Implementation of database systems for order management for Mi Pymes

Carmen Sofia Florez Jujaibioy Systems Engineer- Database Designer
Universidad Distrital Francisco José de Caldas
Bogotá D.C, Colombia. 2024

11 de junio de 2024

Resumen

The technical report provides a detailed description of the database design for order management in MSMEs. An analysis of the structure of the database is presented based on the relationship of entities and attributes. The strategy implemented to fill information in the database is described, detailing the methods and tools used for data extraction, transformation and loading. This strategy was designed with the purpose of guaranteeing the integrity and quality of the stored data, as well as maximizing the efficiency of the database filling process. Finally, the results of the tests carried out on the developed web services are presented, including functionality, performance and security tests.

DESIGN OF THE TOPIC BASE

User Stories

- **As a customer**, I want to browse a diverse and up-to-date product catalog, to easily find what I need and make my purchases: This story reflects the customer's need to access a well-organized and up-to-date product catalog. The database must support queries that allow customers to filter and search for products based on different criteria.
- **As a customer**, I want to place orders quickly and safely, for a convenient shopping experience without leaving home: Customers desire an efficient and secure ordering process. Queries should handle order insertion and inventory updating transactionally to maintain data integrity.
- **As a store manager**, I want to receive immediate alerts when orders are placed, to ensure a quick response and improve service efficiency: The store manager needs to be notified of new orders to respond quickly. A trigger in the database could automate this notification.
- **As a store manager**, I want to consult customers' purchase history and preferences, to offer personalized service and improve customer satisfaction: To offer personalized service, the manager needs to understand customer preferences. Queries should retrieve customer purchase history and preferences.

- **As a store manager**, I want to obtain detailed sales reports and data analysis, to identify trends and make strategic decisions based on reliable information: The manager requires detailed reports to make informed decisions. Queries should enable sales reporting and data analysis.

Identification of Entities

Entities represent real-world objects or concepts that have relevance to the system being modeled. In the context of a store order management application, it is essential to correctly identify these entities to ensure that the database accurately and efficiently captures all the information necessary for store operations. The main entities that could be relevant are described below:

- **Customer:** The person who makes the purchase. It should be listed only once.
- **Manager:** The individual in charge of operating the store day to day.
- **Order:** The set of items that the customer wants to purchase.
- **Ordered Item (Order Detail):** The specific products that are part of an order.
- **Product:** An individual item that the store offers for sale.
- **Category:** The classification of products.
- **Transaction:** The record of the sale, including payment and billing details.
- **Address (Delivery):** The information related to the delivery of the order to the customer.
- **Store:** The entity that represents the physical or online business.

Definition of Attributes

The attribute definition specifies the properties or characteristics of each identified entity. Attributes provide the details necessary to fully describe each entity, facilitating efficient organization and access to information. In the context of our order management application for a store, each principal entity has a set of clearly defined attributes.

Customer:

customer_id (PK)	Unique identifier of the customer
first_name	Customer's first name
last_name	Customer's last name
email	Customer's email address
address	Main address of the customer
phone	Customer's phone number
registration_date	Date the customer registered

Manager:

manager_id (PK)	Unique identifier of the manager
first_name	Manager's first name
last_name	Manager's last name
email	Manager's email address
phone	Manager's phone number
store_id (FK)	Identifier of the store they manage

Order:

order_id (PK)	Unique identifier of the order
customer_id (FK)	Identifier of the customer who placed the order
store_id (FK)	Identifier of the store where the order was placed
order_status	Status of the order (Pending, Shipped, Delivered)
order_date	Date the order was placed
total_order	Total amount of the order

Ordered Item (Order Detail):

ordered_item_id (PK)	Unique identifier of the ordered item
order_id (FK)	Identifier of the order the item belongs to
product_id (FK)	Identifier of the product ordered
quantity	Quantity of products in the order
unit_price	Price per unit of the product at the time of the order

Product:

product_id (PK)	Unique identifier of the product
product_name	Name of the product
product_price	Price of the product
stock_quantity	Current stock level of the product
product_description	Description of the product
category_id (FK)	Foreign key referencing the product's category

Category:

category_id (PK)	Unique identifier of the category
category_name	Name of the category
category_description	Description of the category

Transaction:

transaction_id (PK)	Unique identifier of the transaction
order_id (FK)	Identifier of the associated order
transaction_date	Date the transaction was made
total_amount	Total amount of the transaction
payment_method	Payment method used

Delivery:

delivery_id (PK)	Unique identifier of the delivery
order_id (FK)	Identifier of the associated order
delivery_address	Delivery address
delivery_date	Date of the delivery
delivery_instructions	Special instructions for the delivery

Store:

store_id (PK)	Unique identifier of the store
store_name	Name of the store
store_address	Address of the store
store_phone	Store's phone number
store_email	Store's email address

Relationship Identification

The identification of relationships between entities defines how the different components of the system interact with each other. For the order management application of a store, we use a relationship matrix, also known as an adjacency matrix, to identify and visualize these interactions. This tool allows us to map the connections between entities in a clear and structured way, facilitating the understanding of relationships and ensuring that all relevant interactions are properly represented in the model. In the relationship matrix, each row and column represents an entity, and the cells of the matrix indicate whether there is a relationship between the corresponding entities.

- **Customer to Order:** The customer has a relationship with the order because customers place orders in the store. This relationship is essential for tracking which orders each customer has made, managing the purchase history, and offering personalized service.

- **Owner to Store:** The owner has a relationship with the store because each store is owned by an owner. This relationship is fundamental for centralized administration and strategic decision-making regarding store operations.

- **Manager to Store:** The manager has a relationship with the store because each store needs a manager for its daily operation. This relationship ensures that there is a person responsible for the efficient handling and operation of the store.

- **Order to Ordered Item:** The order has a relationship with the ordered item because each order consists of one or more items that the customer has requested. This relationship is crucial for breaking down an order into its components, facilitating the tracking of the products included in each order.

- **Ordered Item to Product:** The ordered item has a relationship with the product because each ordered item represents a specific product from the inventory. This relationship helps to link the specific items in the orders with the products available in the inventory.

- **Product to Category:** The product has a relationship with the category because each product belongs to a specific category, such as electronics, clothing, etc. This relationship organizes the products in the inventory in a structured way, facilitating the search and analysis by categories.
- **Transaction to Order:** The transaction has a relationship with the order because each payment transaction is associated with a specific order. This relationship is essential for maintaining an accurate financial record and managing the payments related to each order.
- **Delivery to Order:** The delivery has a relationship with the order because each order must be delivered to a specific address. This relationship is key for delivery logistics and ensuring that each order arrives at the right place.
- **Store to Order:** The store has a relationship with the order because each order is processed in a specific store. This relationship allows tracking and analyzing sales by store, aiding in operational management and strategic decision-making about the performance of each store.

Definition of Constraints and Data Types

To ensure the integrity and consistency of the data in our order management database for the store, we have defined a series of specific constraints and properties for each entity. These constraints ensure that the entered data are valid and unique when necessary, and that the structure and relationships defined in our data model are maintained. Below are the detailed constraints and properties for each entity in the system:

Customer:

- **customer_id (auto):** `int [not null]` - Automatically generated unique identifier for each customer.
- **first_name:** `varchar(50) [not null]` - Customer's first name, cannot be empty.
- **last_name:** `varchar(50) [not null]` - Customer's last name, cannot be empty.
- **email:** `varchar(50) [unique, not null]` - Customer's email, must be unique and cannot be empty.
- **address:** `varchar(100)` - Customer's address, optional field.
- **phone:** `varchar(20)` - Customer's phone number, optional field.
- **registration_date:** `timestamp [not null]` - Date and time of customer registration, cannot be empty.

Manager:

- **manager_id (auto):** `int [not null]` - Automatically generated unique identifier for each manager.
- **first_name:** `varchar(50) [not null]` - Manager's first name, cannot be empty.

- **last_name:** `varchar(50) [not null]` - Manager's last name, cannot be empty.
- **email:** `varchar(50) [unique, not null]` - Manager's email, must be unique and cannot be empty.
- **phone:** `varchar(20)` - Manager's phone number, optional field.
- **store_id (auto):** `int [not null]` - Identifier of the store assigned to the manager, cannot be empty.

Order:

- **order_id (auto):** `int [not null]` - Automatically generated unique identifier for each order.
- **customer_id (auto):** `int [not null]` - Identifier of the customer who placed the order, cannot be empty.
- **store_id (auto):** `int [not null]` - Identifier of the store where the order was placed, cannot be empty.
- **order_status:** `varchar(20)` - Current status of the order (e.g., pending, shipped, delivered), optional field.
- **order_date:** `timestamp [not null]` - Date and time the order was placed, cannot be empty.
- **total_order:** `float [not null]` - Total amount of the order, cannot be empty.

Ordered Item (Order Detail):

- **ordered_item_id (auto):** `int [not null]` - Automatically generated unique identifier for each ordered item.
- **order_id (auto):** `int [not null]` - Identifier of the order the item belongs to, cannot be empty.
- **product_id (auto):** `int [not null]` - Identifier of the ordered product, cannot be empty.
- **quantity:** `int [not null]` - Quantity of units of the ordered product, cannot be empty.
- **unit_price:** `float [not null]` - Unit price of the ordered item, cannot be empty.

Product:

- **product_id (auto):** `int [not null]` - Automatically generated unique identifier for each product.
- **product_name:** `varchar(50) [unique, not null]` - Product name, must be unique and cannot be empty.
- **product_price:** `float [not null]` - Product price, cannot be empty.

- **stock_quantity:** `int [not null]` - Quantity of units available in inventory, cannot be empty.
- **category_id (auto):** `int [not null]` - Identifier of the category the product belongs to, cannot be empty.

Category:

- **category_id (auto):** `int [not null]` - Automatically generated unique identifier for each category.
- **category_name:** `varchar(50) [unique, not null]` - Category name, must be unique and cannot be empty.
- **category_description:** `varchar(200)` - Category description, optional field.

Transaction:

- **transaction_id (uuid):** `uuid [not null]` - Automatically generated unique identifier for each transaction as UUID.
- **order_id (auto):** `int [not null]` - Identifier of the order associated with the transaction, cannot be empty.
- **transaction_date:** `timestamp [not null]` - Date and time of the transaction, cannot be empty.
- **total_amount:** `float [not null]` - Total amount of the transaction, cannot be empty.
- **payment_method:** `varchar(20) [not null]` - Payment method used in the transaction, cannot be empty.

Delivery:

- **delivery_id (auto):** `int [not null]` - Automatically generated unique identifier for each delivery.
- **order_id (auto):** `int [not null]` - Identifier of the order associated with the delivery, cannot be empty.
- **delivery_address:** `varchar(100)` - Address where the delivery will be made, optional field.
- **delivery_date:** `timestamp [not null]` - Expected date and time for the delivery, cannot be empty.
- **delivery_instructions:** `varchar(200)` - Additional instructions for the delivery, optional field.

Store:

- **store_id (auto):** `int [not null]` - Automatically generated unique identifier for each store.
- **store_name:** `varchar(50) [not null]` - Store name, cannot be empty.
- **store_address:** `varchar(100)` - Store address, optional field.
- **store_phone:** `varchar(20)` - Store phone number, optional field.
- **store_email:** `varchar(50) [unique, not null]` - Store email, must be unique and cannot be empty.

Construction of the ER Diagram

For the creation of the Entity-Relationship (ER) diagram, I used the online tool draw.io. I started by setting up a new document and adding rectangles to represent each entity. Within these rectangles, I placed the corresponding attributes. Then, I connected the entities with lines to represent the relationships between them, making sure to indicate the cardinality at the end of each line using the appropriate shapes.
This ER diagram is fundamental for the implementation of the database, as it provides a clear and detailed visual representation of the entities, their attributes, and the relationships between them. This ensures that all the requirements of the store's order management system are adequately covered and facilitates communication between developers and other stakeholders in the project, ensuring a robust and efficient database.

DATA DICTIONARY

The data dictionary is an essential tool in database management that provides a detailed description of the structure, content and characteristics of the stored data. In the context of our order management database, the data dictionary plays a vital role in providing a complete reference of each table, its columns and the applied constraints. This detailed compilation of metadata and descriptions allows users to quickly understand the organization of order-related information, including customer details, products ordered, and transaction records. Additionally, the data dictionary makes it easier to develop and maintain applications that interact with the database by providing clear guidance on how to access and manipulate data effectively.

Table: Customer

Description: Stores information about customers.

- **Table Name**: Customer
- **Description**: Stores information about customers.
- **Columns**:
  - **customer-id**

    - **Data Type**: INT
    - **Description**: Unique identifier for the customer
    - **Constraints**: PRIMARY KEY, NOT NULL
  - **customer-name**
    - **Data Type**: VARCHAR(50)
    - **Description**: Customer's first name
    - **Constraints**: NOT NULL
  - **customer-lastname**
    - **Data Type**: VARCHAR(50)
    - **Description**: Customer's last name
    - **Constraints**: NOT NULL
  - **customer-email**
    - **Data Type**: VARCHAR(50)
    - **Description**: Customer's email address
    - **Constraints**: UNIQUE, NOT NULL
  - **customer-address**
    - **Data Type**: VARCHAR(100)
    - **Description**: Customer's address
  - **customer-phone**
    - **Data Type**: VARCHAR(20)
    - **Description**: Customer's phone number
  - **registration-date**
    - **Data Type**: TIMESTAMP
    - **Description**: Registration date
    - **Constraints**: NOT NULL

Table: Manager

Description: Stores information about managers.

- **Table Name**: Manager
- **Description**: Stores information about managers.
- **Columns**:
  - **manager-id**
    - **Data Type**: INT
    - **Description**: Unique identifier for the manager
    - **Constraints**: PRIMARY KEY, NOT NULL
  - **manager-name**
    - **Data Type**: VARCHAR(50)
    - **Description**: Manager's first name

    - **Constraints**: NOT NULL
  - **manager-lastname**
    - **Data Type**: VARCHAR(50)
    - **Description**: Manager’s last name
    - **Constraints**: NOT NULL
  - **manager-email**
    - **Data Type**: VARCHAR(50)
    - **Description**: Manager’s email address
    - **Constraints**: UNIQUE, NOT NULL
  - **manager-phone**
    - **Data Type**: VARCHAR(20)
    - **Description**: Manager’s phone number
  - **store-id**
    - **Data Type**: INT
    - **Description**: Store identifier assigned to the manager
    - **Constraints**: NOT NULL

Table: Category

Description: Stores information about product categories.

- **Table Name**: Category
- **Description**: Stores information about product categories.
- **Columns**:
  - **category-id**
    - **Data Type**: INT
    - **Description**: Unique identifier for the category
    - **Constraints**: PRIMARY KEY, NOT NULL
  - **category-name**
    - **Data Type**: VARCHAR(50)
    - **Description**: Name of the category
    - **Constraints**: UNIQUE, NOT NULL
  - **category-description**
    - **Data Type**: VARCHAR(200)
    - **Description**: Description of the category

Table: Store

Description: Stores information about stores.

- **Table Name**: Store
- **Description**: Stores information about stores.
- **Columns**:
  - **store-id**
    - **Data Type**: INT
    - **Description**: Unique identifier for the store
    - **Constraints**: PRIMARY KEY, NOT NULL
  - **store-name**
    - **Data Type**: VARCHAR(50)
    - **Description**: Name of the store
    - **Constraints**: NOT NULL
  - **store-address**
    - **Data Type**: VARCHAR(100)
    - **Description**: Address of the store
  - **store-phone**
    - **Data Type**: VARCHAR(20)
    - **Description**: Phone number of the store
  - **store-email**
    - **Data Type**: VARCHAR(50)
    - **Description**: Email address of the store
    - **Constraints**: UNIQUE, NOT NULL

Table: Product

Description: Stores information about products.

- **Table Name**: Product
- **Description**: Stores information about products.
- **Columns**:
  - **product-id**
    - **Data Type**: INT
    - **Description**: Unique identifier for the product
    - **Constraints**: PRIMARY KEY, NOT NULL
  - **product-name**
    - **Data Type**: VARCHAR(50)
    - **Description**: Name of the product

    - **Constraints**: UNIQUE, NOT NULL
- **product-price**
    - **Data Type**: FLOAT
    - **Description**: Price of the product
    - **Constraints**: NOT NULL
- **stock-quantity**
    - **Data Type**: INT
    - **Description**: Quantity in stock
    - **Constraints**: NOT NULL
- **category-id**
    - **Data Type**: INT
    - **Description**: Identifier of the product category
    - **Constraints**: FOREIGN KEY, NOT NULL

DEFINITION OF WEB SERVICES THAT ARE REQUIRED BASED ON USER STORIES

In this section, a detailed description of the web services designed based on the needs and requirements of users is presented. Each web service is backed by SQL queries performed in MySQL, which have been directly associated with the user stories they satisfy. This approach provides a comprehensive view of how the technical functionality of the database aligns with business objectives and user expectations. By linking each SQL query to a specific user story, you ensure that web service development meets functional requirements and improves the user experience in your application.

1. As a Client

1.1. Browse a Diverse and Updated Product Catalog

– View 1: ProductCatalogView
– Description: Provides a list of available products in the store's catalog.
CREATE OR REPLACE VIEW ProductCatalogView AS
SELECT product$_{n}ame, product_{p}rice, stock_{q}uantity$
$FROM Product$
$ORDER BY product_{n}ame$;

1.2. Place Orders Quickly and Securely

– Procedimiento 1: InsertOrder
– Description: Allows customers to place a new order for products.
DELIMITER //
CREATE PROCEDURE InsertOrder (
IN customer$_{i}dINT$,
$INproduct_{i}dINT$,
$INamountINT$

)

$BEGIN$

$INSERTINTOOrderr(customer_{i}d, order_{d}ate, order_{s}tatus)$

$VALUES(customer_{i}d, NOW(),'pending');$

$SET@order_{i}d = LAST_{I}NSERT_{I}D();$

$INSERTINTOOrderedItem(order_{i}d, product_{i}d, quantity)$

$VALUES(@order_{i}d, product_{i}d, amount);$

$UPDATEProduct$

$SETstock_{q}uantity = stock_{q}uantity - amount$

$WHEREproduct_{i}d = product_{i}d;$

$END//$

$DELIMITER;$

1.3. Consult Purchase History and Customer Preferences

– View 3: CustomerPurchaseHistoryView

– Description: Provides customer purchase history and preferences.

CREATE OR REPLACE VIEW CustomerPurchaseHistoryView AS

SELECT c.customer$_{i}d, c.customer_{n}ame, o.order_{i}d, o.order_{d}ate, oi.product_{i}d, oi.quantity, oi.unit_{p}rice$

$FROMCustomerc$

$JOINOrderroONc.customer_{i}d = o.customer_{i}d$

$JOINOrderedItemoiONo.order_{i}d = oi.order_{i}d;$

1.4. as Store Manager

– View 10: RecommendedProducts

– Description: Displays a list of recommended products for a specific customer.

CREATE OR REPLACE VIEW RecommendedProducts AS

SELECT p.product$_{i}d, p.product_{n}ame, p.product_{p}rice$

$FROMProductp$

$JOINOrderedItemoiONp.product_{i}d = oi.product_{i}d$

$JOINOrderroONoi.order_{i}d = o.order_{i}d$

$WHEREo.customer_{i}d = 1$

$GROUPBYp.product_{i}d, p.product_{n}ame, p.product_{p}rice$

$ORDERBYCOUNT(*)DESC;$

2. As Store Manager

2.1. Receive Immediate Alerts When Orders Are Placed

– Trigger 1: NotifyManagerOnNewOrder

– Description: Notifies the manager when a new order is placed in the store.

DELIMITER //

CREATE TRIGGER NotifyManagerOnNewOrder

AFTER INSERT ON Orderr

FOR EACH ROW

BEGIN
DECLARE manager$_{e}mailVARCHAR(100);$
$--Getmanager'semail$
$SELECTemailINTOmanager_{e}mailFROMManagerWHEREmanager_{i}d = 1;$

– Send notification email to the manager
INSERT INTO Notification (recipient$_{e}mail, subject, message)$
$VALUES(manager_{e}mail,'Neworderplaced', CONCAT('AneworderhasbeenplacedwithID:'$
$, NEW.order_{i}d));$
$END//$
$DELIMITER;$

2.2. Consult Purchase History and Customer Preferences

– View 3: CustomerPurchaseHistoryView
– Description: Allows the manager to access customer purchase history and preferences.
CREATE OR REPLACE VIEW CustomerPurchaseHistoryView AS
SELECT c.customer$_{i}d, c.customer_{n}ame, o.order_{i}d, o.order_{d}ate, oi.product_{i}d, oi.quantity, oi.unit_{p}rice$
$FROMCustomerc$
$JOINOrderroONc.customer_{i}d = o.customer_{i}d$
$JOINOrderedItemoiONo.order_{i}d = oi.order_{i}d;$

2.3. Get Detailed Sales Reports and Data Analysis

– View 4: SalesReportsDataAnalysisView
– Description: Provides detailed sales reports and data analysis for strategic decision-making.
CREATE OR REPLACE VIEW SalesReportsDataAnalysisView AS
SELECT o.order$_{i}d, o.order_{d}ate, oi.product_{i}d, p.product_{n}ame, oi.quantity, oi.unit_{p}rice$
$FROMOrderro$
$JOINOrderedItemoiONo.order_{i}d = oi.order_{i}d$
$JOINProductpONoi.product_{i}d = p.product_{i}d;$

2.4. Effective Order Tracking System

– View 7: MonthlySalesSummary
– Description: Offers a monthly summary of sales to identify trends and make strategic decisions.
CREATE OR REPLACE VIEW MonthlySalesSummary AS
SELECT YEAR(order$_{d}ate)ASyear, MONTH(order_{d}ate)ASmonth, SUM(oi.quantity*oi.unit_{p}rice)AStotal_{sa}$
$FROMOrderro$
$JOINOrderedItemoiONo.order_{i}d = oi.order_{i}d$
$GROUPBYYEAR(order_{d}ate), MONTH(order_{d}ate)$
$ORDERBYyearDESC, monthDESC$

DATABASE POPULATION STRATEGY

Our strategy for populating the database involves several steps. First, we identify the tables that need to be populated, prioritizing them based on foreign key constraints to ensure referential integrity. Next, we generate coherent test data for each table, taking into account relationships between them to maintain data consistency. We use 'INSERT INTO' statements to add data rows to each table, carefully verifying that the data is valid and consistent with the database constraints.
During this process, we conduct thorough testing using 'SELECT' queries to ensure that data is inserted correctly and that there are no integrity conflicts. If we encounter any errors or inconsistencies, we iterate through the process, adjusting the data and queries as needed to address the identified issues.
Finally, we thoroughly document the entire process, including the queries used, the adjustments made, and any issues encountered during data insertion. This documentation allows us to maintain a clear record of how the database was populated, facilitating review and identification of potential issues in the future, and ensuring that the database is populated with coherent and valid data.

SQL QUERIES FOR PRODUCT AND ORDER MANAGEMENT

View Creation

1. Product Catalog View

Name: ProductCatalogView
Description: This view displays a list of available products in the store's catalog, including the product name, price, and stock quantity. It is ordered by the product name.

2. Completed Orders View

Name: CompletedOrdersView
Description: This view shows all orders that have been completed. It includes all columns from the `Orderr` table and filters the results to display only those where the order status is 'completed'.

3. Customer Purchase History and Preferences View

Name: CustomerPurchaseHistoryView
Description: This view provides the purchase history and preferences of customers. It shows information about the customer, their orders, and the products included in those orders.

4. Sales Reports and Data Analysis View

Name: SalesReportsDataAnalysisView
Description: This view is designed to provide relevant data for sales reports and data analysis. It includes information about orders, sold products, quantity, and unit price.

5. Low Inventory View

Name: LowInventory
Description: This view shows products with stock quantities less than 10 units, helping to identify products that need to be restocked.

6. Top Selling Products View

Name: TopSellingProducts
Description: This view displays the top-selling products, ordered by total quantity sold in descending order. It helps to identify the most popular products.

7. Monthly Sales Summary View

Name: MonthlySalesSummary
Description: This view provides a summary of monthly sales, showing the year, month, and total sales for each period.

8. Products by Category View

Name: ProductsByCategory
Description: This view shows products along with their respective categories, making it easier to identify products by category.

9. Customer Purchase History View

Name: CustomerPurchaseHistory
Description: This view shows the purchase history of a specific customer (with `customer_id` = 2), including order details and acquired products.

10. Recommended Products View

Name: RecommendedProducts
Description: This view shows recommended products for a specific customer (with `customer_id` = 1), based on their previous purchases. The products are ordered by how frequently they have been bought by the customer.

Procedure Creation

11. Procedure to Insert a New Order

Name: InsertOrder
Description: This procedure inserts a new order into the system. It takes the customer ID, product ID, and quantity as inputs. The procedure inserts a new record into the `Orderr` table with the current date and 'pending' status, retrieves the last inserted order ID, inserts the order details into the `Ordered_Item` table, and updates the product's stock quantity.

12. Procedure to Update Order Status

Name: UpdateOrderStatus
Description: This procedure updates the status of an existing order. It takes the order ID and the new status as inputs and updates the corresponding order's status in the `Orderr` table.

13. Procedure to Get Customer Purchase History

Name: CustomerPurchaseHistory
Description: This procedure retrieves the purchase history of a specific customer. It takes the customer ID as input and returns details of the customer's orders, including order ID, order date, product ID, product name, quantity, and unit price.

14. Procedure to Cancel an Order

Name: CancelOrder
Description: This procedure cancels an existing order. It takes the order ID as input, retrieves the products and quantities in the order, updates the stock quantity for each product, and deletes the order from the `Orderr` table.

15. Procedure to Update Product Stock Quantity

Name: UpdateStockQuantity
Description: This procedure updates the stock quantity of a specific product. It takes the product ID and the new stock quantity as inputs and updates the corresponding product's stock quantity in the `Product` table.

16. Procedure to Calculate Total Sales for a Customer

Name: CalculateTotalSales
Description: This procedure calculates the total sales for a specific customer. It takes the customer ID as input and outputs the total sales amount for that customer.

17. Procedure to Get Customer Order History

Name: GetOrderHistory
Description: This procedure retrieves the order history of a specific customer. It takes the customer ID as input and returns all orders made by that customer.

18. Procedure to Get Product Details

Name: GetProductDetails
Description: This procedure retrieves details of a specific product. It takes the product ID as input and outputs the product name, price, and stock quantity.

19. Procedure to Update Product Price

Name: UpdateProductPrice
Description: This procedure updates the price of a specific product. It takes the product ID and the new price as inputs and updates the corresponding product's price in the `Product` table.

20. Procedure to Update Customer Details

Name: UpdateCustomerDetails
Description: This procedure updates the details of a specific customer. It takes the customer ID, new name, and new email as inputs and updates the corresponding customer's details in the `Customer` table.

21. Procedure to Add a New Product

Name: AddNewProduct
Description: This procedure adds a new product to the system. It takes the product name, product price, stock quantity, and category ID as inputs and inserts a new record into the `Product` table with these details.

Trigger Creation

22. Trigger to Notify Manager on New Order

Name: NotifyManagerOnNewOrder
Description: This trigger notifies the manager via email when a new order is placed. After a new record is inserted into the `Orderr` table, the trigger retrieves the manager's email and sends a notification email containing the new order ID.

23. Trigger to Update Inventory on Order Completion

Name: UpdateInventoryOnOrderCompletion
Description: This trigger updates the product inventory when an order is marked as completed. It checks if the order status has changed to 'completed' and, if so, updates the stock quantity of the ordered products based on the quantities specified in the `Ordered_Item` table.

24. Trigger to Log Changes in Order History

Name: OrderHistoryLog
Description: This trigger logs changes to order statuses in the `OrderHistory` table. After an update on the `Orderr` table, it inserts a record into the `OrderHistory` table, capturing the order ID, old status, new status, and the change date.

25. Trigger to Update Order Total on New Order Item

Name: UpdateOrderTotalOnInsert
Description: This trigger updates the total order amount whenever a new order item is inserted. After a new record is inserted into the `Ordered_Item` table, it updates the total amount of the corresponding order in the `Orderr` table by adding the product of the new item's quantity and unit price.

26. Trigger to Prevent Customer Deletion with Associated Orders

Name: PreventCustomerDeletion
Description: This trigger prevents the deletion of a customer if they have associated orders. Before deleting a record from the `Customer` table, it checks if there are any orders associated

with the customer. If there are, it raises an error and prevents the deletion, ensuring data integrity.

SQL Queries

26. Select Products with Price Greater than Average

Description: This query selects products whose price is greater than the average price of all products in the database. It retrieves the product ID, product name, and product price for these products.

27. Select Products Priced Above Average Store Price

Description: This query selects products that have a price higher than the average price of all products in the store. It retrieves the product ID, product name, and product price for these products.

28. Update Stock for Products Sold in the Last Month

Description: This query updates the stock quantity of products that have been sold in the last month. It subtracts the total quantity of each product sold in the last month from the current stock quantity. Note that the placeholder `123` should be replaced with the actual product ID that needs to be updated.

29. Get Customer Name and Total Spent on Last Purchase

Description: This query retrieves the name of the customer and the total amount they spent on their last purchase. It calculates the total spent by summing the product of the quantity and unit price of the items ordered. The query uses a left join to ensure that all customers are included, even if they haven't made any purchases.