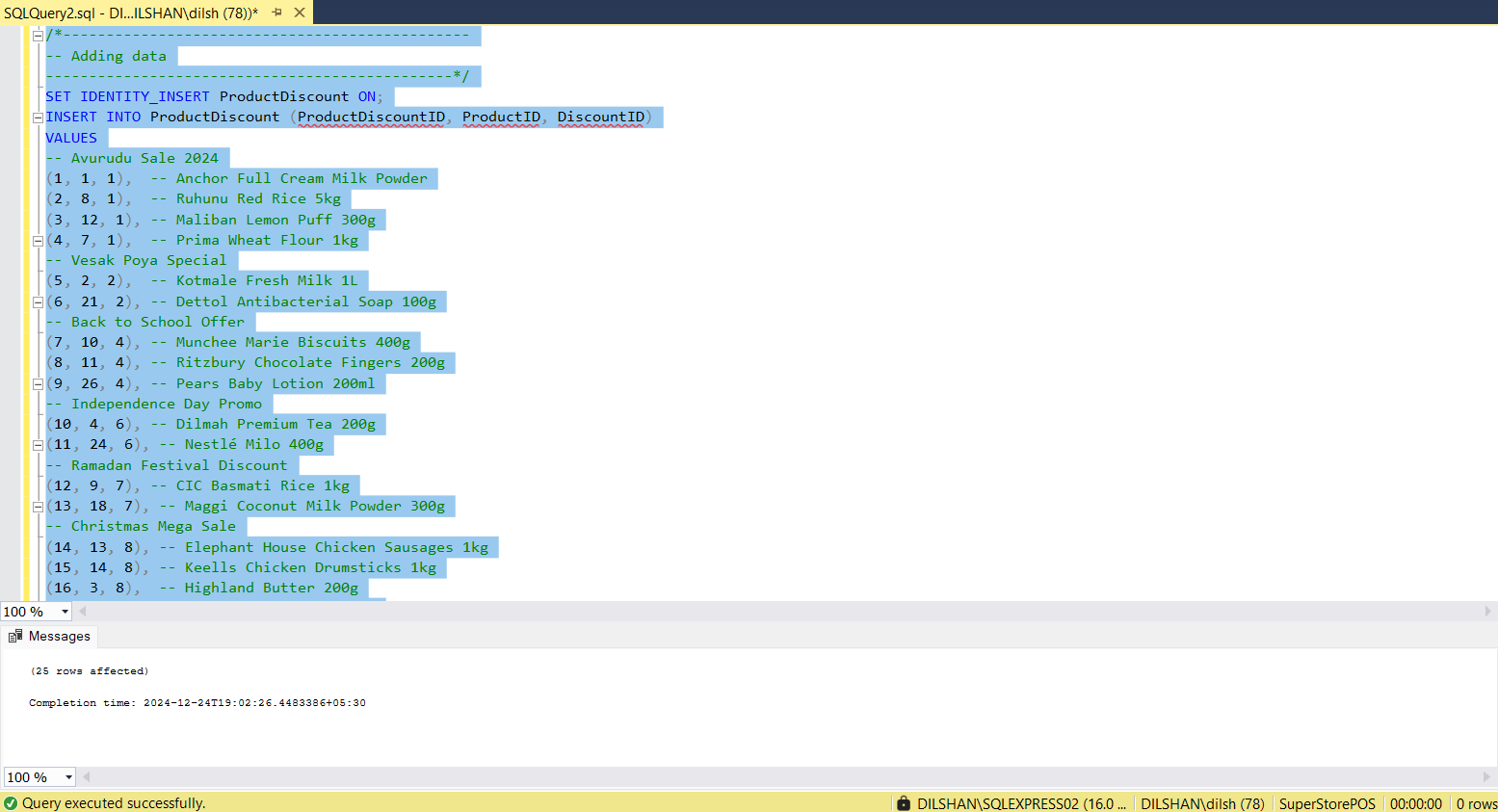
Maps products to specific discounts, enabling easy application of discounts.

### 6.5.2.3. data validation

* **Primary Key Constraint:** (ProductDiscountID) Unique identifier for discount mappings.
* **Foreign Key Constraints:** (ProductID), (DiscountID) Link mappings to Product and Discount.

### 6.5.2.4. code & output results





## sales management

### 6.6.1.1 “paymentmethod” table

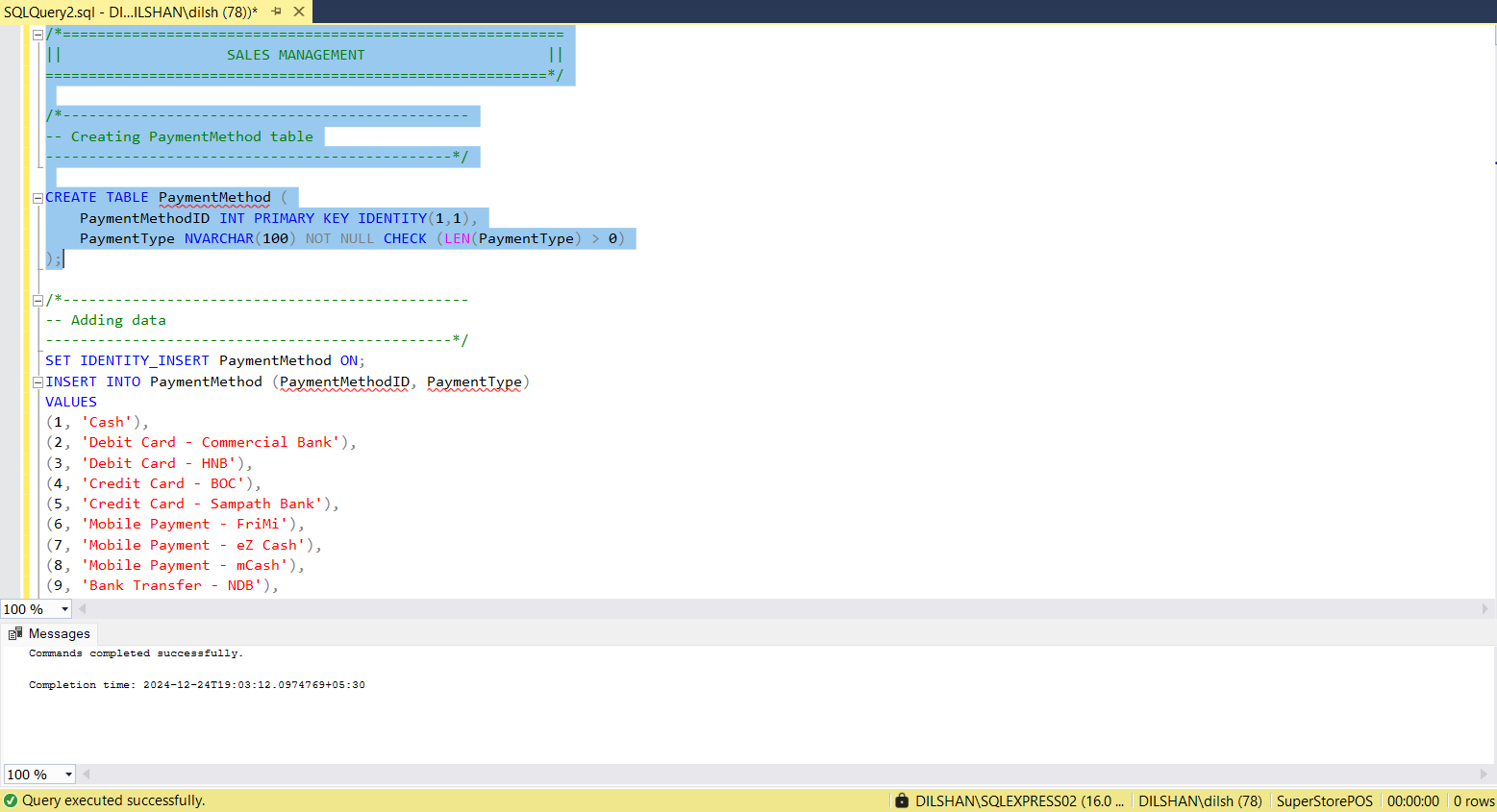
### 6.6.1.2. purpose

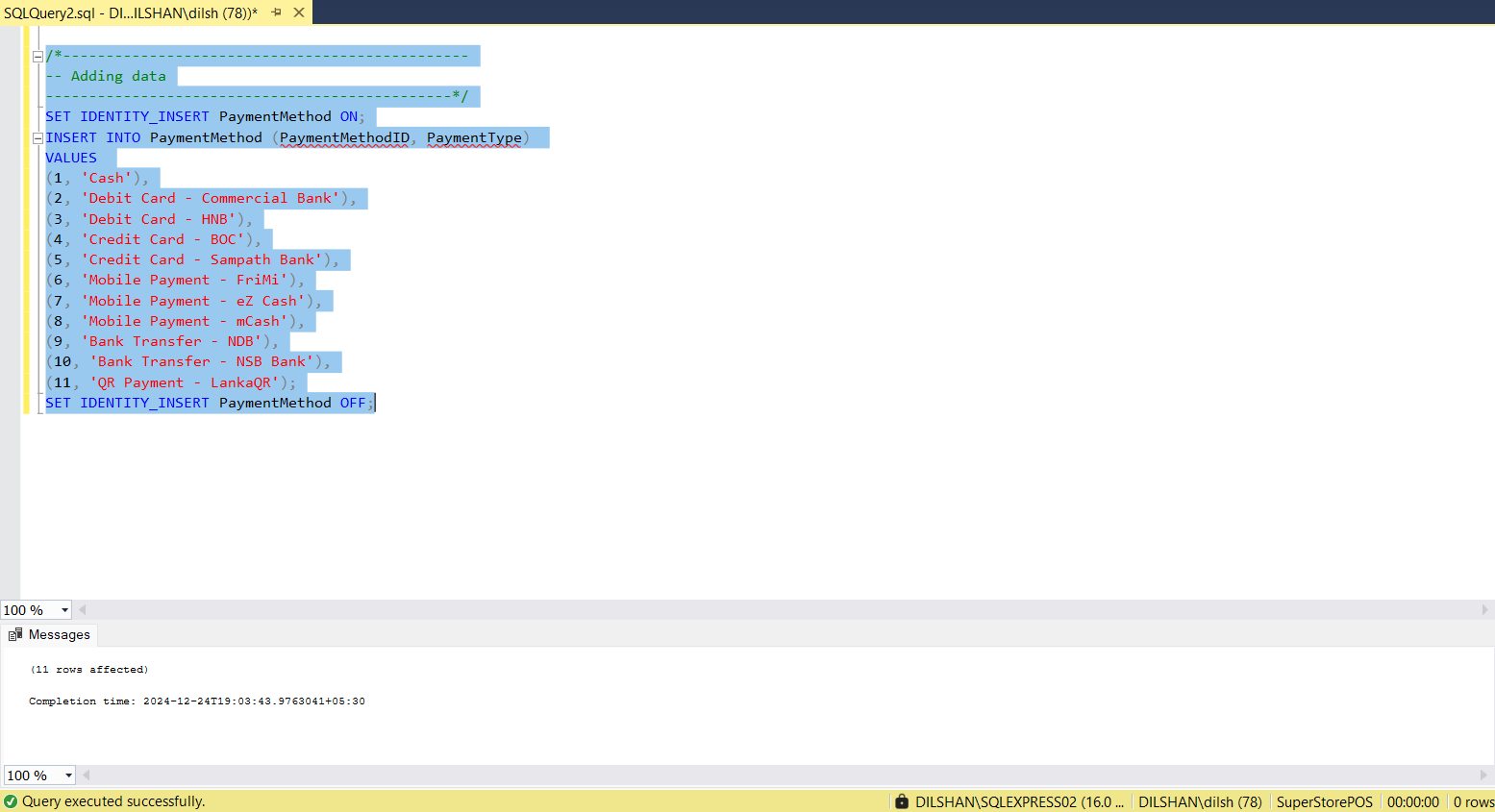
Lists accepted payment methods, ensuring proper classification of transactions.

### 6.6.1.3. data validation

* **Primary Key Constraint:** (PaymentMethodID) Unique identifier for payment methods.
* **NOT NULL Constraints:** Ensure (PaymentType) is not empty.
* **CHECK Constraints:**
* Check (PaymentType) is valid.

### 6.6.1.4. code & output results





### 6.6.2.1. “salestransaction” table

### 6.6.2.2. purpose

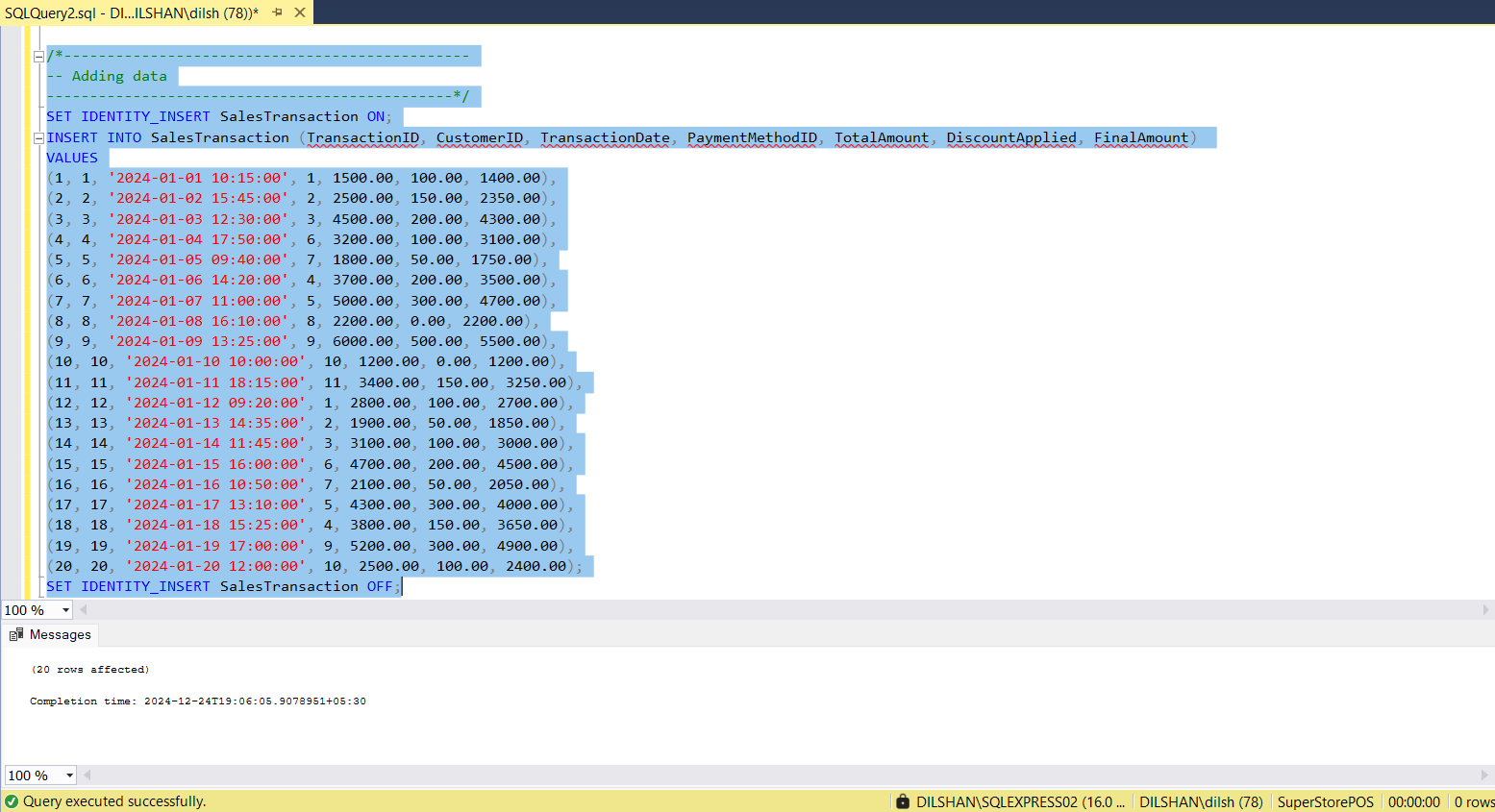
Records sales transactions, including customer, payment method, and final amounts.

### 6.6.2.3. data validation

* **Primary Key Constraint:** (TransactionID) Uniquely identifies transactions.
* **Foreign Key Constraints:** (CustomerID), (PaymentMethodID) Link transactions to Customer and PaymentMethod tables.
* **NOT NULL Constraints:** Prevent missing essential details (TransactionDate), (TotalAmount) and (FinalAmount).
* **CHECK Constraints:**

Ensure (TotalAmount), (DiscountApplied), and (FinalAmount) are non-negative.

### 6.6.2.4. code & output results



### 6.6.3.1. “salestransaction” table

### 6.6.3.2. purpose

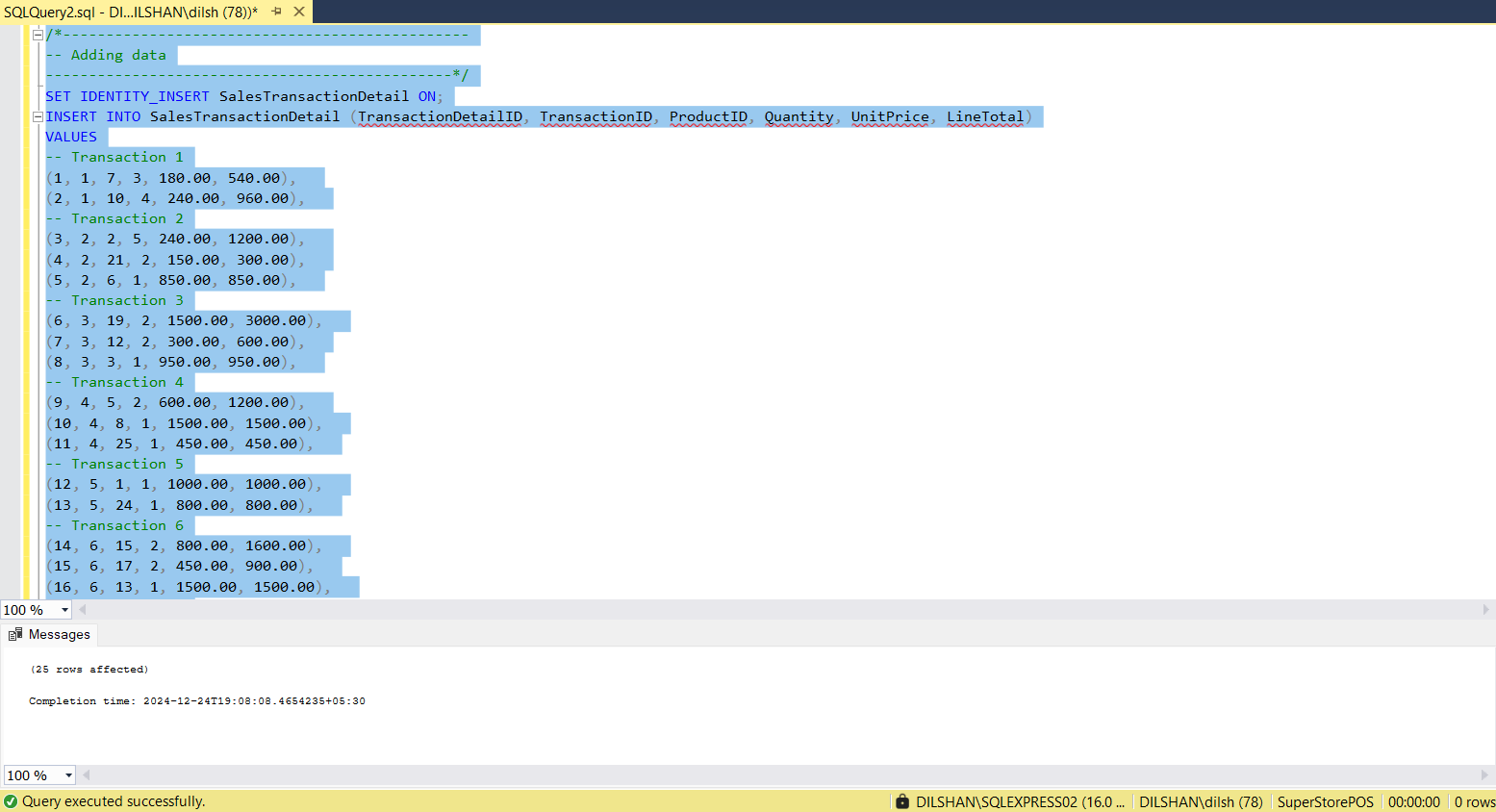
Break down each sales transaction into individual product-level details.

### 6.6.3.3. data validation

* **Primary Key Constraint:** (TransactionDetailID) Unique identifier for transaction details.
* **Foreign Key Constraint:** (TransactionID), (ProductID) Link details to SalesTransaction and Product.
* **NOT NULL Constraints:** Ensure (Quantity), (UnitPrice) and (LineTotal) are not empty.
* **CHECK Constraints:**

Ensure (Quantity), (UnitPrice), and (LineTotal) are positive values.

### 6.6.3.4. code & output results



## database diagrameA computer screen shot of a computer flow chart Description automatically generated with medium confidence

# procedures & triggers

## Stored Procedures

**sp\_InsertCustomer**

The sp\_InsertCustomer procedure will put a new customer into the database and log such an operation into an audit log at the same time. It requires various factors as input, which include: first name, last name, and email of the customer, which are necessary, and other details such as phone number and address which are optional. It then inserts the customer record in the Customer table, chooses the new CustomerID with SCOPE\_IDENTITY(), and logs the operation to the AuditLog table. Error handling is in place to report any error in execution through the procedure.

**sp\_GenerateSalesReport**

This stored procedure provides a sales report for any specified time. This stored procedure includes two parameters: one for the date commencing the period of the report, and one for the date concluding the report period. This stored method collects all the transactions within the supplied range, together with the transaction ID, customer name, product details, quantity, and money per transaction. Also, the action of report production gets noted in the AuditLog table for tracking accountability purposes. Error management (Wikipedia, 2024) has also been provided to control possible execution issues.

**sp\_AssignProductDiscount**

The sp\_AssignProductDiscount process guarantees a product is awarded a discount only once. It takes a ProductID and a DiscountID as inputs. Before making the assignment, it verifies whether the combination already exists in the ProductDiscount database. If not, it does the insert of the new association and returns a success message. If the combination already exists, the method advises the user. Errors are recorded and reported using an error-handling block.

## Triggers

**trg\_UpdateStockAfterSale**

The trg\_UpdateStockAfterSale trigger updates the quantities in stock after the recording of a new sale in the SalesTransactionDetail database. It decreases the quantity of the sold product in the Stock table following an entry, according to the information in the inserted record. Additionally, it logs in the AuditLog table the operation performed to update the stock. This trigger is designed to ensure that, after each sale, the stock is updated correctly.

**trg\_ReorderNotification**

The trg\_ReorderNotification trigger will constantly watch any updates to the Stock table and log an alert if the available quantity of any product goes below the reorder level. This trigger will be invoked following the execution of an update operation to make sure that any low-stock condition will promptly be noted in the AuditLog table for further action, as this would signal serious stock shortages and therefore enhance inventory efficiency.

**trg\_LogCustomerDeletion**

The trg\_LogCustomerDeletion trigger writes into the AuditLog database facts about which customer records have been removed. Whenever a client is deleted from the client table, the trigger extracts data from the DELETED pseudo-table, including the customer name and email, and logs an audit trail. This offers traceability of all client data deletions and promotes system responsibility.

# functions and views

## calculate total sales

### 8.1.1. fn\_calculatetotalsales

A screenshot of a computer

Description automatically generated

### 8.1.2. purpose

Calculates the total amount spent by a customer on all transactions.

### 8.1.3. inputs & returns

Input - CustomerID (INT) - The ID of the customer.

Return - The total amount spent by the customer.

### A screenshot of a computer Description automatically generated8.1.4. code & output results

## calculate product sales

### 8.2.1. FN\_calculatorproductsales

A screenshot of a computer

Description automatically generated

### 8.2.2. purpose

Calculates the total sales for a given product.

### 8.2.3. inputs & returns

Input - ProductID (INT) - The ID of the product.

Return - DECIMAL - The total sales for the product.

### 8.2.4. code & output results

A screenshot of a computer

Description automatically generated

## get product availability

### A screenshot of a computer Description automatically generated8.3.1. fn\_getproductavailability

### 8.3.2 purpose

Returns the available quantity of a product in a specific warehouse.

### 8.3.3. inputs & returns

Inputs - ProductID (INT) - The ID of the product.

WarehouseID (INT) - The ID of the warehouse.

Return - The available quantity of the product in the warehouse.

### A screenshot of a computer Description automatically generated8.3.4. code & output results

## get discount price

### 8.4.1. fn\_getdiscountprice

A screenshot of a computer

Description automatically generated

### 8.4.2. purpose

Calculate the price of a product after applying for a discount.

### 8.4.3. inputs & returns

Inputs – ProductID (INT) - The ID of the product.

DiscountID (INT) - The ID of the discount to be applied.

Return - The discounted price of the product.

### 8.4.4. code & output results

A screenshot of a computer

Description automatically generated

## customer sales summery

### 8.5.1 vw\_customersalessummery

### 8.5.2. purpose

Summary view of total sales and transaction count for each customer.

### 8.5.3. code & output results

A screenshot of a computer

Description automatically generated

## product sales overview

### 8.6.1 vw\_productsalesoverview

A screenshot of a computer

Description automatically generated

### 8.6.2. purpose

Provides an overview of products with total sales and current stock.

### 8.6.3. code & outputs results

A screenshot of a computer

Description automatically generated

## employee role summery

### 8.7.1.vw\_employeerolesummery

A screenshot of a computer

Description automatically generated

### 8.7.2. purpose

Provides employees with details along with their roles and contact info.

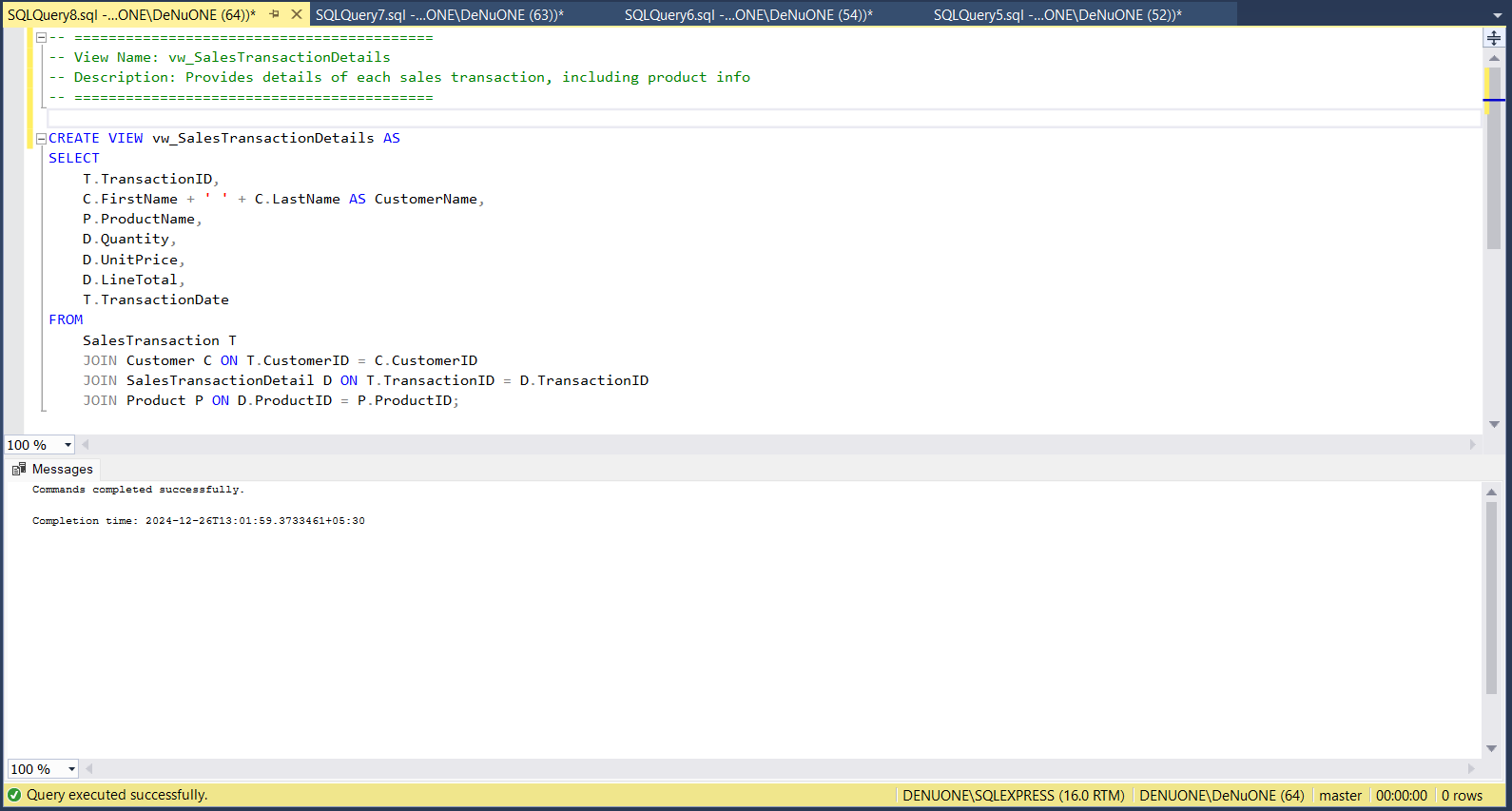
### 8.7.3. code & outpu results

A screenshot of a computer

Description automatically generated

## sales transaction details

### 8.8.1. vw\_salestransactiondetails



### 8.8.2. purpose

Provides details of each sales transaction, including product info.

### 8.8.3. code & output results

# data security measures

**Data Encryption**

Data encryption ensures that sensitive information stored in the database is inaccessible to unauthorized users. For the supermarket POS system:

* **Sensitive Data Encryption**: Encrypt customer personal details, payment methods, and transaction records using advanced encryption standards, such as AES-256.
* **Encrypted Communication**: SSL/TLS for secure data transmission between client and server, preventing man-in-the-middle attacks.
* **Database Encryption**: Implement Transparent Data Encryption (TDE) at the database level to secure data at rest, ensuring that even backups and live data are safe from breaches.

**User Roles**

User roles define access levels for database users based on their responsibilities:

* **Role-based Access Control (RBAC):** Assign roles such as Admin, Manager, Cashier, and Analyst.

Admin: Full access, including modifying database schemas and managing permissions.

Manager: Access to reports and sales data is restricted from schema changes.

Cashier: Limited to sales and billing modules only.

Analyst: Can only analyze consolidated sales and stock data.

* Regularly review and update roles to ensure minimal privilege and avoid unauthorized access.

**Permissions**

Permissions control what users can do:

* **Granular Permissions**: Assign detailed permissions for each role, like SELECT, INSERT, UPDATE, and DELETE.
* **Limit High-Risk Operations**: Limit permissions for data deletion and schema alteration to a few users.
* **Audit Logs**: Enable auditing for all permission changes to track unauthorized modifications.

**Discussion on Potential Risks**

**1. Data Breaches**

A data breach occurs when sensitive information is accessed by unauthorized individuals, leading to financial losses, reputation damage, and regulatory penalties.

**Key Risks:**

* Weak passwords or unencrypted data storage.
* Exposure of sensitive information via unsecured APIs.

**Mitigation Strategies:**

* Use strong password policies (e.g., 12+ characters, including uppercase, lowercase, numbers, and special characters).
* Implement multi-factor authentication (MFA) for database access.
* Use tools like firewalls and intrusion detection/prevention systems (IDS/IPS) to secure the network perimeter.

**2. Insider Threats**

Employees or contractors with legitimate access to the database could misuse their access for personal gain or malicious purposes.

**Key Risks:**

* Unauthorized export or manipulation of data.

**Mitigation Strategies:**

* Conduct background checks before granting access.
* Enable database activity monitoring (DAM) to track and flag unusual behavior, such as large data exports.
* Use time-limited access for sensitive roles (e.g., temporary admin privileges).

**3. Weak Security Configurations**

Misconfigured databases can leave them exposed to attacks.

**Examples:**

* Default passwords left unchanged.
* Databases accessible over the internet without proper restrictions.

**Mitigation Strategies:**

* Disable unused services and close unnecessary ports.
* Regularly update database systems to address security vulnerabilities.
* Follow CIS Benchmarks or other hardening guides for secure configurations.

**4. SQL Injection**

Attackers can exploit vulnerable input fields in applications to execute malicious SQL commands (Port Swigger, 2024).

**Key Risks:**

* Unauthorized data extraction, manipulation, or deletion.

**Mitigation Strategies:**

* Use parameterized queries or ORM frameworks to prevent direct query injection.
* Sanitize and validate user inputs rigorously.
* Regularly perform penetration testing to identify and fix vulnerabilities.

**5. Backup Theft**

Database backups often contain sensitive data and are a common target for attackers.

**Key Risks:**

* Unencrypted backups can be copied and exploited.

**Mitigation Strategies:**

* Encrypt backups using strong encryption methods.
* Store backups in secure, access-controlled environments, both on-premises and in the cloud.
* Use off-site backup replication with controlled access.

**6. Malware and Ransomware**

Malware attacks can encrypt or corrupt data, disrupting operations and leading to data loss.

**Key Risks:**

* Ransom demands for access to encrypted data.

**Mitigation Strategies:**

* Install and regularly update anti-malware tools.
* Conduct regular backups and verify the ability to restore data from them.
* Educate employees about phishing and other malware delivery methods.

**7. Denial of Service (DoS) Attacks**

An attacker could overwhelm the database server, rendering it unavailable.

**Key Risks:**

* Loss of access to critical business systems.

**Mitigation Strategies:**

* Implement rate limiting and traffic filtering to mitigate DoS attacks.
* Use load balancers and redundant systems to handle traffic surges.

# critical analysis

## Highlight the 'key features' of our solution [strengths]

1. Drawing ER Diagram (Entity-Relationship Diagram)

* **Clear Data Structure Representation:** By visualizing the entities and relationships in your system, the ER diagram aids stakeholders in comprehending the database's structure.
* **Integrity and Consistency:** It aids in making sure that one-to-many and many-to-many data connections are appropriately specified, guaranteeing accurate and consistent data mapping in the finished product.
* **Communication Ease:** Developers, designers, and stakeholders can use the diagram as a guide for development and communication.

1. Data Mapping

* **Data Transformation and Flow:** Guarantees the proper flow of data from one system element (also called target) to another while applying different transformations of the source data to the target data.
* **Consistency in Data Types:** Mappings (talend a quick company, 2024) done properly guarantee that the source and destination systems have consistent data types, formats, and values, hence data inconsistency is avoided.
* **Data Quality Control:** Achieving a clear mapping business process makes it possible to easily tell if there are potential errors or misaAutomated Workflows and Actions: Stored procedures eliminate the requirement for application-level coding by enabling the execution of preset, reusable logic, guaranteeing consistent system behavior.
* Enforcement of Business Rules: When certain database actions (like inserts, updates, or deletions) take place, triggers can automatically enforce business rules (such prohibiting invalid data insertion) to guarantee data integrity and compliance.
* Performance Optimization: By moving business logic to the database layer, you may minimize data processing at the application level, which lowers network traffic and boosts performance.
* Error Handling: To ensure that unforeseen problems are effectively handled, procedures and triggers might incorporate error handling techniques.
* lignment that can make the data lose quality.

1. Data Dictionary

* **Centralized Data Documentation**: The data dictionary (Science Direct, 2024) describes in detail each element of the database, including tables, columns, data types, and constraints ensuring that there are no hidden meanings.
* **Improved Collaboration:** Since all the participants involved in the designing or the maintenance of the database system use a single base of information, it improves collaboration and reduces chances of miscommunication.
* **Facilitates Maintenance:** The data dictionary is a useful tool in case of any problems with the database, which makes subsequent changes and maintenance easier.

1. Data Normalization

* **Decreases Redundancy**: Normalization achieves minimum redundancy of data by structuring information into several inter related tables where each data item is entered only one time.
* **Greater Data Quality**: Through normalization (up to 3NF or beyond), you reduce the chances of various anomalies occurring such as insertion, deletion or update anomalies to better the state of the data.
* **Improved Performance of Queries**: Tables that are efficiently normalized are said to enhance the performance of the queries by eliminating the duplication of data that is not required.

1. Creating Databases and Tables

* **Systematic Storage Construction**: Systematizing database schema design and tables construction enhances logical storage and organization of data.
* **Scalability And Flexibility**: The design gives the provision of scaling up of solution and modification of the schema according to the changing needs of the system.
* **Data Integrity:** The tables are designed with the right number of constraints (primary keys, foreign keys, unique constraints) so that the data is accurate and consistent.

1. Functions and Views in tables

* **Implementation of Custom Business Logic**: Functions make it possible to directly encapsulate business logic in the database, which enhances performance and streamlines the application layer.
* **Simplified Data Retrieval:** Views offer a practical means of displaying data without altering the underlying schema, which facilitates straightforward queries on complicated data.
* **Security and Access Control**: By revealing only the information that is required, views can help limit user access to sensitive data, improving security.

1. Stored Procedures and Triggers

* **Automated Workflows and Actions**: Stored procedures eliminate the requirement for application-level coding by enabling the execution of preset, reusable logic, guaranteeing consistent system behavior.
* **Enforcement of Business Rules**: When certain database actions (like inserts, updates, or deletions) take place, triggers can automatically enforce business rules (such prohibiting invalid data insertion) to guarantee data integrity and compliance.
* **Performance Optimization**: By moving business logic to the database layer, you may minimize data processing at the application level, which lowers network traffic and boosts performance.
* **Error Handling:** To ensure that unforeseen problems are effectively handled, procedures and triggers might incorporate error handling techniques.

## Identify and explain the 'areas that need improvements' [weaknesses]

1. ER Diagrams (Entity-Relationship Diagrams)

Weaknesses:

* **Relationship ambiguity**: Relationships between entities can occasionally be unclear, making it difficult to grasp how various system components interact. For instance, ambiguous many-to-many or one-to-many connections.
* **Overcomplication:** When there are too many entities and relationships in an ER diagram, it might become difficult for stakeholders (BYJU'S, 2024) to understand.
* **Inaccurate cardinalities**: Poor database architecture and inaccurate associations can result from the misuse or nonexistence of suitable cardinalities (such as many-to-many, one-to-many, and one-to-one).
* **Lack of normalization**: Inefficient and duplicated data might result from improperly normalized data models, which are occasionally reflected in ER diagrams.
* **Ignoring business rules**: Complex business rules, constraints, or conditional logic that need to be implemented at the database level are frequently overlooked by ER diagrams.

Improvements:

* Make sure the relationships are explicit and well-defined.
* Simplify intricate diagrams by concentrating on high-level entities or dividing them into smaller sub-diagrams.
* Verify each relationship's cardinality in light of practical needs.
* To prevent redundant data, take normalization principles into account when designing.
* As much as you can, incorporate business rules into the ER diagram.

1. Data Mapping

Weakness

* **Data types that are inconsistent**: When various data sources are mapped, it may result in data types that are incompatible or mismatched (e.g., integer vs. string).
* **Incomplete mapping**: Missing characteristics or tables might result from not always capturing all the data necessary.
* **Absence of mapping documentation**: Team members and developers working on the system may become confused if the mapping procedure is not adequately documented.
* **mistakes in layer mapping**: Inadequate layer mapping (such as database, business logic, and user interface) can result in mistakes and inconsistent data.

Improvements:

* Verify that data types are appropriately aligned across systems and that conversion methods are used when needed.
* Make a thorough mapping document that incorporates all of the data relationships and properties.
* Verify the data mapping on a regular basis with business stakeholders to make sure the design satisfies practical requirements.
* Create a structured procedure for managing data inconsistencies and modifications during the mapping process.

1. Data Dictionary

Weakness

* **Lack of standardization**: Teams may become confused if the data dictionary does not adhere to a common naming practice.
* **Inadequate definitions**: Some fields, tables, and relationships are not well explained, which makes it difficult to grasp their use or purpose.
* **Absence of rules or constraints**: Data dictionaries can lack crucial rules for data validation or restrictions (such unique, foreign keys).
* Outdated or erroneous information The data dictionary may become a source of errors and false information if it is not updated on a regular basis.

Improvements

* Create and implement uniform naming guidelines.
* Give each table, column, constraint, and relationship a precise definition.
* Make that the dictionary contains all applicable validation, rules, and constraints for the data.
* Update the data dictionary frequently to take into account modifications to the data structure or schema.

1. Data Normalization

Weaknesses

* **Over-normalization**: Too many tables may result from excessive normalization, which may slow down queries by requiring too many joins.
* **Under-normalization**: Inadequate normalization can result in abnormalities, redundant data, and trouble preserving consistency.
* **Ignorance of trade-offs**: It's common to ignore the need to strike a balance between normalization and performance considerations (e.g., denormalization for speed).
* **Inconsistent application**: A difficult-to-maintain design may arise from applying normalization inconsistently across tables.

Improvements

* Maintain performance requirements while applying normalization up to the proper level, which is often 3NF.
* Review the schema frequently to make sure normalization is applied correctly.
* When performance optimizations are required, such as for large-scale reporting, take into account denormalization or partial normalization.
* Assess the normalization procedure in light of the database workload and the particular requirements of the application.

1. Creating Database and Tables

Weaknesses

* **accurate relationships between primary and foreign keys**: Inconsistencies in the database and problems with referential integrity might result from improper key configuration.
* Ignoring indexing Performance issues may arise if frequently searched columns are not indexed.
* **Disregarding performance and scalability requirements**: Inadequate table design may function well for little datasets but not well at scale.
* **Absence of nullability constraints or default values**: Certain columns may not have default values or may permit null values in inappropriate places.

Improvements:

* Give primary keys, foreign keys, and any required restrictions careful definitions.
* Take indexing on commonly used columns into account while designing for performance
* Use sharding, partitioning, or other effective data distribution strategies to account for scalability.
* Establish nullability constraints and reasonable defaults to implement business rules at the database level.

1. Functions and Views in Tables

Weaknesses:

* **Performance problems**: Functions and views can have a detrimental impact on performance if they are abused or misused, particularly if they are not optimized or include intricate logic.
* **Poor visibility**: It may be challenging to debug or comprehend the underlying actions due to complex logic within functions or views.
* **Lack of reusability**: Sometimes the design of functions or views makes it difficult to use them in different queries or contexts.

Improvements

* Reduce performance overhead by optimizing functions and views, especially when they are utilized in complex queries.
* Make sure that views and functions have adequate documentation so that other developers can understand them.
* Make an effort to create reusable views and functions that can be used in other application sections.

1. Procedures and Triggers in Tables and Database

Weaknesses

* **Overcomplicating logic:** Complex or business-specific logic may be more difficult to maintain and debug when stored procedures and triggers are used.
* **Absence of transaction management**: When processes and triggers don't manage transactions correctly, it might result in inconsistent data.
* **Performance issues**: If triggers and procedures are not properly optimized, they may cause data manipulation activities to lag.
* **Dependencies that are difficult to control**: Triggers may produce unforeseen side effects or circular dependencies in the database.

Improvements

* To prevent adding needless complexity, keep stored processes and triggers as basic as possible.
* Implement appropriate transaction management and error handling within triggers and methods.
* Analyze how triggers and procedures affect database operations to maximize their performance.
* Make sure stored procedures and triggers don't generate performance bottlenecks or hidden dependencies by reviewing and refactoring them on a regular basis.

## Future implementation

1. ER Diagram Drawing

Future Implementations:

* **include sophisticated connection types**: To better simulate real-world situations, include more intricate relationships such ternary relationships, recursive relationships, and subtype/supertype hierarchies.
* Implementing tools that can automatically create ER diagrams from the database schema or code will help to ensure that design and implementation are in sync.
* Enable real-time collaboration on ER diagrams so that various stakeholders, like as developers and business analysts, can work together to update and discuss the diagram.
* **Tracking changes and managing versions:** To monitor changes over time and facilitate the reversal or review of design choices, use version control for ER diagrams.
* **Combining more tools**: Permit the ER diagram tool to interface with CI/CD, testing, and project management systems.

1. Data Mapping

Future Implementations:

* **Automation of data mapping**: Use algorithms or automated technologies to help with data mapping, which can expedite the process and minimize human mistake. Based on past trends, machine learning may be able to forecast data mapping.
* Enable data lineage features to monitor the movement of data from its source to its destination, improving transparency and comprehension of data transformations.
* **Advanced validation of mapping**: To find discrepancies, mismatches, or missing properties in data mappings, use automated validation. This guarantees error-free and seamless data integration.
* **Assistance with a variety of data sources**: Allow data mapping to accommodate a greater range of data sources (such as NoSQL databases, external APIs, and cloud storage) and formats (such as JSON, XML) as systems grow more complicated.

1. Data Dictionary

Future Implementations:

* **Data dictionary updates that happen automatically**: Create tools that update the data dictionary automatically whenever the schema is modified. This would lessen the possibility of missing or out-of-date documentation.
* **Advanced search and tagging:** Enhance the data dictionary's search capabilities to enable looking for information using relationships, business terms, or data lineage.
* **Features of collaboration**: Allow several team members to participate and check definitions, restrictions, and other metadata by enabling real-time collaboration when changing and evaluating the data dictionary.
* **Connecting to tools for managing metadata**: Provide insights into data consumption, linkages, and impact across systems by integrating the data dictionary with more comprehensive metadata management frameworks.
* **Metadata version control**: Implement version control for the data dictionary to manage several schema versions and monitor changes over time.

1. Data Normalization

Future Implementations:

* Use tools that can automatically recommend different levels of normalization depending on the quantity of the data, query patterns, and performance requirements. These are known as dynamic normalization tools.
* **In favor of hybrid normalization:** Permit hybrid approaches in which some tables are denormalized for performance reasons (e.g., in reporting or OLAP systems) while others can be highly normalized.
* **Denormalization with intelligence**: Use machine learning models to predict when and when denormalization will enhance performance, such as in workloads that require a lot of reading or reporting.
* **Use case-based normalization guidelines**: To balance data integrity and performance, define normalization standards according to particular use cases, such as transactional systems (OLTP) or analytical systems (OLAP).
* **NoSQL database normalization**: Provide normalization strategies appropriate for non-relational databases as NoSQL databases proliferate.

1. Creating Database and Tables

Future Implementations:

* **Automated database schema generation**: Programs that can produce database schema straight from user stories, business requirements, or ER diagrams will help create databases more quickly and with fewer errors.
* **Database sharding and partitioning**: Use automated sharding and partitioning techniques to provide scalability and effective data distribution for extremely big databases.
* **Multi-model database support**: Allow the system to support and smoothly integrate several database types (such as relational, graph, and document-based) as database technology advances.
* **Management of database evolution**: As the system develops over time, handle schema changes with database versioning and migration tools to maintain backward compatibility and seamless version transitions.
* **Cloud database optimization:** Provide capabilities like automatic scalability, backup, and recovery to enable the best possible database creation and scaling in cloud environments.

1. Functions and Views in Tables

Future Implementations:

* **Materialized views**: Add support for materialized views, which store and refresh query results on a regular basis to enhance performance, particularly for intricate, often-used queries.
* Using performance optimization tools that automatically recommend or restructure SQL functions in response to query performance metrics is known as automated function optimization.
* **Serverless functions**: Reduce operational cost by using serverless compute to handle sophisticated functions that can scale dynamically without requiring infrastructure management.
* Implementing a more adaptable data access layer will enable functions and views to be abstracted into reusable modules, making it simpler to integrate them across various application components.
* **AI-assisted views**: By automatically combining frequently accessed data, AI can be used to help construct complicated views based on user behavior or data trends, enhancing query efficiency.

1. Procedures and Triggers in Tables and Database

Future Implementations:

* Transition to event-driven architectures, which allow for dynamic and responsive database interactions by having triggers respond in real-time to system events.
* **Distributed procedures**: Allow stored procedures to be defined and run across distributed databases or microservices architectures as databases grow, guaranteeing consistent logic execution throughout the system.
* **Procedural versioning**: When a new version of the database is released or schema changes are made, incorporate version control for triggers and procedures to guarantee backward compatibility.
* **Automated error handling and logging**: Improve error handling, logging, and monitoring in triggers and stored procedures to aid in system tracking and debugging.
* **AI for trigger optimization**: Make sure that triggers don't result in performance snags by using AI to examine trigger behavior.

# appendices

## creat tables

**CUSTOMER MANAGEMENT**

**Creating customer table**  
  
CREATE TABLE Customer (

CustomerID INT PRIMARY KEY IDENTITY(1,1),

FirstName NVARCHAR(100) NOT NULL CHECK (LEN(FirstName) > 1),

LastName NVARCHAR(100) NOT NULL CHECK (LEN(LastName) > 1),

Email NVARCHAR(255) UNIQUE CHECK (Email LIKE '%@%.%'),

PhoneNumber NVARCHAR(15) CHECK (PhoneNumber LIKE '[0-9]%'),

AddressLine1 NVARCHAR(255),

AddressLine2 NVARCHAR(255),

City NVARCHAR(100),

PostalCode NVARCHAR(20) CHECK (LEN(PostalCode) <= 20),

RegistrationDate DATE NOT NULL DEFAULT GETDATE()

);

**REPORTING AND ADMINISTRATIVE DATA**

**Creating Role table**

CREATE TABLE Role (

RoleID INT PRIMARY KEY IDENTITY(1,1),

RoleName NVARCHAR(100) NOT NULL CHECK (LEN(RoleName) > 0)

);

**Creating Employee table**

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY IDENTITY(1,1),

FirstName NVARCHAR(100) NOT NULL CHECK (LEN(FirstName) > 1),

LastName NVARCHAR(100) NOT NULL CHECK (LEN(LastName) > 1),

RoleID INT NOT NULL,

Email NVARCHAR(255) UNIQUE NOT NULL CHECK (Email LIKE '%@%.%'),

PhoneNumber NVARCHAR(15) CHECK (PhoneNumber LIKE '[0-9]%'),

HireDate DATE NOT NULL CHECK (HireDate <= GETDATE()),

FOREIGN KEY (RoleID) REFERENCES Role(RoleID) ON DELETE CASCADE

);

**Creating Auditlog Table**

CREATE TABLE AuditLog (

LogID INT PRIMARY KEY IDENTITY(1,1),

Action NVARCHAR(255) NOT NULL,

PerformedBy INT NULL, -- Allow NULL values for SET NULL action

ActionTimestamp DATETIME NOT NULL,

TableName NVARCHAR(100),

FOREIGN KEY (PerformedBy) REFERENCES Employee(EmployeeID) ON DELETE SET NULL

);

**Report table**CREATE TABLE Report (

ReportID INT PRIMARY KEY IDENTITY(1,1),

ReportName NVARCHAR(255) NOT NULL CHECK (LEN(ReportName) > 0),

GeneratedBy INT NOT NULL,

GeneratedDate DATETIME NOT NULL DEFAULT GETDATE(),

ReportType NVARCHAR(100) NOT NULL,

ReportData NVARCHAR(MAX),

FOREIGN KEY (GeneratedBy) REFERENCES Employee(EmployeeID) ON DELETE CASCADE

);

**PRODUCT AND STOCK MANAGEMENT**

**Creating category table**

CREATE TABLE Category (

CategoryID INT PRIMARY KEY IDENTITY(1,1),

CategoryName NVARCHAR(100) NOT NULL UNIQUE CHECK (LEN(CategoryName) > 0)

);

**Creating Brand table**

CREATE TABLE Brand (

BrandID INT PRIMARY KEY IDENTITY(1,1),

BrandName NVARCHAR(100) NOT NULL UNIQUE CHECK (LEN(BrandName) > 0)

);

**Creating TaxRate table**

CREATE TABLE TaxRate (

TaxRateID INT PRIMARY KEY IDENTITY(1,1),

TaxRate DECIMAL(5, 2) NOT NULL CHECK (TaxRate >= 0 AND TaxRate <= 100)

);

**Creating Product table**

CREATE TABLE Product (

ProductID INT PRIMARY KEY IDENTITY(1,1),

ProductName NVARCHAR(150) NOT NULL CHECK (LEN(ProductName) > 0),

CategoryID INT NOT NULL,

BrandID INT NOT NULL,

UnitPrice DECIMAL(10, 2) NOT NULL CHECK (UnitPrice >= 0),

TaxRateID INT NOT NULL,

ReorderLevel INT NOT NULL CHECK (ReorderLevel >= 0),

IsActive BIT NOT NULL DEFAULT 1,

FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID) ON DELETE CASCADE,

FOREIGN KEY (BrandID) REFERENCES Brand(BrandID) ON DELETE CASCADE,

FOREIGN KEY (TaxRateID) REFERENCES TaxRate(TaxRateID) ON DELETE CASCADE

);

**Creating Warehouse table**

CREATE TABLE Warehouse (

WarehouseID INT PRIMARY KEY IDENTITY(1,1),

WarehouseName NVARCHAR(150) NOT NULL CHECK (LEN(WarehouseName) > 0),

Location NVARCHAR(255),

ManagerID INT,

FOREIGN KEY (ManagerID) REFERENCES Employee(EmployeeID) ON DELETE SET NULL

);

**Creating Stock table**

CREATE TABLE Stock (

StockID INT PRIMARY KEY IDENTITY(1,1),

ProductID INT NOT NULL,

WarehouseID INT NOT NULL,

QuantityAvailable INT NOT NULL CHECK (QuantityAvailable >= 0),

LastRestockedDate DATE CHECK (LastRestockedDate <= GETDATE()),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE,

FOREIGN KEY (WarehouseID) REFERENCES Warehouse(WarehouseID) ON DELETE CASCADE

);

**DISCOUNT MANAGEMENT MODULE**

**Creating Discount table**

CREATE TABLE Discount (

DiscountID INT PRIMARY KEY IDENTITY(1,1),

DiscountName NVARCHAR(150) NOT NULL,

StartDate DATE NOT NULL,

EndDate DATE NOT NULL,

DiscountPercentage DECIMAL(5, 2),

DiscountType NVARCHAR(50) NOT NULL,

CONSTRAINT CK\_Discount\_DateRange CHECK (EndDate >= StartDate) -- Table-level CHECK constraint

);

**Creating a Product Discount table**CREATE TABLE ProductDiscount (

ProductDiscountID INT PRIMARY KEY IDENTITY(1,1),

ProductID INT NOT NULL,

DiscountID INT NOT NULL,

FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE,

FOREIGN KEY (DiscountID) REFERENCES Discount(DiscountID) ON DELETE CASCADE

);

**SALES MANAGEMENT MODULE**

**Creating Payment Method table**

CREATE TABLE PaymentMethod (

PaymentMethodID INT PRIMARY KEY IDENTITY(1,1),

PaymentType NVARCHAR(100) NOT NULL CHECK (LEN(PaymentType) > 0)

);

**Sales Transaction table**

CREATE TABLE SalesTransaction (

TransactionID INT PRIMARY KEY IDENTITY(1,1),

CustomerID INT,

TransactionDate DATETIME NOT NULL DEFAULT GETDATE(),

PaymentMethodID INT NOT NULL,

TotalAmount DECIMAL(10, 2) NOT NULL CHECK (TotalAmount >= 0),

DiscountApplied DECIMAL(10, 2) CHECK (DiscountApplied >= 0),

FinalAmount DECIMAL(10, 2) NOT NULL CHECK (FinalAmount >= 0),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE SET NULL,

FOREIGN KEY (PaymentMethodID) REFERENCES PaymentMethod(PaymentMethodID) ON DELETE CASCADE

);

**Creating Sales Transaction Detail table**

CREATE TABLE SalesTransactionDetail (

TransactionDetailID INT PRIMARY KEY IDENTITY(1,1),

TransactionID INT NOT NULL,

ProductID INT NOT NULL,

Quantity INT NOT NULL CHECK (Quantity > 0),

UnitPrice DECIMAL(10, 2) NOT NULL CHECK (UnitPrice >= 0),

LineTotal DECIMAL(10, 2) NOT NULL CHECK (LineTotal >= 0),

FOREIGN KEY (TransactionID) REFERENCES SalesTransaction(TransactionID) ON DELETE CASCADE,

FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE

);

## Sample Data

**CUSTOMER MANAGEMENT**

**Adding data to Customer table**

SET IDENTITY\_INSERT Customer ON;

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, PhoneNumber, AddressLine1, AddressLine2, City, PostalCode, RegistrationDate)

VALUES

(1, 'Kumar', 'Perera', 'kumar.perera@example.com', '0771234567', '123 Temple Road', 'Apartment 4B', 'Colombo', '00100', '2023-01-15'),

(2, 'Nuwan', 'Fernando', 'nuwan.fernando@example.com', '0719876543', '45 Beach Drive', NULL, 'Galle', '80000', '2023-02-10'),

(3, 'Amali', 'Wijesinghe', 'amali.wijesinghe@example.com', '0723456789', '78 Lake View', 'Suite 3', 'Kandy', '20000', '2023-03-05'),

(4, 'Saman', 'Silva', 'saman.silva@example.com', '0751234560', '89 High Street', NULL, 'Matara', '81000', '2023-04-12'),

(5, 'Lakshmi', 'De Silva', 'lakshmi.desilva@example.com', '0762345671', '15 Palm Grove', 'Near Park', 'Negombo', '11500', '2023-05-18'),

(6, 'Ruwan', 'Jayasinghe', 'ruwan.jayasinghe@example.com', '0783456782', '52 Hill Crest', 'Flat 6A', 'Nuwara Eliya', '22200', '2023-06-07'),

(7, 'Chathura', 'Karunaratne', 'chathura.k@example.com', '0709876544', '102 Green Lane', NULL, 'Jaffna', '40000', '2023-07-02'),

(8, 'Dilini', 'Gunasekara', 'dilini.g@example.com', '0775671234', '56 River Side', 'Opposite School', 'Batticaloa', '30000', '2023-08-19'),

(9, 'Thilini', 'Ratnayake', 'thilini.ratnayake@example.com', '0743456789', '28 Mountain Pass', NULL, 'Badulla', '90000', '2023-09-03'),

(10, 'Prasanna', 'Abeywickrama', 'prasanna.abeywickrama@example.com', '0712349876', '16 Lotus Street', 'Near Temple', 'Kurunegala', '60000', '2023-10-01'),

(11, 'Harsha', 'Senanayake', 'harsha.senanayake@example.com', '0721234568', '9 Orchid Path', NULL, 'Ratnapura', '70000', '2023-10-15'),

(12, 'Shanika', 'Dias', 'shanika.d@example.com', '0784567890', '67 Cinnamon Drive', NULL, 'Trincomalee', '31000', '2023-11-20'),

(13, 'Anjana', 'Bandara', 'anjana.bandara@example.com', '0702345679', '34 Jasmine Avenue', 'Apartment 5C', 'Anuradhapura', '50000', '2023-12-01'),

(14, 'Kavindu', 'Ekanayake', 'kavindu.ekanayake@example.com', '0753456781', '77 Sunset Boulevard', 'Flat 2B', 'Polonnaruwa', '51000', '2023-12-05'),

(15, 'Nadeeka', 'Rajapakse', 'nadeeka.r@example.com', '0715678901', '101 Palm Court', 'Opposite Market', 'Hambantota', '82000', '2023-12-08'),

(16, 'Sunil', 'Herath', 'sunil.herath@example.com', '0767890123', '12 Garden Path', NULL, 'Kalutara', '12000', '2023-12-12'),

(17, 'Malith', 'Gamage', 'malith.gamage@example.com', '0778901234', '24 Cliff Road', 'Suite 101', 'Ampara', '32000', '2023-12-15'),

(18, 'Rashmi', 'Liyanage', 'rashmi.liyanage@example.com', '0759012345', '88 Oak Lane', NULL, 'Puttalam', '61000', '2023-12-20'),

(19, 'Janaka', 'Dissanayake', 'janaka.d@example.com', '0701237890', '40 Main Street', 'Building A', 'Vavuniya', '43000', '2023-12-25'),

(20, 'Ishara', 'Peiris', 'ishara.peiris@example.com', '0713456782', '17 Cherry Blossom', NULL, 'Kilinochchi', '44000', '2023-12-30');

SET IDENTITY\_INSERT Customer OFF;

**EMPLOYEE AND ROLE MANAGEMENT**

**Adding data to Role table**

SET IDENTITY\_INSERT Role ON;

INSERT INTO Role (RoleID, RoleName)

VALUES

(1, 'Admin'),

(2, 'Cashier'),

(3, 'Manager'),

(4, 'Inventory Supervisor'),

(5, 'Sales Representative'),

(6, 'Accountant'),

(7, 'Warehouse Manager'),

(8, 'Customer Service Representative'),

(9, 'IT Support Specialist'),

(10, 'Marketing Executive'),

(11, 'Delivery Coordinator');

SET IDENTITY\_INSERT Role OFF;

**Adding data to Employee table**

SET IDENTITY\_INSERT Employee ON;

INSERT INTO Employee (EmployeeID, FirstName, LastName, RoleID, Email, PhoneNumber, HireDate)

VALUES

-- Admins

(1, 'Samantha', 'Wijeratne', 1, 'samantha.wijeratne@pos.lk', '0711234567', '2022-01-05'),

(2, 'Dilshan', 'Edirisinghe', 1, 'dilshan.edirisinghe@pos.lk', '0741234567', '2022-05-15'),

-- Cashiers

(3, 'Dinesh', 'Kumarasinghe', 2, 'dinesh.kumarasinghe@pos.lk', '0772345678', '2022-03-12'),

(4, 'Sachini', 'Rajapaksa', 2, 'sachini.rajapaksa@pos.lk', '0750123456', '2022-04-08'),

-- Managers

(5, 'Tharindu', 'Jayasinghe', 3, 'tharindu.jayasinghe@pos.lk', '0703456789', '2021-05-10'),

(6, 'Ruwan', 'Senanayake', 3, 'ruwan.senanayake@pos.lk', '0717890123', '2023-01-10'),

-- Inventory Supervisors

(7, 'Chamodi', 'Fernando', 4, 'chamodi.fernando@pos.lk', '0784567890', '2022-06-15'),

(8, 'Isuru', 'Bandara', 4, 'isuru.bandara@pos.lk', '0789012345', '2023-03-01'),

-- Sales Representatives

(9, 'Kasun', 'Perera', 5, 'kasun.perera@pos.lk', '0755678901', '2022-07-20'),

(10, 'Thilini', 'Wijesinghe', 5, 'thilini.wijesinghe@pos.lk', '0785678901', '2022-09-12'),

-- Accountants

(11, 'Nishadi', 'De Silva', 6, 'nishadi.desilva@pos.lk', '0746789012', '2021-08-25'),

(12, 'Sanduni', 'Wickramasinghe', 6, 'sanduni.wickramasinghe@pos.lk', '0758901234', '2022-12-12'),

-- Warehouse Managers

(13, 'Ajith', 'Kumarasinghe', 7, 'ajith.kumarasinghe@pos.lk', '0771234561', '2023-01-15'),

(14, 'Kavinda', 'Perera', 7, 'kavinda.perera@pos.lk', '0719876523', '2022-05-18'),

(15, 'Nadeesha', 'Fernando', 7, 'nadeesha.fernando@pos.lk', '0703456729', '2022-08-25'),

(16, 'Priyanka', 'De Silva', 7, 'priyanka.desilva@pos.lk', '0742345671', '2021-10-10'),

(17, 'Sunil', 'Weerasinghe', 7, 'sunil.weerasinghe@pos.lk', '0784567888', '2022-03-30'),

(18, 'Rasika', 'Jayawardena', 7, 'rasika.jayawardena@pos.lk', '0775678998', '2022-06-12'),

(19, 'Tharaka', 'Senanayake', 7, 'tharaka.senanayake@pos.lk', '0746789001', '2022-09-17'),

(20, 'Chathura', 'Gunathilaka', 7, 'chathura.gunathilaka@pos.lk', '0787891234', '2023-04-22'),

(21, 'Janaka', 'Dissanayake', 7, 'janaka.dissanayake@pos.lk', '0758901239', '2022-02-14'),

(22, 'Dilini', 'Ranawaka', 7, 'dilini.ranawaka@pos.lk', '0719012346', '2023-07-19'),

(23, 'Chaminda', 'Senarath', 7, 'chaminda.senarath@pos.lk', '0779123450', '2022-11-05'),

(24, 'Sanjeewa', 'Wickramaratne', 7, 'sanjeewa.wickramaratne@pos.lk', '0701234568', '2023-01-25'),

(25, 'Ruwanthi', 'Karunathilaka', 7, 'ruwanthi.karunathilaka@pos.lk', '0782345679', '2022-12-01'),

(26, 'Dulanjali', 'Madushanka', 7, 'dulanjali.madushanka@pos.lk', '0713456789', '2022-07-15'),

(27, 'Lakshman', 'Gunasekara', 7, 'lakshman.gunasekara@pos.lk', '0744567890', '2022-04-09'),

(28, 'Shanika', 'Wijerathne', 7, 'shanika.wijerathne@pos.lk', '0755678902', '2022-08-20'),

(29, 'Supun', 'Ranasinghe', 7, 'supun.ranasinghe@pos.lk', '0706789013', '2023-02-11'),

(30, 'Harindra', 'Wijesooriya', 7, 'harindra.wijesooriya@pos.lk', '0787890124', '2023-03-27'),

(31, 'Menaka', 'Jayasekara', 7, 'menaka.jayasekara@pos.lk', '0758901235', '2022-06-30'),

(32, 'Ishani', 'Wimalasuriya', 7, 'ishani.wimalasuriya@pos.lk', '0719012347', '2022-12-15'),

-- Customer Service Representatives

(33, 'Amali', 'Gunaratne', 8, 'amali.gunaratne@pos.lk', '0708901234', '2023-02-05'),

(34, 'Lakshan', 'Fernando', 8, 'lakshan.fernando@pos.lk', '0709012345', '2023-01-20'),

-- IT Support Specialists

(35, 'Sameera', 'Ranasinghe', 9, 'sameera.ranasinghe@pos.lk', '0746789012', '2023-10-09'),

-- Marketing Executives

(36, 'Ashani', 'Mendis', 10, 'ashani.mendis@pos.lk', '0704567890', '2023-08-01'),

-- Delivery Coordinators

(37, 'Dilini', 'Ratnayake', 11, 'dilini.ratnayake@pos.lk', '0780123456', '2023-02-25'),

(38, 'Tharuka', 'Wijesooriya', 11, 'tharuka.wijesooriya@pos.lk', '0756783456', '2023-03-15'),

(39, 'Mahesh', 'Senarath', 11, 'mahesh.senarath@pos.lk', '0719234567', '2023-04-10');

SET IDENTITY\_INSERT Employee OFF;

**Adding data to AuditLog table**

SET IDENTITY\_INSERT AuditLog ON;

INSERT INTO AuditLog (LogID, Action, PerformedBy, ActionTimestamp, TableName)

VALUES

-- Admin Actions

(1, 'Added new employee: Amali Gunaratne', 1, '2023-02-05 09:15:30', 'Employee'),

(2, 'Updated product price for Ceylon Tea 500g', 2, '2023-06-12 10:22:15', 'Product'),

(3, 'Deleted inactive customer record', 1, '2023-08-01 14:45:05', 'Customer'),

-- Cashier Actions

(4, 'Processed sales transaction #10123', 3, '2023-05-25 13:34:50', 'SalesTransaction'),

(5, 'Applied discount on sales transaction #10256', 4, '2023-06-18 15:25:10', 'SalesTransactionDetail'),

(6, 'Updated payment method for transaction #10321', 4, '2023-07-02 16:40:00', 'PaymentMethod'),

-- Manager Actions

(7, 'Approved bulk order request for Lanka Spices Ltd.', 5, '2023-09-10 11:10:45', 'Stock'),

(8, 'Assigned new warehouse manager to Warehouse #3', 6, '2023-10-20 10:00:15', 'Warehouse'),

(9, 'Updated stock reorder level for Rice 10kg', 5, '2023-11-11 09:45:30', 'Product'),

-- Inventory Supervisor Actions

(10, 'Stock restocked for Product #205 (Milk Powder)', 7, '2023-08-23 14:15:55', 'Stock'),

(11, 'Updated supplier details for Product #109 (Coconut Oil)', 8, '2023-09-30 12:10:25', 'Product'),

(12, 'Performed monthly stock audit for Warehouse #1', 7, '2023-12-01 11:00:00', 'AuditLog'),

-- Sales Representative Actions

(13, 'Generated sales report for July 2023', 9, '2023-07-31 17:45:30', 'Report'),

(14, 'Generated promotional sales summary', 10, '2023-09-12 18:00:00', 'Report'),

(15, 'Submitted customer feedback report', 9, '2023-10-18 15:20:15', 'Report'),

-- Accountants Actions

(16, 'Processed supplier invoice #20435', 11, '2023-09-05 10:05:40', 'AuditLog'),

(17, 'Generated tax report for Q2 2023', 12, '2023-07-01 12:45:55', 'Report'),

(18, 'Corrected payment record for Invoice #20987', 11, '2023-08-20 14:30:25', 'PaymentMethod'),

-- Warehouse Manager Actions

(19, 'Added new stock location: Warehouse #4', 13, '2023-04-01 10:20:00', 'Warehouse'),

(20, 'Updated manager details for Warehouse #2', 14, '2023-05-18 15:15:35', 'Warehouse'),

(21, 'Approved stock transfer to Warehouse #3', 15, '2023-06-22 09:50:30', 'Stock'),

-- Customer Service Actions

(22, 'Resolved customer complaint for Order #23145', 33, '2023-07-25 11:30:15', 'Customer'),

(23, 'Updated customer phone number for account #12345', 34, '2023-08-10 14:05:50', 'Customer'),

-- IT Support Actions

(24, 'Performed system backup for Q3 2023', 35, '2023-10-15 20:00:00', 'AuditLog'),

(25, 'Restored customer database from backup', 35, '2023-11-20 22:10:05', 'Customer');

SET IDENTITY\_INSERT AuditLog OFF;

**REPORT GENERATION MODULE**

**Adding data to Report table**

SET IDENTITY\_INSERT Report ON;

INSERT INTO Report (ReportID, ReportName, GeneratedBy, GeneratedDate, ReportType, ReportData)

VALUES

-- Daily Sales Reports

(1, 'Daily Sales Report - 2023-09-15', 9, '2023-09-15 20:10:00', 'Daily', '{TotalSales: 456700, Transactions: 158}'),

(2, 'Daily Sales Report - 2023-09-16', 10, '2023-09-16 20:10:00', 'Daily', '{TotalSales: 512300, Transactions: 172}'),

(3, 'Daily Sales Report - 2023-09-17', 9, '2023-09-17 20:10:00', 'Daily', '{TotalSales: 478200, Transactions: 164}'),

-- Monthly Sales Summary

(4, 'Monthly Sales Summary - August 2023', 12, '2023-09-01 09:15:00', 'Monthly', '{TotalSales: 14567000, TotalTransactions: 4520, TopProduct: "Ceylon Tea 500g"}'),

(5, 'Monthly Sales Summary - September 2023', 12, '2023-10-01 09:15:00', 'Monthly', '{TotalSales: 15784000, TotalTransactions: 4805, TopProduct: "Rice 10kg"}'),

-- Stock and Inventory Reports

(6, 'Low Stock Alert - September 2023', 7, '2023-09-20 14:30:00', 'Monthly', '{Products: ["Milk Powder", "Sugar", "Flour"]}'),

(7, 'Stock Reorder Summary - October 2023', 8, '2023-10-05 14:30:00', 'Monthly', '{Products: ["Rice 5kg", "Coconut Oil"]}'),

-- Customer Reports

(8, 'New Customer Registrations - September 2023', 33, '2023-09-30 16:00:00', 'Monthly', '{TotalCustomers: 325, NewCustomers: 115}'),

(9, 'Inactive Customers Report - Q3 2023', 34, '2023-10-05 16:30:00', 'Quarterly', '{InactiveCustomers: 45}'),

-- Financial Reports

(10, 'Profit & Loss Statement - August 2023', 11, '2023-09-01 12:45:00', 'Monthly', '{Revenue: 15784000, Expenses: 10230000, NetProfit: 5554000}'),

(11, 'Tax Summary Report - Q3 2023', 12, '2023-10-10 13:00:00', 'Quarterly', '{VAT: 2045000, OtherTaxes: 150000}'),

-- Employee Performance Reports

(12, 'Employee Performance - Sales Reps Q3 2023', 5, '2023-10-12 10:45:00', 'Quarterly', '{TopEmployee: "Kasun Perera", TotalSales: 740500}'),

(13, 'Employee Attendance - September 2023', 6, '2023-09-30 17:00:00', 'Monthly', '{TotalAbsences: 12, BestAttendance: "Tharindu Jayasinghe"}'),

-- Promotional Campaigns and Marketing

(14, 'Promotion Impact Report - Mid-Year Sale 2023', 10, '2023-08-10 15:15:00', 'Campaign', '{SalesBoost: "25%", TopSelling: "Ceylon Tea"}'),

(15, 'Festival Campaign Summary - Sinhala New Year 2023', 10, '2023-04-20 16:00:00', 'Campaign', '{TotalSales: 9500000, TopCategory: "Food & Beverages"}'),

-- Supplier and Delivery Performance Reports

(16, 'Supplier Performance - Q3 2023', 8, '2023-10-15 14:30:00', 'Quarterly', '{TopSupplier: "Lanka Spices Ltd.", DeliverySuccessRate: "98%"}'),

(17, 'Delivery Timeliness Report - September 2023', 11, '2023-09-30 16:00:00', 'Monthly', '{OnTimeDeliveries: 350, LateDeliveries: 15}'),

-- System Logs and Security

(18, 'System Audit Log Summary - Q3 2023', 35, '2023-10-01 12:00:00', 'Quarterly', '{CriticalActions: 18, SystemErrors: 5}'),

(19, 'System Backup Report - October 2023', 35, '2023-10-31 21:30:00', 'Monthly', '{BackupStatus: "Successful", BackupSize: "120GB"}'),

-- Business Development Reports

(20, 'Store Expansion Plan - 2024', 1, '2023-11-01 10:30:00', 'Planning', '{NewStores: ["Galle", "Kandy", "Jaffna"]}'),

(21, 'Market Trend Analysis - Q3 2023', 10, '2023-10-25 14:15:00', 'Analysis', '{Trend: "Organic Food Demand Increase"}'),

-- Custom Reports

(22, 'Top Selling Products Report - Q3 2023', 9, '2023-10-10 11:00:00', 'Custom', '{Products: ["Rice 10kg", "Milk Powder", "Ceylon Tea 500g"]}'),

(23, 'Year-End Sales Forecast - 2023', 1, '2023-11-30 10:00:00', 'Forecast', '{ExpectedSales: 62000000, KeySeasons: ["Christmas", "New Year"]}'),

(24, 'Supplier Cost Summary - October 2023', 11, '2023-11-01 13:15:00', 'Monthly', '{TotalCost: 8400000, KeySuppliers: ["Lanka Spices Ltd."]}'),

(25, 'Customer Loyalty Program Impact - 2023', 34, '2023-12-01 15:45:00', 'Analysis', '{TotalMembers: 1250, ActiveMembers: 850}');

SET IDENTITY\_INSERT Report OFF;

**PRODUCT AND STOCK MANAGEMENT MODULE**

**Adding data to Category table**

SET IDENTITY\_INSERT Category ON;

INSERT INTO Category (CategoryID, CategoryName)

VALUES

(1, 'Dairy'),

(2, 'Tea'),

(3, 'Staples'),

(4, 'Snacks'),

(5, 'Frozen Food'),

(6, 'Condiments'),

(7, 'Household'),

(8, 'Beverages'),

(9, 'Personal Care'),

(10, 'Baby Care'),

(11, 'Pet Supplies'),

(12, 'Health & Wellness'),

(13, 'Bakery'),

(14, 'Breakfast Items'),

(15, 'Canned Goods'),

(16, 'Fresh Produce'),

(17, 'Meat & Seafood'),

(18, 'Spices'),

(19, 'Cleaning Supplies'),

(20, 'Electronics');

SET IDENTITY\_INSERT Category OFF;

**Adding data to Brand table**

SET IDENTITY\_INSERT Brand ON;

INSERT INTO Brand (BrandID, BrandName)

VALUES

(1, 'Nestlé'),

(2, 'Dilmah'),

(3, 'Prima'),

(4, 'Mlesna'),

(5, 'Maliban'),

(6, 'Keells'),

(7, 'Anchor'),

(8, 'Kist'),

(9, 'Sunlight'),

(10, 'LUX'),

(11, 'CIC'),

(12, 'Harpic'),

(13, 'Lanka Soy'),

(14, 'Elephant House'),

(15, 'Coca-Cola'),

(16, 'Munchee'),

(17, 'Ritzbury'),

(18, 'Dettol'),

(19, 'Kotmale'),

(20, 'Fonterra'),

(21, 'Elephant House Ice Cream'),

(22, 'Hela Bōjun'),

(23, 'Perera & Sons'),

(24, 'Hemas'),

(25, 'Pears'),

(26, 'Gold Leaf'),

(27, 'MD'),

(28, 'Maggi'),

(29, 'Samaposha'),

(30, 'Araliya'),

(31, 'Renuka'),

(32, 'Malwatta'),

(33, 'Ruhunu Foods'),

(34, 'Singer'),

(35, 'Milo'),

(36, 'Highland'),

(37, 'Harischandra'),

(38, 'Seven Seas'),

(39, 'Unilever');

SET IDENTITY\_INSERT Brand OFF;

**Adding data to the TaxRate table**

SET IDENTITY\_INSERT TaxRate ON;

INSERT INTO TaxRate (TaxRateID, TaxRate)

VALUES

(1, 8.0),

(2, 12.0),

(3, 15.0),

(4, 5.0),

(5, 18.0),

(6, 22.0),

(7, 10.0),

(8, 20.0),

(9, 25.0),

(10, 30.0),

(11, 0.0),

(12, 7.5),

(13, 14.5),

(14, 17.0),

(15, 6.0),

(16, 9.0),

(17, 11.0),

(18, 19.0),

(19, 16.0),

(20, 13.0);

SET IDENTITY\_INSERT TaxRate OFF;

**Adding data to Product table**

SET IDENTITY\_INSERT Product ON;

INSERT INTO Product (ProductID, ProductName, CategoryID, BrandID, UnitPrice, TaxRateID, ReorderLevel, IsActive)

VALUES

-- Dairy Products

(1, 'Anchor Full Cream Milk Powder', 1, 7, 1000.00, 3, 50, 1),

(2, 'Kotmale Fresh Milk 1L', 1, 19, 240.00, 1, 30, 1),

(3, 'Highland Butter 200g', 1, 37, 950.00, 3, 20, 1),

-- Tea

(4, 'Dilmah Premium Tea 200g', 2, 2, 750.00, 2, 40, 1),

(5, 'Mlesna Black Tea 100g', 2, 4, 600.00, 2, 25, 1),

(6, 'Gold Leaf Tea 400g', 2, 26, 850.00, 3, 50, 1),

-- Staples

(7, 'Prima Wheat Flour 1kg', 3, 3, 180.00, 1, 100, 1),

(8, 'Ruhunu Red Rice 5kg', 3, 34, 1500.00, 2, 50, 1),

(9, 'CIC Basmati Rice 1kg', 3, 11, 1100.00, 2, 30, 1),

-- Snacks

(10, 'Munchee Marie Biscuits 400g', 4, 16, 240.00, 2, 70, 1),

(11, 'Ritzbury Chocolate Fingers 200g', 4, 17, 380.00, 3, 40, 1),

(12, 'Maliban Lemon Puff 300g', 4, 5, 300.00, 2, 60, 1),

-- Frozen Food

(13, 'Elephant House Chicken Sausages 1kg', 5, 14, 1500.00, 3, 20, 1),

(14, 'Keells Chicken Drumsticks 1kg', 5, 6, 1200.00, 3, 25, 1),

(15, 'Kotmale Ice Cream Vanilla 1L', 5, 19, 800.00, 3, 15, 1),

-- Condiments

(16, 'MD Mango Chutney 250g', 6, 27, 500.00, 2, 40, 1),

(17, 'Kist Tomato Sauce 750ml', 6, 8, 450.00, 2, 50, 1),

(18, 'Maggi Coconut Milk Powder 300g', 6, 28, 600.00, 2, 35, 1),

-- Household

(19, 'Sunlight Detergent Powder 2kg', 7, 9, 1500.00, 3, 20, 1),

(20, 'Harpic Toilet Cleaner 500ml', 7, 12, 550.00, 2, 25, 1),

(21, 'Dettol Antibacterial Soap 100g', 7, 18, 150.00, 2, 60, 1),

-- Beverages

(22, 'Coca-Cola Bottle 1.5L', 8, 15, 300.00, 2, 100, 1),

(23, 'Elephant House Cream Soda Can 330ml', 8, 14, 120.00, 2, 150, 1),

(24, 'Nestlé Milo 400g', 8, 35, 800.00, 3, 30, 1),

-- Personal Care

(25, 'LUX Body Wash 250ml', 9, 10, 450.00, 2, 20, 1),

(26, 'Pears Baby Lotion 200ml', 9, 25, 550.00, 2, 15, 1),

(27, 'Hemas Velvet Soap 100g', 9, 24, 100.00, 1, 50, 1);

SET IDENTITY\_INSERT Product OFF;

**Adding data to Warehouse table**

SET IDENTITY\_INSERT Warehouse ON;

INSERT INTO Warehouse (WarehouseID, WarehouseName, Location, ManagerID)

VALUES

(1, 'Central Warehouse', 'Colombo', 13),

(2, 'Kandy Distribution Center', 'Kandy', 14),

(3, 'Southern Storage', 'Galle', 15),

(4, 'Northern Depot', 'Jaffna', 16),

(5, 'Eastern Hub', 'Trincomalee', 17),

(6, 'Western Storage', 'Negombo', 18),

(7, 'Hill Country Depot', 'Nuwara Eliya', 19),

(8, 'North Western Facility', 'Kurunegala', 20),

(9, 'Uva Warehouse', 'Badulla', 21),

(10, 'Sabaragamuwa Depot', 'Ratnapura', 22),

(11, 'Eastern Highlands Center', 'Ampara', 23),

(12, 'Coastal Hub', 'Matara', 24),

(13, 'Dry Zone Depot', 'Anuradhapura', 25),

(14, 'Urban Storage', 'Batticaloa', 26),

(15, 'Central Highlands Depot', 'Hatton', 27),

(16, 'Industrial Hub', 'Kalutara', 28),

(17, 'Administrative Warehouse', 'Chilaw', 29),

(18, 'Port Storage', 'Hambantota', 30),

(19, 'Heritage Hub', 'Polonnaruwa', 31),

(20, 'Frontier Depot', 'Monaragala', 32);

SET IDENTITY\_INSERT Warehouse OFF;

**Adding data to Stock table**

SET IDENTITY\_INSERT Stock ON;

INSERT INTO Stock (StockID, ProductID, WarehouseID, QuantityAvailable, LastRestockedDate)

VALUES

(1, 1, 1, 500, '2024-12-01'),

(2, 2, 2, 300, '2024-12-02'),

(3, 3, 3, 200, '2024-12-03'),

(4, 4, 4, 600, '2024-12-04'),

(5, 5, 5, 450, '2024-12-05'),

(6, 6, 6, 700, '2024-12-06'),

(7, 7, 7, 800, '2024-12-01'),

(8, 8, 8, 400, '2024-12-02'),

(9, 9, 9, 350, '2024-12-03'),

(10, 10, 10, 500, '2024-12-04'),

(11, 11, 11, 550, '2024-12-05'),

(12, 12, 12, 250, '2024-12-06'),

(13, 13, 13, 600, '2024-12-01'),

(14, 14, 14, 300, '2024-12-02'),

(15, 15, 15, 450, '2024-12-03'),

(16, 16, 16, 700, '2024-12-04'),

(17, 17, 17, 500, '2024-12-05'),

(18, 18, 18, 350, '2024-12-06'),

(19, 19, 19, 400, '2024-12-01'),

(20, 20, 20, 600, '2024-12-02');

SET IDENTITY\_INSERT Stock OFF;

**DISCOUNT MANAGEMENT MODULE**

**Adding data to Discount table**

SET IDENTITY\_INSERT Discount ON;

INSERT INTO Discount (DiscountID, DiscountName, StartDate, EndDate, DiscountPercentage, DiscountType)

VALUES

(1, 'Avurudu Sale 2024', '2024-04-01', '2024-04-14', 15.00, 'Seasonal'),

(2, 'Vesak Poya Special', '2024-05-21', '2024-05-25', 10.00, 'Religious'),

(3, 'Sinhala Tamil New Year Bonanza', '2024-04-10', '2024-04-20', 20.00, 'Seasonal'),

(4, 'Back to School Offer', '2024-01-05', '2024-01-15', 12.50, 'Event'),

(5, 'Weekend Super Saver', '2024-06-01', '2024-06-02', 5.00, 'Weekend'),

(6, 'Independence Day Promo', '2024-02-01', '2024-02-07', 10.00, 'National Holiday'),

(7, 'Ramadan Festival Discount', '2024-03-11', '2024-03-30', 8.00, 'Religious'),

(8, 'Christmas Mega Sale', '2024-12-15', '2024-12-25', 25.00, 'Seasonal'),

(9, 'Black Friday Deals', '2024-11-29', '2024-11-30', 30.00, 'Event'),

(10, 'Poya Day Essentials Discount', '2024-06-21', '2024-06-21', 5.00, 'Religious'),

(11, 'Maha Shivaratri Offer', '2024-02-20', '2024-02-21', 12.00, 'Religious'),

(12, 'Mother’s Day Special', '2024-05-11', '2024-05-12', 15.00, 'Event'),

(13, 'Father’s Day Promotion', '2024-06-15', '2024-06-16', 15.00, 'Event'),

(14, 'Govi Sathiya Discounts', '2024-09-01', '2024-09-07', 10.00, 'Agricultural'),

(15, 'Back to Office Sale', '2024-07-01', '2024-07-10', 7.50, 'Event'),

(16, 'Deepavali Celebrations', '2024-10-15', '2024-10-25', 12.00, 'Religious'),

(17, 'End of Season Clearance', '2024-08-25', '2024-08-31', 20.00, 'Clearance'),

(18, 'School Holiday Snacks Promo', '2024-08-01', '2024-08-10', 5.00, 'Event'),

(19, 'Kandy Esala Perahera Special', '2024-07-20', '2024-07-30', 8.50, 'Cultural'),

(20, 'Poson Poya Discount', '2024-06-20', '2024-06-22', 10.00, 'Religious'),

(21, 'Pre-Avurudu Stock Clearance', '2024-03-25', '2024-03-31', 18.00, 'Clearance'),

(22, 'Valentine’s Day Promo', '2024-02-12', '2024-02-14', 20.00, 'Event'),

(23, 'New Year Kickoff Sale', '2024-01-01', '2024-01-05', 10.00, 'Event'),

(24, 'Thrift Thursday Offer', '2024-06-13', '2024-06-13', 5.00, 'Weekly'),

(25, 'Budget Sunday Promo', '2024-06-16', '2024-06-16', 6.00, 'Weekly');

SET IDENTITY\_INSERT Discount OFF;

**Adding data to Product Discount table**

SET IDENTITY\_INSERT ProductDiscount ON;

INSERT INTO ProductDiscount (ProductDiscountID, ProductID, DiscountID)

VALUES

-- Avurudu Sale 2024

(1, 1, 1), -- Anchor Full Cream Milk Powder

(2, 8, 1), -- Ruhunu Red Rice 5kg

(3, 12, 1), -- Maliban Lemon Puff 300g

(4, 7, 1), -- Prima Wheat Flour 1kg

-- Vesak Poya Special

(5, 2, 2), -- Kotmale Fresh Milk 1L

(6, 21, 2), -- Dettol Antibacterial Soap 100g

-- Back to School Offer

(7, 10, 4), -- Munchee Marie Biscuits 400g

(8, 11, 4), -- Ritzbury Chocolate Fingers 200g

(9, 26, 4), -- Pears Baby Lotion 200ml

-- Independence Day Promo

(10, 4, 6), -- Dilmah Premium Tea 200g

(11, 24, 6), -- Nestlé Milo 400g

-- Ramadan Festival Discount

(12, 9, 7), -- CIC Basmati Rice 1kg

(13, 18, 7), -- Maggi Coconut Milk Powder 300g

-- Christmas Mega Sale

(14, 13, 8), -- Elephant House Chicken Sausages 1kg

(15, 14, 8), -- Keells Chicken Drumsticks 1kg

(16, 3, 8), -- Highland Butter 200g

(17, 27, 8), -- Hemas Velvet Soap 100g

-- Black Friday Deals

(18, 5, 9), -- Mlesna Black Tea 100g

(19, 6, 9), -- Gold Leaf Tea 400g

(20, 20, 9), -- Harpic Toilet Cleaner 500ml

-- Poya Day Essentials Discount

(21, 15, 10), -- Kotmale Ice Cream Vanilla 1L

(22, 19, 10), -- Sunlight Detergent Powder 2kg

-- End of Season Clearance

(23, 22, 17), -- Coca-Cola Bottle 1.5L

(24, 23, 17), -- Elephant House Cream Soda Can 330ml

(25, 17, 17); -- Kist Tomato Sauce 750ml

SET IDENTITY\_INSERT ProductDiscount OFF;

**SALES MANAGEMENT MODULE**

**Adding data to Payment Method table**

SET IDENTITY\_INSERT PaymentMethod ON;

INSERT INTO PaymentMethod (PaymentMethodID, PaymentType)

VALUES

(1, 'Cash'),

(2, 'Debit Card - Commercial Bank'),

(3, 'Debit Card - HNB'),

(4, 'Credit Card - BOC'),

(5, 'Credit Card - Sampath Bank'),

(6, 'Mobile Payment - FriMi'),

(7, 'Mobile Payment - eZ Cash'),

(8, 'Mobile Payment - mCash'),

(9, 'Bank Transfer - NDB'),

(10, 'Bank Transfer - NSB Bank'),

(11, 'QR Payment - LankaQR');

SET IDENTITY\_INSERT PaymentMethod OFF;

**Adding data to Sales Transaction table**

SET IDENTITY\_INSERT SalesTransaction ON;

INSERT INTO SalesTransaction (TransactionID, CustomerID, TransactionDate, PaymentMethodID, TotalAmount, DiscountApplied, FinalAmount)

VALUES

(1, 1, '2024-01-01 10:15:00', 1, 1500.00, 100.00, 1400.00),

(2, 2, '2024-01-02 15:45:00', 2, 2500.00, 150.00, 2350.00),

(3, 3, '2024-01-03 12:30:00', 3, 4500.00, 200.00, 4300.00),

(4, 4, '2024-01-04 17:50:00', 6, 3200.00, 100.00, 3100.00),

(5, 5, '2024-01-05 09:40:00', 7, 1800.00, 50.00, 1750.00),

(6, 6, '2024-01-06 14:20:00', 4, 3700.00, 200.00, 3500.00),

(7, 7, '2024-01-07 11:00:00', 5, 5000.00, 300.00, 4700.00),

(8, 8, '2024-01-08 16:10:00', 8, 2200.00, 0.00, 2200.00),

(9, 9, '2024-01-09 13:25:00', 9, 6000.00, 500.00, 5500.00),

(10, 10, '2024-01-10 10:00:00', 10, 1200.00, 0.00, 1200.00),

(11, 11, '2024-01-11 18:15:00', 11, 3400.00, 150.00, 3250.00),

(12, 12, '2024-01-12 09:20:00', 1, 2800.00, 100.00, 2700.00),

(13, 13, '2024-01-13 14:35:00', 2, 1900.00, 50.00, 1850.00),

(14, 14, '2024-01-14 11:45:00', 3, 3100.00, 100.00, 3000.00),

(15, 15, '2024-01-15 16:00:00', 6, 4700.00, 200.00, 4500.00),

(16, 16, '2024-01-16 10:50:00', 7, 2100.00, 50.00, 2050.00),

(17, 17, '2024-01-17 13:10:00', 5, 4300.00, 300.00, 4000.00),

(18, 18, '2024-01-18 15:25:00', 4, 3800.00, 150.00, 3650.00),

(19, 19, '2024-01-19 17:00:00', 9, 5200.00, 300.00, 4900.00),

(20, 20, '2024-01-20 12:00:00', 10, 2500.00, 100.00, 2400.00);

SET IDENTITY\_INSERT SalesTransaction OFF;

**Adding data to Sales Transaction Detail table**

SET IDENTITY\_INSERT SalesTransactionDetail ON;

INSERT INTO SalesTransactionDetail (TransactionDetailID, TransactionID, ProductID, Quantity, UnitPrice, LineTotal)

VALUES

-- Transaction 1

(1, 1, 7, 3, 180.00, 540.00),

(2, 1, 10, 4, 240.00, 960.00),

-- Transaction 2

(3, 2, 2, 5, 240.00, 1200.00),

(4, 2, 21, 2, 150.00, 300.00),

(5, 2, 6, 1, 850.00, 850.00),

-- Transaction 3

(6, 3, 19, 2, 1500.00, 3000.00),

(7, 3, 12, 2, 300.00, 600.00),

(8, 3, 3, 1, 950.00, 950.00),

-- Transaction 4

(9, 4, 5, 2, 600.00, 1200.00),

(10, 4, 8, 1, 1500.00, 1500.00),

(11, 4, 25, 1, 450.00, 450.00),

-- Transaction 5

(12, 5, 1, 1, 1000.00, 1000.00),

(13, 5, 24, 1, 800.00, 800.00),

-- Transaction 6

(14, 6, 15, 2, 800.00, 1600.00),

(15, 6, 17, 2, 450.00, 900.00),

(16, 6, 13, 1, 1500.00, 1500.00),

-- Transaction 7

(17, 7, 11, 3, 380.00, 1140.00),

(18, 7, 9, 2, 1100.00, 2200.00),

(19, 7, 20, 1, 550.00, 550.00),

-- Transaction 8

(20, 8, 14, 2, 1200.00, 2400.00),

-- Transaction 9

(21, 9, 4, 2, 750.00, 1500.00),

(22, 9, 18, 3, 600.00, 1800.00),

(23, 9, 22, 2, 300.00, 600.00),

-- Transaction 10

(24, 10, 16, 1, 500.00, 500.00),

(25, 10, 27, 7, 100.00, 700.00);

SET IDENTITY\_INSERT SalesTransactionDetail OFF;

## Function And Views

**Functions**

-- Function Name: dbo.fn\_CalculateTotalSales

-- Description: Calculates the total amount spent by a customer on all transactions

-- Inputs:

-- @CustomerID (INT) - The ID of the customer

-- Returns:

-- DECIMAL - The total amount spent by the customer

CREATE FUNCTION dbo.fn\_CalculateTotalSales (@CustomerID INT)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @TotalSales DECIMAL(10, 2);

-- Calculate the total sales for the customer by summing the FinalAmount from SalesTransaction

SELECT @TotalSales = SUM(T.FinalAmount)

FROM SalesTransaction T

WHERE T.CustomerID = @CustomerID;

-- Return the result

RETURN ISNULL(@TotalSales, 0); -- Return 0 if no sales are found

END

-- Function Name: dbo.fn\_CalculateProductSales

-- Description: Calculates the total sales for a given product

-- Inputs:

-- @ProductID (INT) - The ID of the product

-- Returns:

-- DECIMAL - The total sales for the product

CREATE FUNCTION dbo.fn\_CalculateProductSales (@ProductID INT)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @TotalSales DECIMAL(10, 2);

-- Calculate the total sales for the product by summing the LineTotal from SalesTransactionDetail

SELECT @TotalSales = SUM(D.LineTotal)

FROM SalesTransactionDetail D

WHERE D.ProductID = @ProductID;

-- Return the result

RETURN ISNULL(@TotalSales, 0); -- Return 0 if no sales are found

END

-- Function Name: dbo.fn\_GetProductAvailability

-- Description: Returns the available quantity of a product in a specific warehouse

-- Inputs:

-- @ProductID (INT) - The ID of the product

-- @WarehouseID (INT) - The ID of the warehouse

-- Returns:

-- INT - The available quantity of the product in the warehouse

CREATE FUNCTION dbo.fn\_GetProductAvailability (@ProductID INT, @WarehouseID INT)

RETURNS INT

AS

BEGIN

DECLARE @QuantityAvailable INT;

-- Get the quantity of the product in the specified warehouse

SELECT @QuantityAvailable = QuantityAvailable

FROM Stock

WHERE ProductID = @ProductID AND WarehouseID = @WarehouseID;

-- Return the result

RETURN ISNULL(@QuantityAvailable, 0); -- Return 0 if the product is not found

END

-- Function Name: dbo.fn\_GetDiscountedPrice

-- Description: Calculates the price of a product after applying a discount

-- Inputs:

-- @ProductID (INT) - The ID of the product

-- @DiscountID (INT) - The ID of the discount to be applied

-- Returns:

-- DECIMAL - The discounted price of the product

CREATE FUNCTION dbo.fn\_GetDiscountedPrice (@ProductID INT, @DiscountID INT)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @DiscountPercentage DECIMAL(5, 2);

DECLARE @UnitPrice DECIMAL(10, 2);

DECLARE @DiscountedPrice DECIMAL(10, 2);

-- Get the unit price of the product

SELECT @UnitPrice = UnitPrice

FROM Product

WHERE ProductID = @ProductID;

-- Get the discount percentage for the given discount ID

SELECT @DiscountPercentage = DiscountPercentage

FROM Discount

WHERE DiscountID = @DiscountID;

-- Calculate the discounted price

SET @DiscountedPrice = @UnitPrice - (@UnitPrice \* @DiscountPercentage / 100);

-- Return the discounted price

RETURN @DiscountedPrice;

END

**Views**

**Customer related views**

-- View Name: vw\_CustomerSalesSummary

-- Description: Summary view of total sales and transaction count for each customer

CREATE VIEW vw\_CustomerSalesSummary AS

SELECT

C.CustomerID,

C.FirstName + ' ' + C.LastName AS CustomerName,

COUNT(T.TransactionID) AS TotalTransactions,

SUM(T.FinalAmount) AS TotalSpent,

MAX(T.TransactionDate) AS LastTransactionDate

FROM

Customer C

LEFT JOIN SalesTransaction T ON C.CustomerID = T.CustomerID

GROUP BY

C.CustomerID, C.FirstName, C.LastName;

**Product related views**

-- View Name: vw\_ProductSalesOverview

-- Description: Provides an overview of products with total sales and current stock

CREATE VIEW vw\_ProductSalesOverview AS

SELECT

P.ProductID,

P.ProductName,

ISNULL(SUM(D.LineTotal), 0) AS TotalSales,

ISNULL(S.QuantityAvailable, 0) AS StockAvailable

FROM

Product P

LEFT JOIN SalesTransactionDetail D ON P.ProductID = D.ProductID

LEFT JOIN Stock S ON P.ProductID = S.ProductID

GROUP BY

P.ProductID, P.ProductName, S.QuantityAvailable;

**Employee related views**

-- View Name: vw\_EmployeeRoleSummary

-- Description: Provides employee details along with their roles and contact info

CREATE VIEW vw\_EmployeeRoleSummary AS

SELECT

E.EmployeeID,

E.FirstName + ' ' + E.LastName AS EmployeeName,

R.RoleName,

E.Email,

E.PhoneNumber,

E.HireDate

FROM

Employee E

JOIN Role R ON E.RoleID = R.RoleID;

**Sales and Transaction**

-- View Name: vw\_SalesTransactionDetails

-- Description: Provides details of each sales transaction, including product info

CREATE VIEW vw\_SalesTransactionDetails AS

SELECT

T.TransactionID,

C.FirstName + ' ' + C.LastName AS CustomerName,

P.ProductName,

D.Quantity,

D.UnitPrice,

D.LineTotal,

T.TransactionDate

FROM

SalesTransaction T

JOIN Customer C ON T.CustomerID = C.CustomerID

JOIN SalesTransactionDetail D ON T.TransactionID = D.TransactionID

JOIN Product P ON D.ProductID = P.ProductID;

## Procedures & Triggers

**Procedure**

**Customer management procedure**

-- Procedure Name: sp\_InsertCustomer

-- Description: Inserts a new customer into the Customer table and logs the action in AuditLog

-- Inputs:

-- @FirstName (NVARCHAR) - First name of the customer

-- @LastName (NVARCHAR) - Last name of the customer

-- @Email (NVARCHAR) - Email of the customer (must be unique)

-- @PhoneNumber (NVARCHAR) - Phone number of the customer (optional)

-- @AddressLine1, @AddressLine2, @City, @PostalCode - Customer address details (optional)

CREATE PROCEDURE sp\_InsertCustomer

@FirstName NVARCHAR(100),

@LastName NVARCHAR(100),

@Email NVARCHAR(255),

@PhoneNumber NVARCHAR(15) = NULL,

@AddressLine1 NVARCHAR(255) = NULL,

@AddressLine2 NVARCHAR(255) = NULL,

@City NVARCHAR(100) = NULL,

@PostalCode NVARCHAR(20) = NULL

AS

BEGIN

-- Begin error handling block

BEGIN TRY

-- Insert new customer into the Customer table

INSERT INTO Customer

(FirstName, LastName, Email, PhoneNumber, AddressLine1, AddressLine2, City, PostalCode, RegistrationDate)

VALUES

(@FirstName, @LastName, @Email, @PhoneNumber, @AddressLine1, @AddressLine2, @City, @PostalCode, GETDATE());

-- Get the CustomerID of the newly inserted record

DECLARE @CustomerID INT = SCOPE\_IDENTITY();

-- Log the action in the AuditLog

INSERT INTO AuditLog (Action, PerformedBy, ActionTimestamp, TableName)

VALUES ('Insert New Customer', 1, GETDATE(), 'Customer'); -- Assuming PerformedBy = 1 for now

-- Print success message

PRINT 'Customer inserted successfully with CustomerID = ' + CAST(@CustomerID AS NVARCHAR(20));

END TRY

BEGIN CATCH

-- Print error message

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH

END

**Sales Management Procedures**

-- Procedure Name: sp\_GenerateSalesReport

-- Description: Generate a sales report for a given date range

-- Inputs:

-- @StartDate (DATE) - The start date of the report

-- @EndDate (DATE) - The end date of the report

CREATE PROCEDURE sp\_GenerateSalesReport

@StartDate DATE,

@EndDate DATE

AS

BEGIN

-- Begin error handling block

BEGIN TRY

-- Select all transactions that happened in the date range

SELECT

T.TransactionID,

C.FirstName + ' ' + C.LastName AS CustomerName,

T.TransactionDate,

P.ProductName,

D.Quantity,

D.LineTotal

FROM

SalesTransaction T

JOIN Customer C ON T.CustomerID = C.CustomerID

JOIN SalesTransactionDetail D ON T.TransactionID = D.TransactionID

JOIN Product P ON D.ProductID = P.ProductID

WHERE

T.TransactionDate BETWEEN @StartDate AND @EndDate;

-- Log the report generation

INSERT INTO AuditLog (Action, PerformedBy, ActionTimestamp, TableName)

VALUES ('Generate Sales Report', 1, GETDATE(), 'SalesTransaction');

END TRY

BEGIN CATCH

-- Print error message

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH

END

**Assign Product Discount**

-- Procedure Name: sp\_AssignProductDiscount

-- Description: Assigns a discount to a product and ensures no duplicates

-- Inputs:

-- @ProductID (INT) - The ID of the product to which the discount will be assigned

-- @DiscountID (INT) - The ID of the discount to be applied

CREATE PROCEDURE sp\_AssignProductDiscount

@ProductID INT,

@DiscountID INT

AS

BEGIN

-- Begin error handling block

BEGIN TRY

-- Check if this product-discount combination already exists

IF NOT EXISTS (

SELECT 1

FROM ProductDiscount

WHERE ProductID = @ProductID AND DiscountID = @DiscountID

)

BEGIN

-- Insert a new product-discount association

INSERT INTO ProductDiscount (ProductID, DiscountID)

VALUES (@ProductID, @DiscountID);

-- Print success message

PRINT 'Product assigned to discount successfully.';

END

ELSE

BEGIN

-- Notify if the product-discount combo already exists

PRINT 'Product is already assigned to this discount.';

END

END TRY

BEGIN CATCH

-- Print error message

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH

END

**Triggers**

**Stock management triggers**

-- Trigger Name: trg\_UpdateStockAfterSale

-- Description: After a new sale, reduce the stock of the sold product

-- Trigger Fires: AFTER INSERT on SalesTransactionDetail

CREATE TRIGGER trg\_UpdateStockAfterSale

ON SalesTransactionDetail

AFTER INSERT

AS

BEGIN

-- Turn off additional result sets for the trigger

SET NOCOUNT ON;

-- Reduce the stock quantity based on the product and quantity sold

UPDATE S

SET S.QuantityAvailable = S.QuantityAvailable - I.Quantity

FROM Stock S

INNER JOIN INSERTED I ON S.ProductID = I.ProductID

WHERE S.WarehouseID = 1; -- Adjust WarehouseID as required

-- Log the stock update action

INSERT INTO AuditLog (Action, PerformedBy, ActionTimestamp, TableName)

VALUES ('Stock Updated After Sale', 1, GETDATE(), 'Stock');

END

-- Trigger Name: trg\_ReorderNotification

-- Description: Triggers when stock is updated, logs an alert if quantity falls below reorder level

-- Trigger Fires: AFTER UPDATE on Stock

CREATE TRIGGER trg\_ReorderNotification

ON Stock

AFTER UPDATE

AS

BEGIN

-- Turn off additional result sets for the trigger

SET NOCOUNT ON;

-- Insert an alert into the AuditLog if stock is below reorder level

INSERT INTO AuditLog (Action, PerformedBy, ActionTimestamp, TableName)

SELECT

'Reorder Alert: Stock Low for ProductID ' + CAST(I.ProductID AS NVARCHAR(10)),

1,

GETDATE(),

'Stock'

FROM INSERTED I

JOIN Product P ON I.ProductID = P.ProductID

WHERE I.QuantityAvailable < P.ReorderLevel;

END

**Customer related triggers**

-- Trigger Name: trg\_LogCustomerDeletion

-- Description: Logs the deletion of a customer in the AuditLog

-- Trigger Fires: AFTER DELETE on Customer

CREATE TRIGGER trg\_LogCustomerDeletion

ON Customer

AFTER DELETE

AS

BEGIN

-- Turn off additional result sets for the trigger

SET NOCOUNT ON;

-- Insert log into AuditLog table

INSERT INTO AuditLog (Action, PerformedBy, ActionTimestamp, TableName)

SELECT

'Deleted Customer: ' + D.FirstName + ' ' + D.LastName + ' (Email: ' + D.Email + ')',

1, -- Assuming PerformedBy = 1 (can be adjusted based on user context)

GETDATE(),

'Customer'

FROM DELETED D;

END

## project links

**GITHUB REPOSITORY**

Consists Of :

* + Final Coursework Deliverables
    - Report
    - Database Backup
    - Presentation
  + Report Sections
  + SQL Code Sections
  + Database Screenshots
  + Application Screenshots
  + Windows Application [C#]

**Link :** [**https://github.com/DilshanIBY/Supermarket-POS-System**](https://github.com/DilshanIBY/Supermarket-POS-System)

**GITHUB PROJECT**

Consists Of :

* Task Assignees
* Sprint Workloads
* Tasks Completion states
* Product Backlog
* Teams
* Gantt Chat

**Link :** [**https://github.com/users/DilshanIBY/projects/6/views/1**](https://github.com/users/DilshanIBY/projects/6/views/1)

**CANVA PRESENTATION**

Consists Of Each Member’s :

* Workload
* Contact details

**Link :** [**https://www.canva.com/design/DAGaRxZBWyc/04gwCSZ7jRxjjoAvZ\_7rVg/edit**](https://www.canva.com/design/DAGaRxZBWyc/04gwCSZ7jRxjjoAvZ_7rVg/edit)

# pos system screenshots

## applications

### A logo for a company Description automatically generated12.1.1 overview.

A screenshot of a graph

Description automatically generatedA person and person sitting at a table

Description automatically generated

### 12.1.2. customer management

### A screenshot of a product management Description automatically generated12.1.3. product management

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

### A screenshot of a computer Description automatically generated12.1.4. stock management

A screenshot of a computer

Description automatically generated

### A screenshot of a computer Description automatically generated12.1.5. sales management

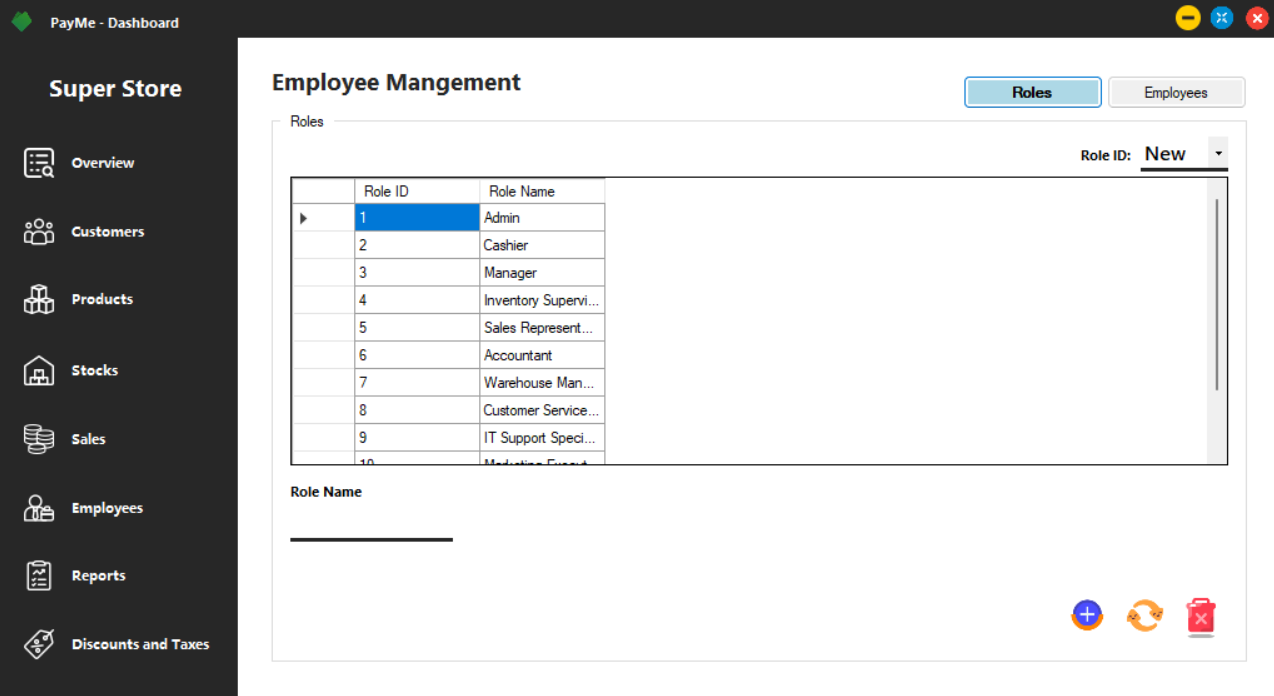
A screenshot of a computer

Description automatically generated

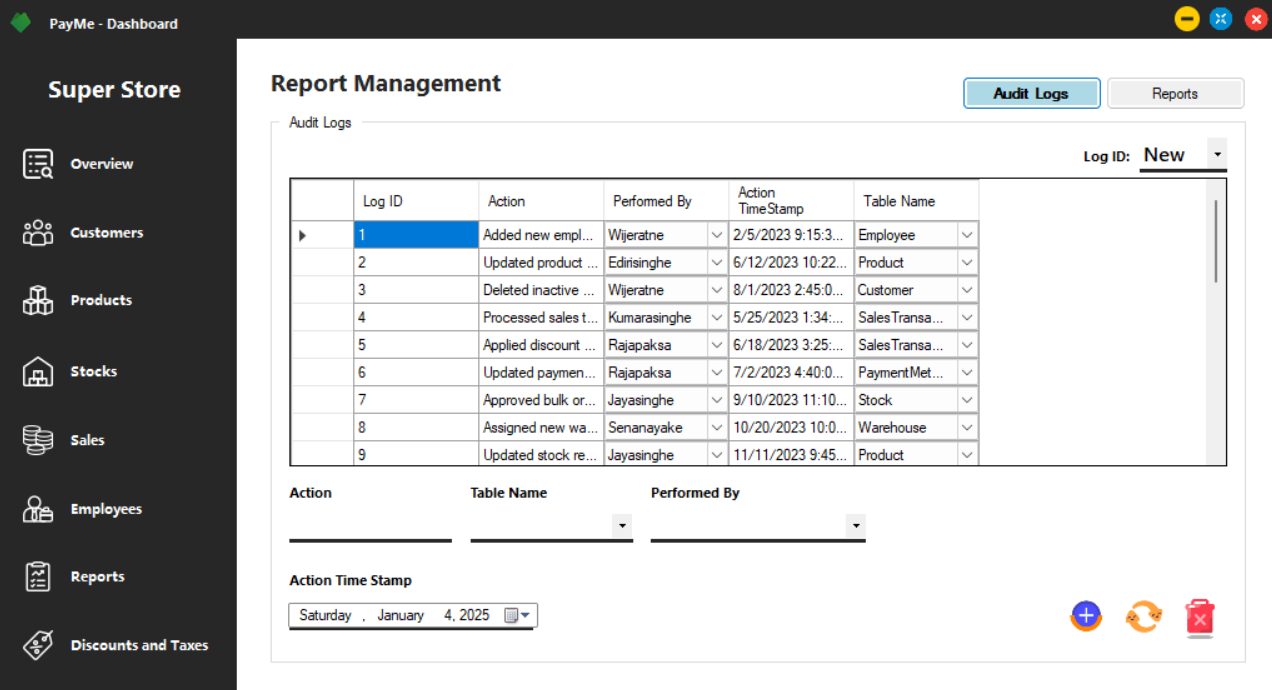
A screenshot of a computer

Description automatically generated

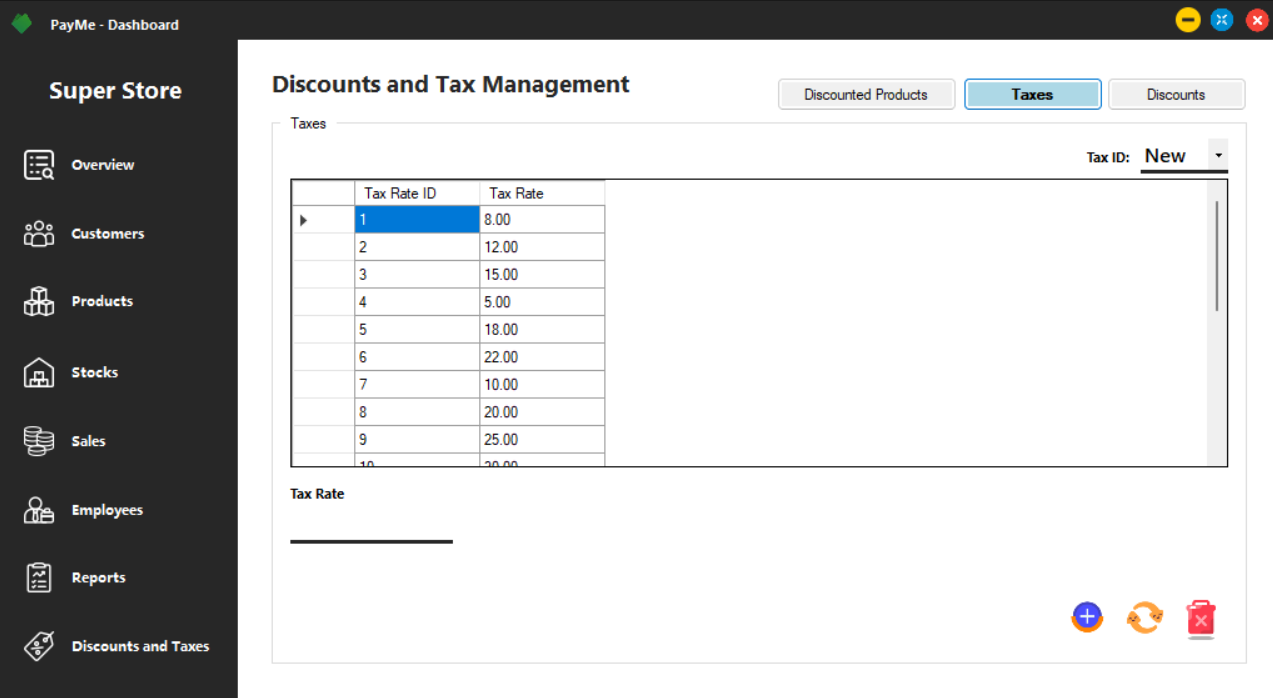
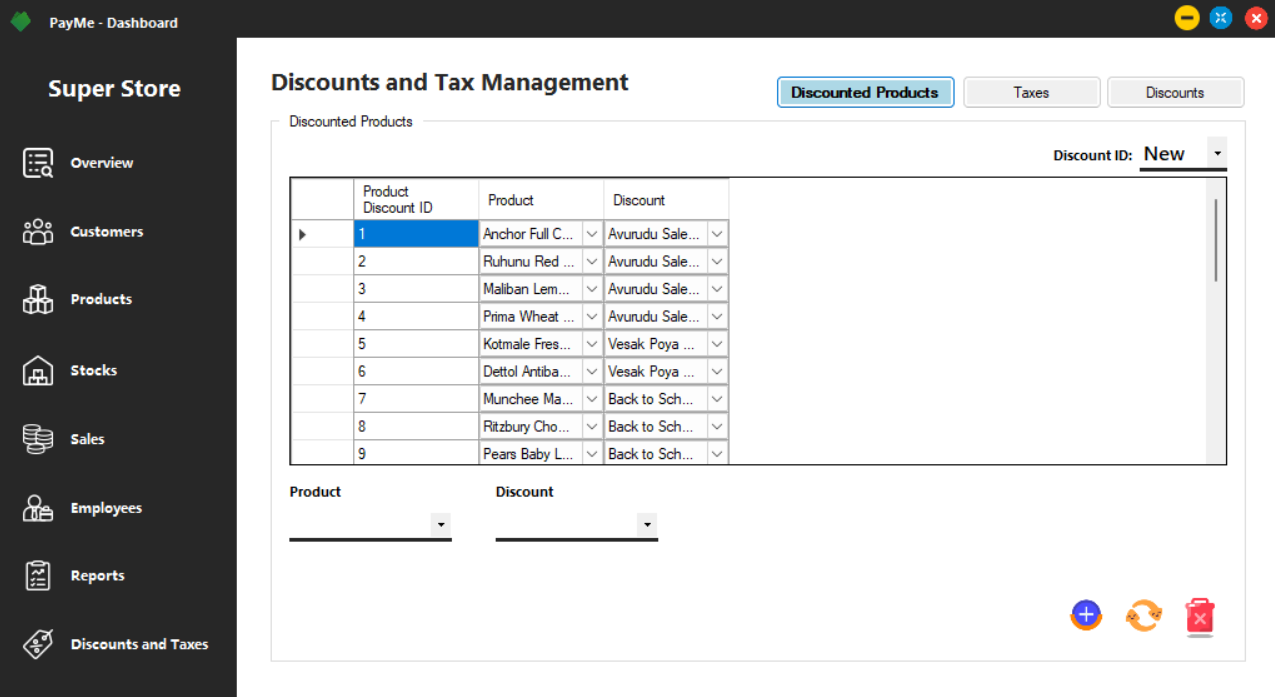
### 12.1.6. Employee management



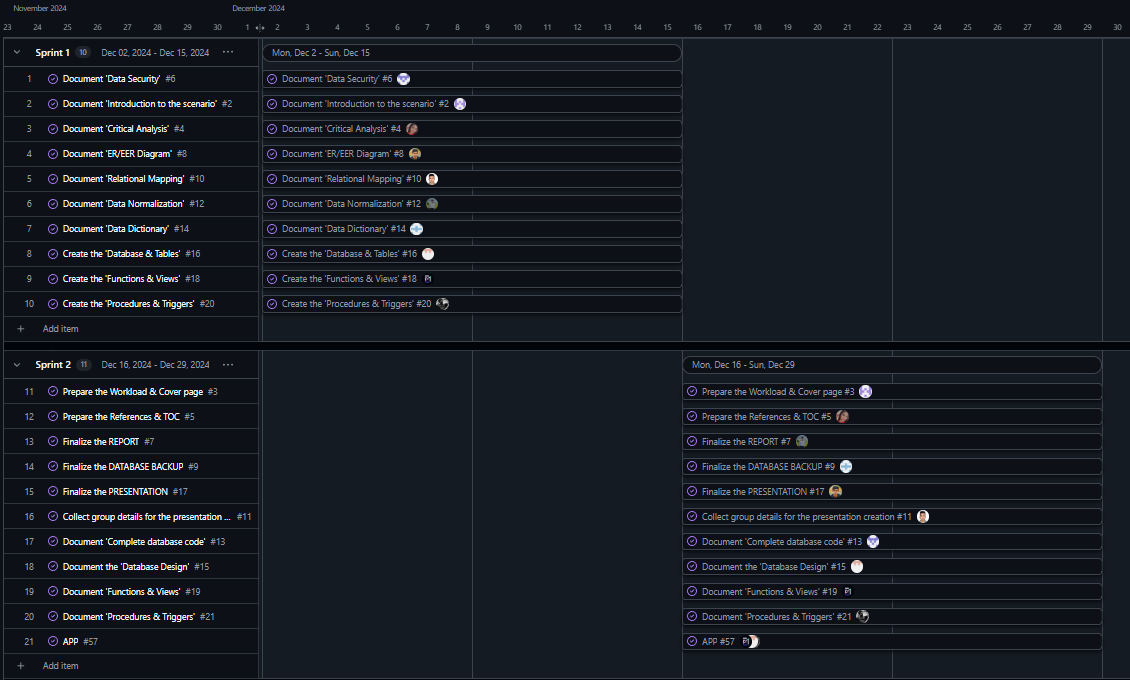
### 12.1.7. report genaration



### 12.1.8. discounts & taxes



## project management



# References

* *BYJU'S. (2024, December 30). What is Stockholder? Retrieved from What is Stockholder?: https://byjus.com/commerce/what-is-stockholder/#:~:text=A%20stockholder%20is%20also%20known,form%20of%20increased%20stock%20valuation.*
* *CFI. (2024, December 30). Data Validation. Retrieved from Data Validation: https://corporatefinanceinstitute.com/resources/data-science/data-validation/#:~:text=by%20Sebastian%20Taylor-,What%20is%20Data%20Validation%3F,of%20input%20and%20stored%20data.*
* *EDB. (2024, December 30). DATABASE MODERNIZATION. Retrieved from DATABASE MODERNIZATION: https://www.enterprisedb.com/use-case/oracle-database-modernization?campaign=&adgroup=&term=oracle%20database&matchtype=p&adposition=&device=c&gad\_source=1*
* *Geeks for Geeks. (2024, December 30). https://www.geeksforgeeks.org/normal-forms-in-dbms/. Retrieved from https://www.geeksforgeeks.org/normal-forms-in-dbms/: https://www.geeksforgeeks.org/normal-forms-in-dbms/*
* *IBM. (2024, December 30). What is a database. Retrieved from What is a database: https://www.ibm.com/think/topics/database*
* *Microsoft. (2024, December 30). Create entity relationship diagrams. Retrieved from Create entity relationship diagrams: https://support.microsoft.com/en-us/office/create-entity-relationship-diagrams-in-visio-7e44448c-9415-490b-8af1-f548f46ae90c*
* *Port Swigger. (2024, December 30). SQL injection. Retrieved from SQL injection: https://portswigger.net/web-security/sql-injection*
* *Resorce for Employee. (2024, December 30). What is employee management. Retrieved from What is employee management: https://resources.workable.com/hr-toolkit/what-is-employee-management*
* *Science Direct. (2024, December 30). Data Directory. Retrieved from Data Directory: https://www.sciencedirect.com/topics/computer-science/data-directory#:~:text=A%20data%20directory%20is%20defined,managing%20distributed%20databases%20and%20files.*
* *talend a quick company. (2024, December 30). What is Data Mapping. Retrieved from What is Data Mapping: https://www.talend.com/resources/data-mapping/#:~:text=Data%20mapping%20is%20the%20process,it%20accessible%20to%20decision%20makers.*
* *Wikipedia. (2024, December 30). Database normalization. Retrieved from Database normalization: https://en.wikipedia.org/wiki/Database\_normalization*
* *Wikipedia. (2024, December 30). Error management theory. Retrieved from Error management theory: https://en.wikipedia.org/wiki/Error\_management\_theory*
* *Wikipedia. (2024, December 30). Payment system. Retrieved from Payment system: https://en.wikipedia.org/wiki/Payment\_system*