GLOBAL GADGETS RETAILER DATABASE DESIGN AND IMPLEMENTATION

NAME

AKITOYE MICHAEL OLUWASEYI

COURSE

SPLENDOR ANALYTICS: SQL IMMERSIVE 1

2025

TUTOR

MR UCHENNA SPLENDOR

EXECUTIVE SUMMARY / ABSTRACT

GlobalGadgetsDB is a SQL Server–based relational database system designed to support the operations of a consumer Electronics retail company. The system manages core business entities such as Products, Suppliers, Customers, Orders, Payments, Shipping and Inventory.

The project includes a fully normalized schema with fifteen interconnected tables, supported by stored procedures, functions, views, and triggers to enforce business logic and maintain data integrity. It automates key workflows such as restocking inventory on order cancellation, tracking employee activity, and validating payments.

This system also incorporates backup and recovery mechanisms to ensure data resilience and restoration in case of. Through structured data creation with non-clustered indexes, data insertion, adding constraints and query optimization, the project practically demonstrates how SQL – based solutions can enhance operational efficiency and decision – making in a retail – based environment.

This report documents the design, implementation, and strategic management of the database, including schema diagrams, T-SQL scripts, views, stored procedures , functions, triggers and recommendations for scalability, security, and backup.

TABLE OF CONTENTS

II. SECTION 1: INTRODUCTION AND SCOPE

1.1 PROJECT CONTEXT AND CLIENT REQUIREMENT

Global Gadgets is a mid-sized consumer electronics retail Company that deals with Gadgets and Electronics. The business faced challenges managing its growing inventory, tracking customer orders, coordinating supplier deliveries, and maintaining accurate payment records. The client required an accurate and business based database system to streamline operations, enforce business rules, and support decision-making.

Key requirements included:

- A relational schema normalized to 3NF to eliminate redundancy and improve data integrity.
- Automation of inventory restocking when orders are cancelled.
- Support for multiple payment types (Cash, Card, Transfer).
- Stored procedures and functions to simplify complex operations and avoid repeated actions.
- Views and queries for sales analysis and customer, product behavior insights.
- Triggers to enforce real-time business logic and maintain audit trails and make sure Product price remains positive.

The solution needed to be implemented in SQL Server and designed for extensibility, security, and performance

1.2 METHODOLOGY

The database was developed using a structured, task-driven methodology aligned with best practices in relational design and SQL Server implementation:

Requirement Analysis: Reviewed client needs and translated them into technical specifications.

Conceptual Modeling: Designed an Entity-Relationship Diagram (ERD) to visualize core entities and relationships.

Normalization: Applied UNF up to 3NF to ensure minimal redundancy and optimal structure

Schema Implementation: Created tables using T-SQL with appropriate data types, constraints, and relationships.

Data Population: Inserted sample data to simulate real-world operations and check if everything works as it should.

Functional Development: Built stored procedures, functions, views, and triggers to implement business logic.
Testing and Validation: Ran queries and updates to verify functionality, integrity, and performance.
Documentation and Deployment: Structured the project in GitHub with scripts, diagrams and a backup file for easy reuse.

This approach ensured that the final system was robust, maintainable, and aligned with the client's operational goals.

SECTION 2: DATABASE DESIGN AND NORMALIZATION

2.1 CONCEPTUAL DATA MODEL (E-R Diagram)

The conceptual data model was developed using an Entity-Relationship Diagram (ERD) to represent the core entities and relationships within the GlobalGadgets retail system. The ERD includes entities such as:

- Products
- Suppliers
- Customers
- Orders
- OrderDetails
- Payments
- Employees
- Shipping
- Inventory

Key relationships:

- One-to-many between Customers and Orders
- Many-to-many between Orders and Products via OrderDetails
- One-to-many between Suppliers and Products
- One-to-many between Employees and Orders
- One-to-one between Orders and Payments

The ERD diagram below shows the relationships between each tables

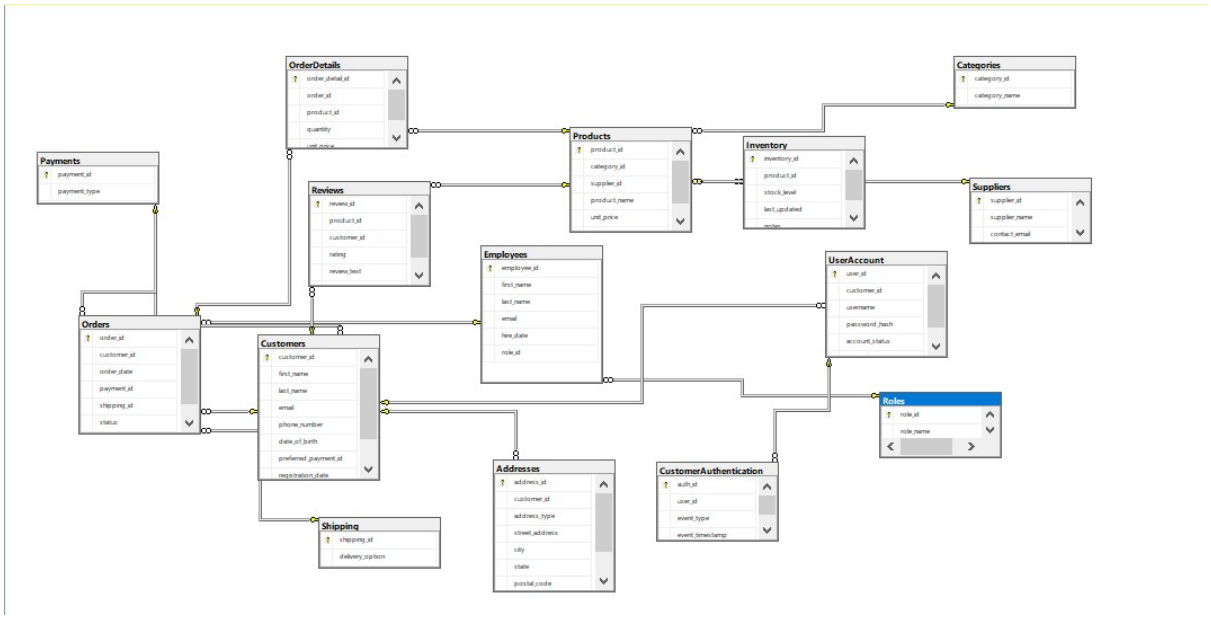


Fig 1.1 ERD Diagram

2.2 NORMALIZATION PROCESS AND JUSTIFICATION (UNF → 3NF)

The database was normalized through the following stages:

- UNF (Unnormalized Form): Initial data captured in flat tables with repeating groups.
- 1NF (First Normal Form): Removed repeating groups; ensured atomicity of attributes.
- 2NF (Second Normal Form): Eliminated partial dependencies by creating separate tables for multi-attribute keys.
- 3NF (Third Normal Form): Removed transitive dependencies; ensured that non-key attributes depend only on the primary key.

Normalization helped reduce redundancy, improve data integrity, and optimize query performance. For example:

- Customer details were separated from Orders.
- Products and Suppliers were decoupled to allow flexible sourcing.
- Payments were separated to support multiple transaction types.

2.3 FINAL SCHEMA IMPLEMENTATION

The final schema was implemented using **T-SQL CREATE TABLE** statements and total of fifteen tables were created with appropriate data types, primary keys, foreign keys, and constraints.

Highlights include:

- Use of **INT, VARCHAR, NVARCHAR, DATE,** and **DECIMAL** for precision.
- Foreign key constraints to enforce referential integrity.
- **CHECK** constraints for valid payment types, statuses of orders and to make sure the product price is always positive.
- **DEFAULT** values for timestamps and flags.

Database Creation

The database called 'GlobalGadgetsDB' was created to store all information regarding the retail platform with associated tables needed in it.

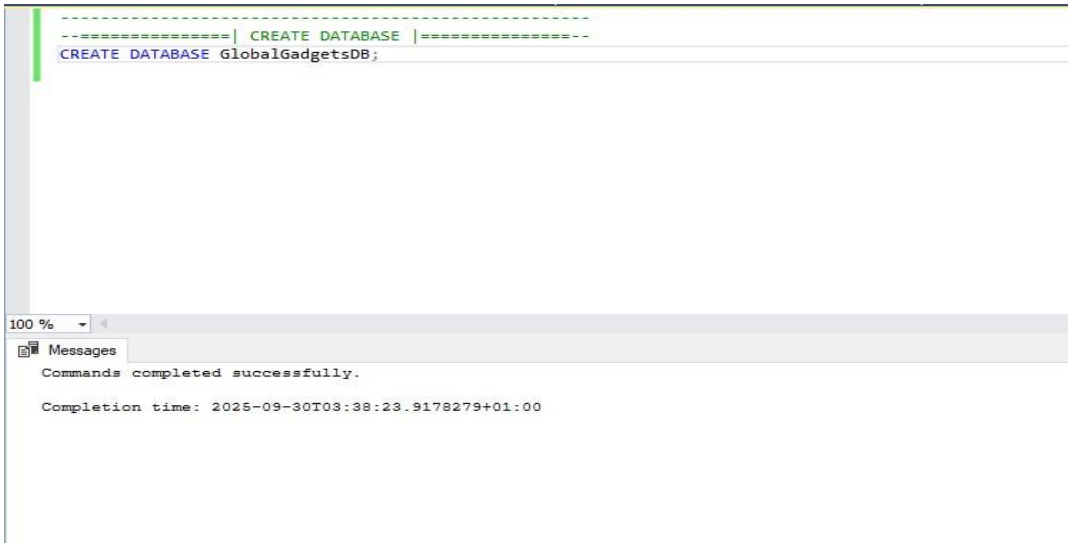


Fig 1.1Screenshot showing the SQL script used to create the Global Gadgets Database.

Creation process of each tables are as follows:

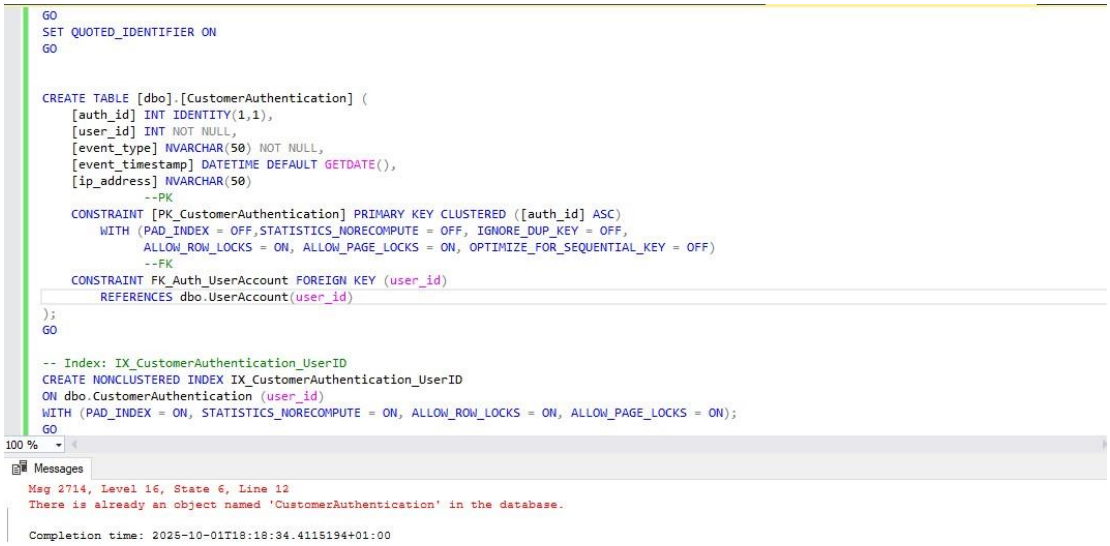
Categories Table Creation

This table defines the product categories and their names , helping organizing the inventory.

Fig1.2 Screenshot showing the SQL script used to create the Categories Table, including primary and unique constraints.

Customer Authentication Table Creation

This table verifies login credentials by checking the provided username and password hash against records in the customers table ensuring access to the system.



```
GO
SET QUOTED_IDENTIFIER ON
GO


CREATE TABLE [dbo].[CustomerAuthentication] (
    [auth_id] INT IDENTITY(1,1),
    [user_id] INT NOT NULL,
    [event_type] NVARCHAR(50) NOT NULL,
    [event_timestamp] DATETIME DEFAULT GETDATE(),
    [ip_address] NVARCHAR(50)
                --PK
    CONSTRAINT [PK_CustomerAuthentication] PRIMARY KEY CLUSTERED ([auth_id] ASC)
        WITH (PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
              --FK
    CONSTRAINT FK_Auth_UserAccount FOREIGN KEY (user_id)
        REFERENCES dbo.UserAccount(user_id)
);
GO

-- Index: IX_CustomerAuthentication_UserID
CREATE NONCLUSTERED INDEX IX_CustomerAuthentication_UserID
ON dbo.CustomerAuthentication (user_id)
WITH (PAD_INDEX = ON, STATISTICS_NORECOMPUTE = ON, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON);
GO
```

100 %

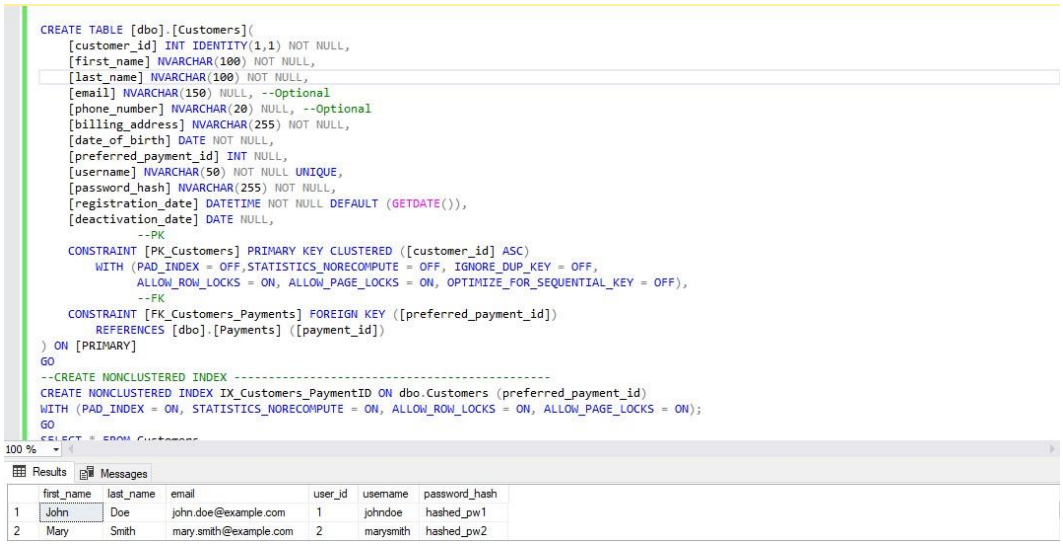
Messages

```
Msg 2714, Level 16, State 6, Line 12
There is already an object named 'CustomerAuthentication' in the database.

Completion time: 2025-10-01T18:18:34.4115194+01:00
```

Fig 1.3 Screenshot showing the SQL script used to create the Categories Table, including primary and foreign key constraints.

Customers Table Creation

This tables is one of the key tables, it stores customer details including contact information and date of birth.



```
CREATE TABLE [dbo].[Customers](
    [customer_id] INT IDENTITY(1,1) NOT NULL,
    [first_name] NVARCHAR(100) NOT NULL,
    [last_name] NVARCHAR(100) NOT NULL,
    [email] NVARCHAR(150) NULL, --Optional
    [phone_number] NVARCHAR(20) NULL, --Optional
    [billing_address] NVARCHAR(255) NOT NULL,
    [date_of_birth] DATE NOT NULL,
    [preferred_payment_id] INT NULL,
    [username] NVARCHAR(50) NOT NULL UNIQUE,
    [password_hash] NVARCHAR(255) NOT NULL,
    [registration_date] DATETIME NOT NULL DEFAULT (GETDATE()),
    [deactivation_date] DATE NULL,
                --PK
    CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED ([customer_id] ASC)
        WITH (PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF),
              --FK
    CONSTRAINT [FK_Customers_Payments] FOREIGN KEY ([preferred_payment_id])
        REFERENCES [dbo].[Payments] ([payment_id])
) ON [PRIMARY]
GO
--CREATE NONCLUSTERED INDEX ---------------------------------------------
CREATE NONCLUSTERED INDEX IX_Customers_PaymentID ON dbo.Customers (preferred_payment_id)
WITH (PAD_INDEX = ON, STATISTICS_NORECOMPUTE = ON, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON);
GO
```

100 %

Results | Messages

	first_name	last_name	email	user_id	username	password_hash
1	John	Doe	john.doe@example.com	1	johndoe	hashed_pw1
2	Mary	Smith	mary.smith@example.com	2	marysmith	hashed_pw2

Fig 1.4 Screenshot showing the SQL script used to create the Customers Table, including primary, foreign and unique constraints.

Employee Table Creation

This table shows where each employee is linked to a role using a foreign key and stores datils of the employees.

```
CREATE TABLE [dbo].[Employees](
    [employee_id] INT IDENTITY(1,1) NOT NULL,
    [first_name] NVARCHAR(200) NOT NULL,
    [last_name] NVARCHAR(150) NULL,
    [email] VARCHAR(100) NOT NULL UNIQUE,
    [hire_date] DATE,
    [role_id] INT NOT NULL
    --PK
    CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED ([employee_id] ASC)
        WITH (PAD_INDEX = OFF,
              STATISTICS_NORECOMPUTE = OFF,
              IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON,
              ALLOW_PAGE_LOCKS = ON,
              OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
              --FK
    CONSTRAINT [FK_Employees_Role_id] FOREIGN KEY ([role_id])
        REFERENCES [dbo].[Roles] ([role_id])
) ON [PRIMARY]
GO
--=============================================================================
--==CREATE NONCLUSTERED INDEX--================================================
CREATE NONCLUSTERED INDEX IX_Employees_Email ON dbo.Employees ([email]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
CREATE NONCLUSTERED INDEX IX_Employees_RoleID ON dbo.Employees ([role_id]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
100 %
Messages
Commands completed successfully.
```

Fig 1.5 Screenshot showing the SQL script used to create the Employees Table, including primary, foreign and unique constraints.

Inventory Table Creation

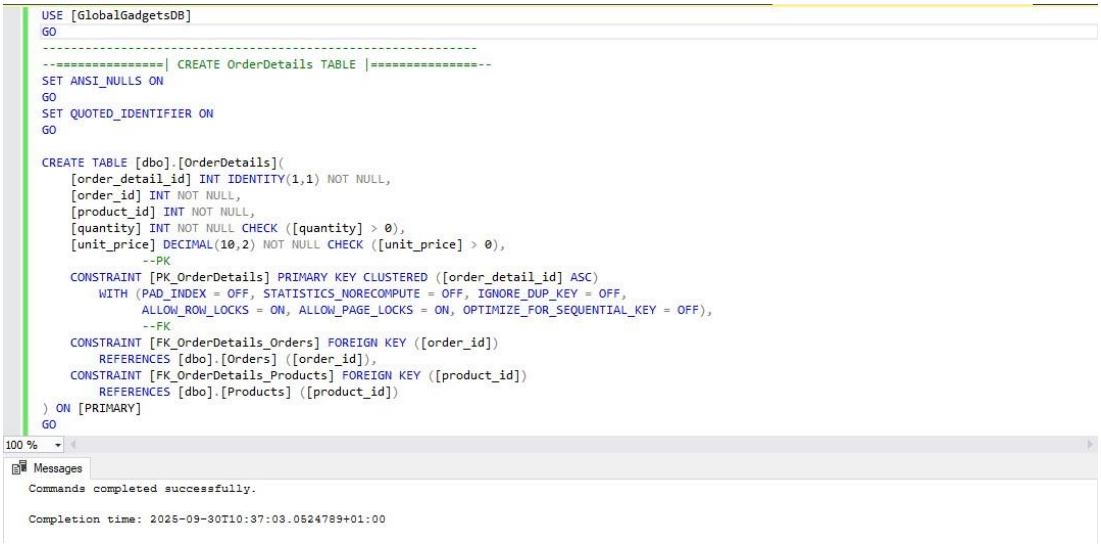
This table tracks product stock levels and restock dates.

```
------------------------------------------------------------
--===============| CREATE Inventory TABLE |===============--
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Inventory](
    [inventory_id] INT IDENTITY(1,1) NOT NULL,
    [product_id] INT NOT NULL,
    [stock_level] INT NOT NULL CHECK ([stock_level] >= 0),
    [last_updated] DATETIME NOT NULL DEFAULT (GETDATE()),
    [notes] NVARCHAR(500) NULL,
    --PK
    CONSTRAINT [PK_Inventory] PRIMARY KEY CLUSTERED ([inventory_id] ASC)
        WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF),
    --FK
    CONSTRAINT [FK_Inventory_Products] FOREIGN KEY ([product_id])
        REFERENCES [dbo].[Products] ([product_id])
) ON [PRIMARY]
GO
--=============================================================================
--==CREATE NONCLUSTERED INDEX--================================================
CREATE NONCLUSTERED INDEX IX_Inventory_ProductID ON dbo.Inventory ([product_id]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
100 %
Messages
```

Fig 1.6 Screenshot showing the SQL script used to create the Inventory Table, including primary, foreign and check constraints.

Order Details Table Creation

This table connect products to specific orders and records quantities ordered.



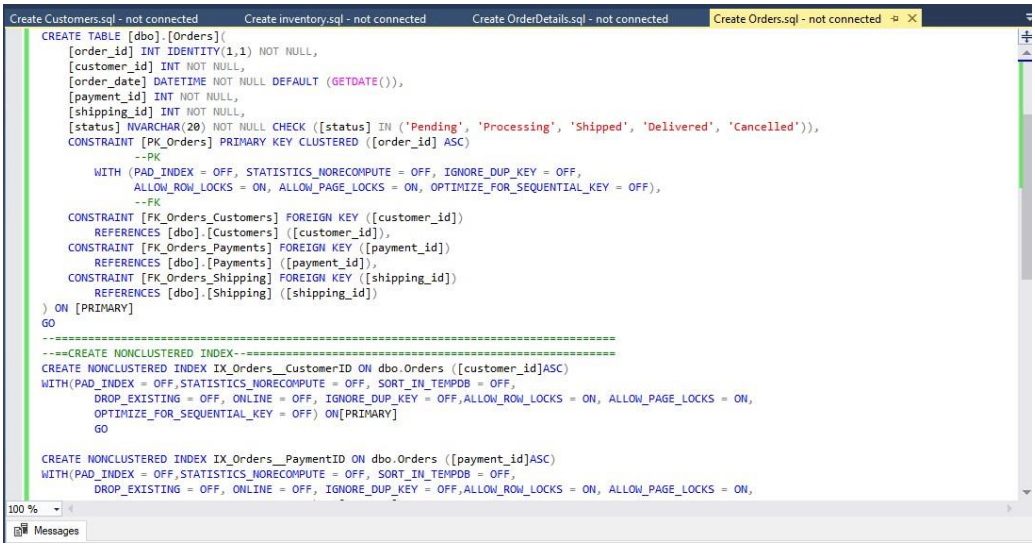
```
USE [GlobalGadgetsDB]
GO
------------------------------------------------------------
--===============| CREATE OrderDetails TABLE |===============--
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[OrderDetails](
    [order_detail_id] INT IDENTITY(1,1) NOT NULL,
    [order_id] INT NOT NULL,
    [product_id] INT NOT NULL,
    [quantity] INT NOT NULL CHECK ([quantity] > 0),
    [unit_price] DECIMAL(10,2) NOT NULL CHECK ([unit_price] > 0),
            --PK
    CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED ([order_detail_id] ASC)
        WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF),
              --FK
    CONSTRAINT [FK_OrderDetails_Orders] FOREIGN KEY ([order_id])
        REFERENCES [dbo].[Orders] ([order_id]),
    CONSTRAINT [FK_OrderDetails_Products] FOREIGN KEY ([product_id])
        REFERENCES [dbo].[Products] ([product_id])
) ON [PRIMARY]
GO
```

```
100 %
Messages
Commands completed successfully.

Completion time: 2025-09-30T10:37:03.0524789+01:00
```

Fig 1.7 Screenshot showing the SQL script used to create the Order Details Table, including primary, foreign and check constraints.

Orders Table Creation

This tables tracks customer purchases , order dates , and delivery status of an order.



```
Create Customers.sql - not connected | Create inventory.sql - not connected | Create OrderDetails.sql - not connected | Create Orders.sql - not connected

CREATE TABLE [dbo].[Orders](
    [order_id] INT IDENTITY(1,1) NOT NULL,
    [customer_id] INT NOT NULL,
    [order_date] DATETIME NOT NULL DEFAULT (GETDATE()),
    [payment_id] INT NOT NULL,
    [shipping_id] INT NOT NULL,
    [status] NVARCHAR(20) NOT NULL CHECK ([status] IN ('Pending', 'Processing', 'Shipped', 'Delivered', 'Cancelled')),
    CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED ([order_id] ASC)
            --PK
        WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF),
              --FK
    CONSTRAINT [FK_Orders_Customers] FOREIGN KEY ([customer_id])
        REFERENCES [dbo].[Customers] ([customer_id]),
    CONSTRAINT [FK_Orders_Payments] FOREIGN KEY ([payment_id])
        REFERENCES [dbo].[Payments] ([payment_id]),
    CONSTRAINT [FK_Orders_Shipping] FOREIGN KEY ([shipping_id])
        REFERENCES [dbo].[Shipping] ([shipping_id])
) ON [PRIMARY]
GO
--============================================================================
--==CREATE NONCLUSTERED INDEX--============================================
CREATE NONCLUSTERED INDEX IX_Orders__CustomerID ON dbo.Orders ([customer_id]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO

CREATE NONCLUSTERED INDEX IX_Orders__PaymentID ON dbo.Orders ([payment_id]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
100 %
Messages
```

Fig 1.8 Screenshot showing the SQL script used to create the Orders Table, including primary, foreign and check constraints.

Orders Table Alteration

This is a query that adds the employee id to the Orders table to link orders with employees.

```
ALTER TABLE Orders
ADD employee_id INT;
ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_EmployeeID FOREIGN KEY (employee_id) REFERENCES Employees(employee_id);
```

Fig 1.9 Screenshot showing the SQL script used to alter the Orders Table to add employee id column from Employees table and create a relationship through a foreign key.

Payments Table Creation

This table records payment methods, amounts, and dates for each order.

```
--===============| CREATE Payments TABLE |===============----
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Payments](
    [payment_id] INT IDENTITY(1,1) NOT NULL,
    [payment_type] NVARCHAR(50) NOT NULL,
    --PK
    CONSTRAINT [PK_Payments] PRIMARY KEY CLUSTERED ([payment_id] ASC)
        WITH (
        PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
        );
GO
--=============================================================================
--==CREATE NONCLUSTERED INDEX--=============================================
CREATE NONCLUSTERED INDEX IX_Payments_Type ON dbo.Payments ([payment_type]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
100 %
Messages
Commands completed successfully.

Completion time: 2025-09-30T09:38:24.4535780+01:00
100 %
```

Fig 2.0 Screenshot showing the SQL script used to create the Payments Table, including primary constraints.

Products Table Creation

This table stores product names, prices and supplier associations to the organisation.

```
CREATE TABLE [dbo].[Products](
    [product_id] INT IDENTITY(1,1) NOT NULL,
    [category_id] INT NOT NULL,
    [supplier_id] INT NOT NULL,
    [product_name] NVARCHAR(200) NOT NULL,
    [unit_price] DECIMAL(10,2) NOT NULL CHECK ([unit_price] > 0),
    [stock_quantity] INT NOT NULL DEFAULT (0) CHECK ([stock_quantity] >= 0),
    --PK
    CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED ([product_id] ASC)
        WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF),
              --FK
    CONSTRAINT [FK_Products_Categories] FOREIGN KEY ([category_id])
        REFERENCES [dbo].[Categories] ([category_id]),
    CONSTRAINT [FK_Products_Suppliers] FOREIGN KEY ([supplier_id])
        REFERENCES [dbo].[Suppliers] ([supplier_id])
) ON [PRIMARY]
GO
--============================================================================================
--==CREATE NONCLUSTERED INDEX ==========================----------------------------------
CREATE NONCLUSTERED INDEX [IX_Products_CategoryID] ON [dbo].[Products] ([category_id] ASC)
    WITH(
        PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON);
GO
CREATE NONCLUSTERED INDEX [IX_Products_SupplierID] ON [dbo].[Products] ([supplier_id] ASC)
    WITH(
        PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
```

Fig 2.1 Screenshot showing the SQL script used to create the Products Table, including primary, foreign and check constraints.

Reviews Table Creation

This table stores customer's feedback and ratings for the product.

```
CREATE TABLE [dbo].[Reviews](
    [review_id] INT IDENTITY(1,1) NOT NULL,
    [product_id] INT NOT NULL,
    [customer_id] INT NOT NULL,
    [rating] INT NOT NULL CHECK ([rating] BETWEEN 1 AND 5),
    [review_text] NVARCHAR(MAX) NULL,
    [review_date] DATETIME NOT NULL DEFAULT (GETDATE()),
    CONSTRAINT [PK_Reviews] PRIMARY KEY CLUSTERED ([review_id] ASC)
        WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF),
    CONSTRAINT [FK_Reviews_Products] FOREIGN KEY ([product_id])
        REFERENCES [dbo].[Products] ([product_id]),
    CONSTRAINT [FK_Reviews_Customers] FOREIGN KEY ([customer_id])
        REFERENCES [dbo].[Customers] ([customer_id])
) ON [PRIMARY]
GO


--============================================================================================
--==CREATE NONCLUSTERED INDEX--============================================================
CREATE NONCLUSTERED INDEX IX_Reviews_ProductID ON dbo.Reviews ([product_id]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
CREATE NONCLUSTERED INDEX IX_Reviews_CustomerID ON dbo.Reviews ([customer_id]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
```

100 %

Messages

Fig 2.2 Screenshot showing the SQL script used to create the Reviews Table, including primary and foreign key constraints.

Shipping Table Creation

This table stores details of delivery which consist of the type of delivery that is needed by the customer.

```
--===============| CREATE Shipping TABLE |===============--
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Shipping](
    [shipping_id] INT IDENTITY(1,1) NOT NULL,
    [delivery_option] NVARCHAR(50) NOT NULL,
    --PK
    CONSTRAINT [PK_Shipping] PRIMARY KEY CLUSTERED ([shipping_id] ASC)
        WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
) ON [PRIMARY];
GO
--=============================================================================
--==CREATE NONCLUSTERED INDEX--=============================================
CREATE NONCLUSTERED INDEX IX_Shipping_Option ON dbo.Shipping ([delivery_option]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
```

```
100 %
Messages
Commands completed successfully.

Completion time: 2025-09-30T09:44:38.0079693+01:00
```

Fig 2.3 Screenshot showing the SQL script used to create the Shipping Table, including primary key constraint.

Suppliers Table Creation

This table is used to store supplier contact details and identifiers.

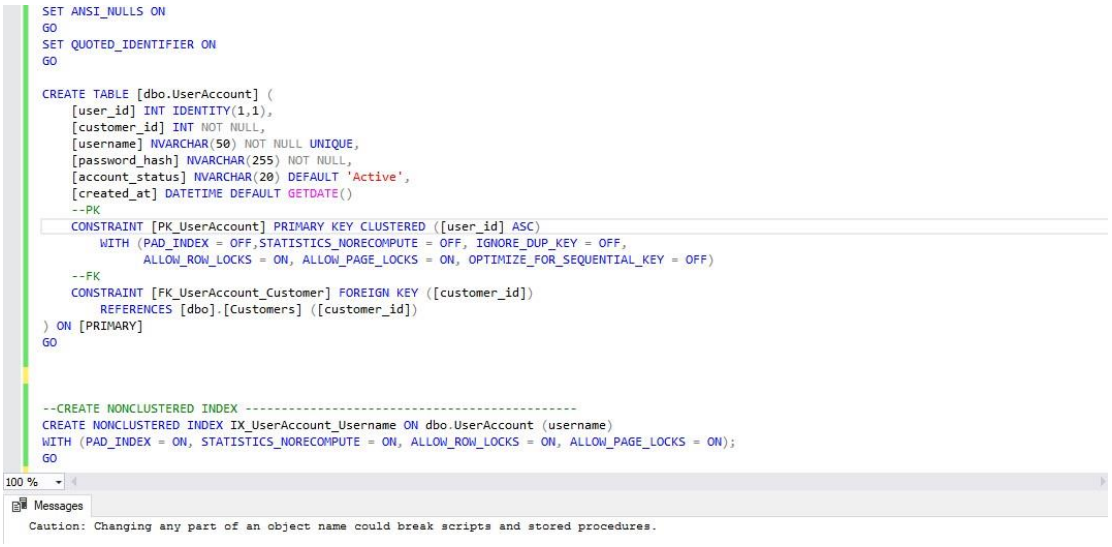
```
------------------------------------------------------------
--===============| CREATE Suppliers TABLE |===============--
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Suppliers](
    [supplier_id] INT IDENTITY(1,1) NOT NULL,
    [supplier_name] NVARCHAR(200) NOT NULL,
    [contact_email] NVARCHAR(150) NULL,
    --PK
    CONSTRAINT [PK_Suppliers] PRIMARY KEY CLUSTERED ([supplier_id] ASC)
        WITH (PAD_INDEX = OFF,
              STATISTICS_NORECOMPUTE = OFF,
              IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON,
              ALLOW_PAGE_LOCKS = ON,
              OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
) ON [PRIMARY]
GO
--=============================================================================
--==CREATE NONCLUSTERED INDEX--=============================================
CREATE NONCLUSTERED INDEX IX_Suppliers_Email ON dbo.Suppliers ([contact_email]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
        DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
        GO
```

```
100 %
Messages
Commands completed successfully.
```

Fig 2.4 Screenshot showing the SQL script used to create the Suppliers Table, including primary key constraint.

User Account Table Creation

This table stores every login details about a customer and comprises of the username and password of the customer.



```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo.UserAccount] (
    [user_id] INT IDENTITY(1,1),
    [customer_id] INT NOT NULL,
    [username] NVARCHAR(50) NOT NULL UNIQUE,
    [password_hash] NVARCHAR(255) NOT NULL,
    [account_status] NVARCHAR(20) DEFAULT 'Active',
    [created_at] DATETIME DEFAULT GETDATE()
    --PK
    CONSTRAINT [PK_UserAccount] PRIMARY KEY CLUSTERED ([user_id] ASC)
        WITH (PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
    --FK
    CONSTRAINT [FK_UserAccount_Customer] FOREIGN KEY ([customer_id])
        REFERENCES [dbo].[Customers] ([customer_id])
) ON [PRIMARY]
GO



--CREATE NONCLUSTERED INDEX -----------------------------------------
CREATE NONCLUSTERED INDEX IX_UserAccount_Username ON dbo.UserAccount (username)
WITH (PAD_INDEX = ON, STATISTICS_NORECOMPUTE = ON, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON);
GO
```

100 %

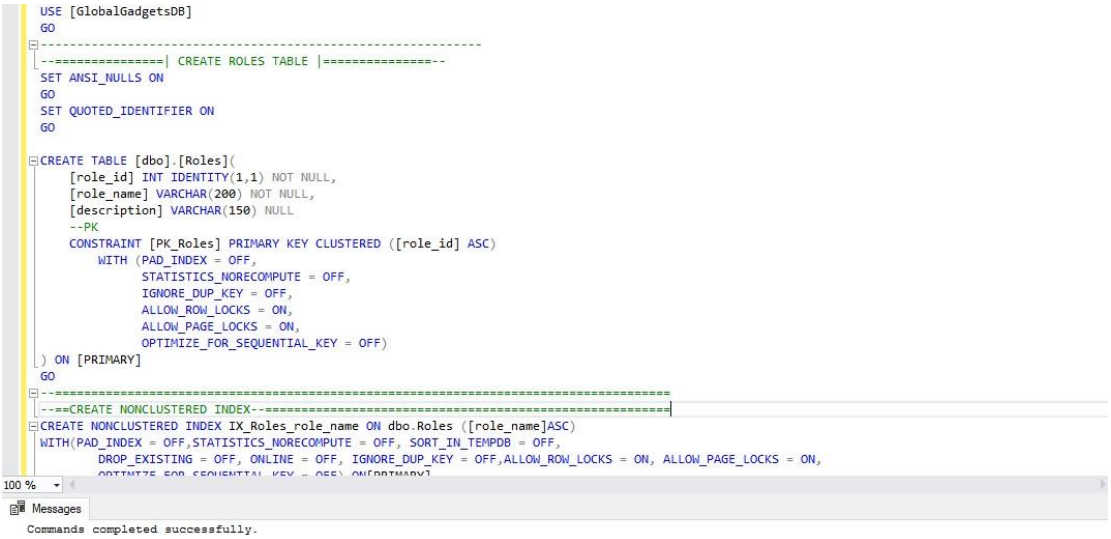
Messages

```
Caution: Changing any part of an object name could break scripts and stored procedures.
```

Fig 2.5 Screenshot showing the SQL script used to create the User Account Table, including primary, foreign and check constraints.

Roles Table Creation

This table specifies the job titles and descriptions for employees.



```
USE [GlobalGadgetsDB]
GO
------------------------------------------------------------
--===============| CREATE ROLES TABLE |==============--
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Roles](
    [role_id] INT IDENTITY(1,1) NOT NULL,
    [role_name] VARCHAR(200) NOT NULL,
    [description] VARCHAR(150) NULL
    --PK
    CONSTRAINT [PK_Roles] PRIMARY KEY CLUSTERED ([role_id] ASC)
        WITH (PAD_INDEX = OFF,
              STATISTICS_NORECOMPUTE = OFF,
              IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON,
              ALLOW_PAGE_LOCKS = ON,
              OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
) ON [PRIMARY]
GO
--============================================================================
--==CREATE NONCLUSTERED INDEX--=============================================
CREATE NONCLUSTERED INDEX IX_Roles_role_name ON dbo.Roles ([role_name]ASC)
WITH(PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
       DROP_EXISTING = OFF, ONLINE = OFF, IGNORE_DUP_KEY = OFF,ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
       OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON[PRIMARY]
```

100 %

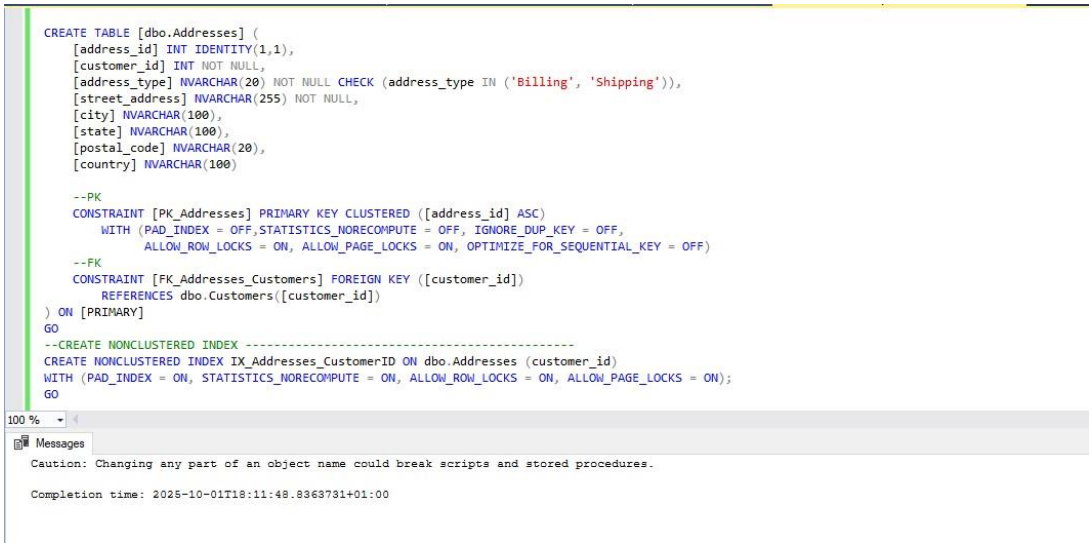
Messages

```
Commands completed successfully.
```

Fig 2.5 Screenshot showing the SQL script used to create the Roles Table, including primary constraint.

Addresses Table Creation

This table show the address details of the customer and their preferable billing type.



```
CREATE TABLE [dbo.Addresses] (
    [address_id] INT IDENTITY(1,1),
    [customer_id] INT NOT NULL,
    [address_type] NVARCHAR(20) NOT NULL CHECK (address_type IN ('Billing', 'Shipping')),
    [street_address] NVARCHAR(255) NOT NULL,
    [city] NVARCHAR(100),
    [state] NVARCHAR(100),
    [postal_code] NVARCHAR(20),
    [country] NVARCHAR(100)

    --PK
    CONSTRAINT [PK_Addresses] PRIMARY KEY CLUSTERED ([address_id] ASC)
        WITH (PAD_INDEX = OFF,STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
              ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
    --FK
    CONSTRAINT [FK_Addresses_Customers] FOREIGN KEY ([customer_id])
        REFERENCES dbo.Customers([customer_id])
) ON [PRIMARY]
GO
--CREATE NONCLUSTERED INDEX -----------------------------------------------
CREATE NONCLUSTERED INDEX IX_Addresses_CustomerID ON dbo.Addresses (customer_id)
WITH (PAD_INDEX = ON, STATISTICS_NORECOMPUTE = ON, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON);
GO
```

```
100 %
Messages
  Caution: Changing any part of an object name could break scripts and stored procedures.

  Completion time: 2025-10-01T18:11:48.8363731+01:00
```

Fig 2.6 Screenshot showing the SQL script used to create the Address Table, including primary , foreign and check constraint.

2.4 DATA POPULATION

This section demonstrates the population of tables within the retailer's database using sample records. The data inserted using the **INSERT INTO** syntax reflects realistic entries for roles, employees, products, suppliers, customers, and orders. These records are essential for testing the integrity of relationships, constraints, and stored procedures defined in the system.

This insertion are shown below:

Addresses Table Insertion

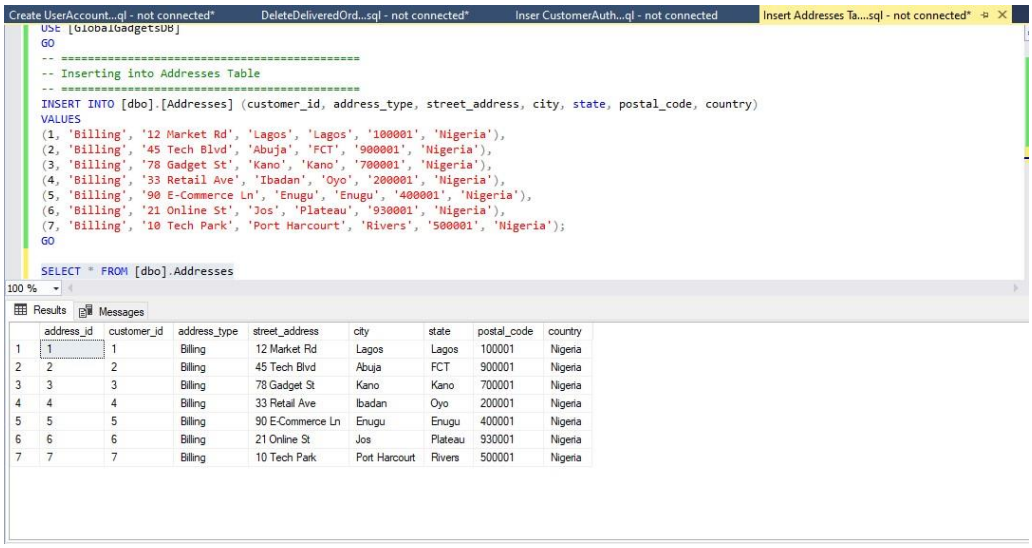
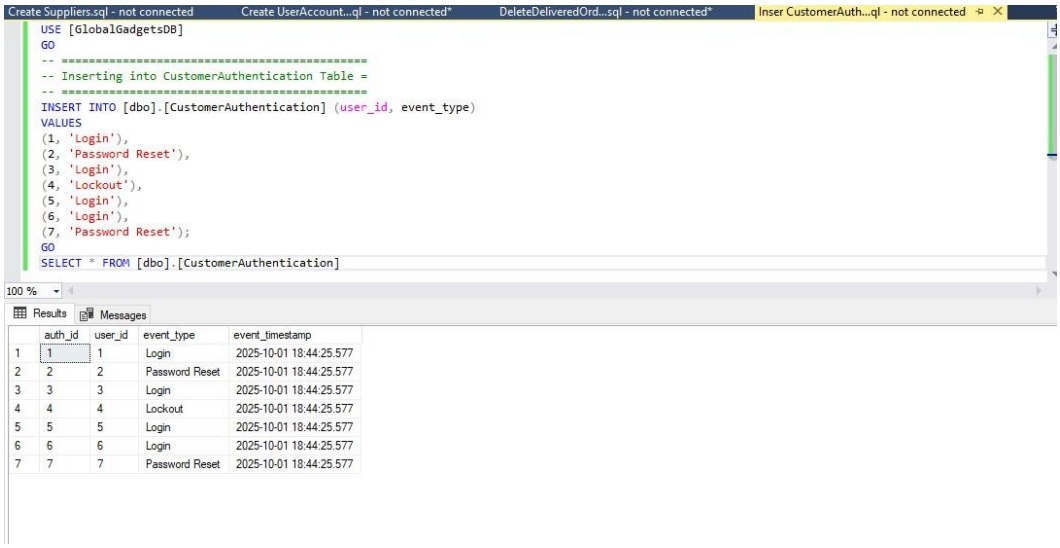


Fig 2.7 Screenshot showing the insertion of Addresses records into the Addresses Table. It consist of 7 records.

Categories Table Insertion

Fig 2.8 Screenshot showing the insertion of Categories records into the Categories Table. It consist of 7 records.

Customer Authentication Table Insertion



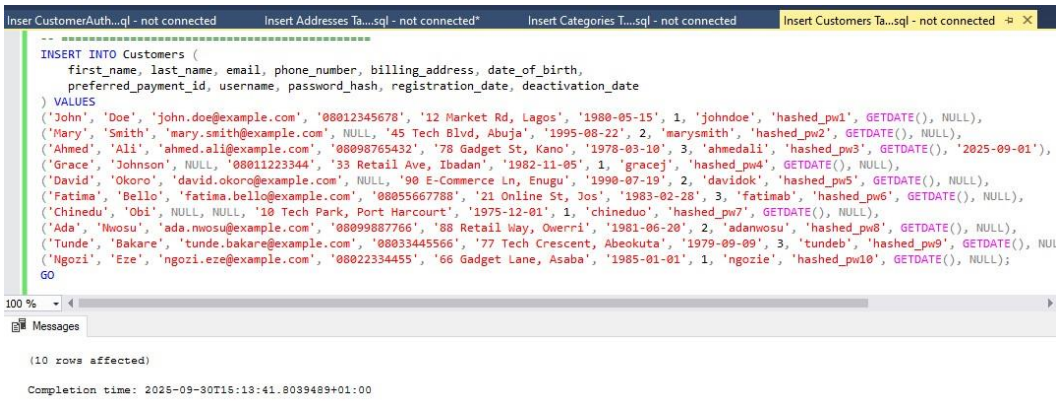
```
USE [GlobalGadgetsDB]
GO
-- ============================================
-- Inserting into CustomerAuthentication Table =
-- ============================================
INSERT INTO [dbo].[CustomerAuthentication] (user_id, event_type)
VALUES
(1, 'Login'),
(2, 'Password Reset'),
(3, 'Login'),
(4, 'Lockout'),
(5, 'Login'),
(6, 'Login'),
(7, 'Password Reset');
GO
SELECT * FROM [dbo].[CustomerAuthentication]
```

Results | Messages

	auth_id	user_id	event_type	event_timestamp
1	1	1	Login	2025-10-01 18:44:25.577
2	2	2	Password Reset	2025-10-01 18:44:25.577
3	3	3	Login	2025-10-01 18:44:25.577
4	4	4	Lockout	2025-10-01 18:44:25.577
5	5	5	Login	2025-10-01 18:44:25.577
6	6	6	Login	2025-10-01 18:44:25.577
7	7	7	Password Reset	2025-10-01 18:44:25.577

Fig 2.9 Screenshot showing the insertion of Customer Authentication records into the Customer Authentication Table. It consist of 7 records.

Customers Table Insertion



```
-- ============================================
INSERT INTO Customers (
    first_name, last_name, email, phone_number, billing_address, date_of_birth,
    preferred_payment_id, username, password_hash, registration_date, deactivation_date
) VALUES
('John', 'Doe', 'john.doe@example.com', '08012345678', '12 Market Rd, Lagos', '1980-05-15', 1, 'johndoe', 'hashed_pw1', GETDATE(), NULL),
('Mary', 'Smith', 'mary.smith@example.com', NULL, '45 Tech Blvd, Abuja', '1995-08-22', 2, 'marysmith', 'hashed_pw2', GETDATE(), NULL),
('Ahmed', 'Ali', 'ahmed.ali@example.com', '08098765432', '78 Gadget St, Kano', '1978-03-10', 3, 'ahmedali', 'hashed_pw3', GETDATE(), '2025-09-01'),
('Grace', 'Johnson', NULL, '08011223344', '33 Retail Ave, Ibadan', '1982-11-05', 1, 'gracej', 'hashed_pw4', GETDATE(), NULL),
('David', 'Okoro', 'david.okoro@example.com', NULL, '90 E-Commerce Ln, Enugu', '1990-07-19', 2, 'davidok', 'hashed_pw5', GETDATE(), NULL),
('Fatima', 'Bello', 'fatima.bello@example.com', '08055667788', '21 Online St, Jos', '1983-02-28', 3, 'fatimab', 'hashed_pw6', GETDATE(), NULL),
('Chinedu', 'Obi', NULL, NULL, '10 Tech Park, Port Harcourt', '1975-12-01', 1, 'chineduo', 'hashed_pw7', GETDATE(), NULL),
('Ada', 'Nwosu', 'ada.nwosu@example.com', '08099887766', '88 Retail Way, Owerri', '1981-06-20', 2, 'adanwosu', 'hashed_pw8', GETDATE(), NULL),
('Tunde', 'Bakare', 'tunde.bakare@example.com', '08033445566', '77 Tech Crescent, Abeokuta', '1979-09-09', 3, 'tundeb', 'hashed_pw9', GETDATE(), NUL
('Ngozi', 'Eze', 'ngozi.eze@example.com', '08022334455', '66 Gadget Lane, Asaba', '1985-01-01', 1, 'ngozie', 'hashed_pw10', GETDATE(), NULL);
GO
```

Messages

```
(10 rows affected)

Completion time: 2025-09-30T15:13:41.8039489+01:00
```

Fig 3.1 Screenshot showing the insertion of Customer records into the Customer Table. It consists of 10 records.

Inventory Table Insertion

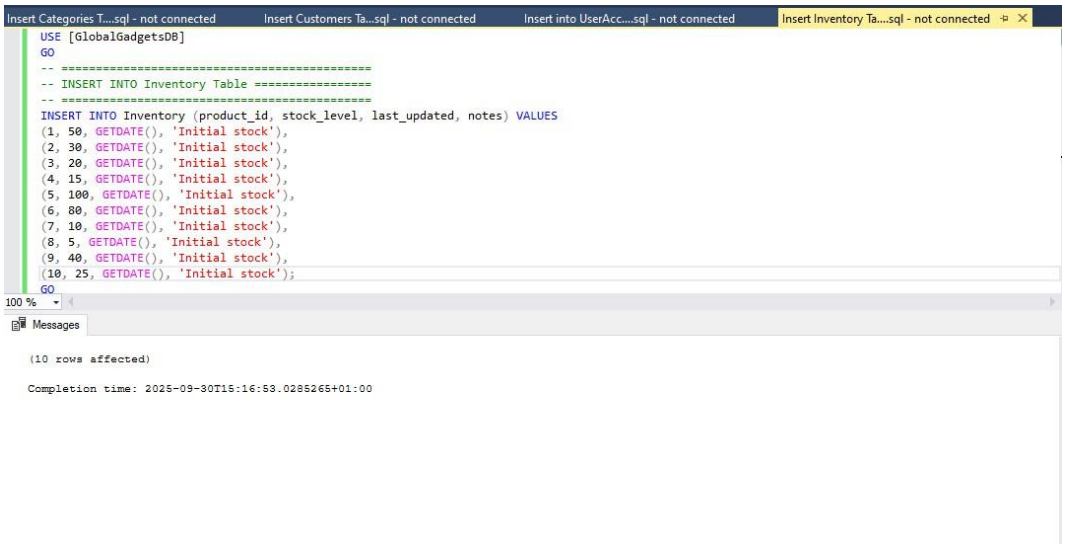


Fig 3.2 Screenshot showing the insertion of Inventory records into the Inventory Table. It consists of 10 records.

Order Details Table Insertion

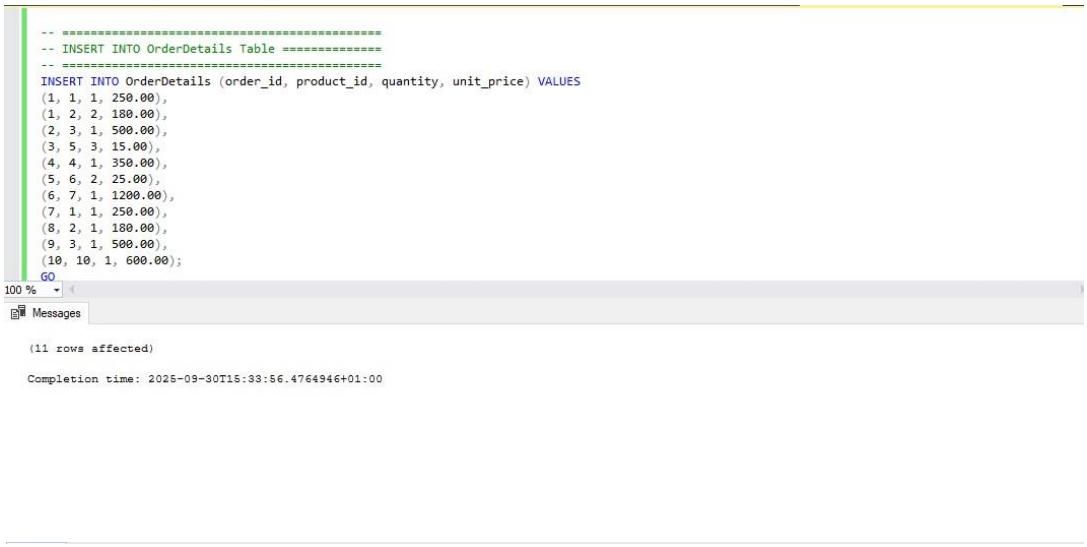


Fig 3.3 Screenshot showing the insertion of Order Details records into the Order Details Table. It consists of 10 records.

Orders Table Insertion

```
USE [GlobalGadgetsDB]
GO
-- ============================================
-- INSERT INTO Orders Table ==================
-- ============================================
INSERT INTO Orders (customer_id, order_date, payment_id, shipping_id, status) VALUES
(1, GETDATE(), 1, 1, 'Delivered'),
(2, GETDATE(), 2, 2, 'Processing'),
(3, GETDATE(), 3, 1, 'Cancelled'),
(4, DATEADD(DAY, -2, GETDATE()), 1, 2, 'Delivered'),
(5, DATEADD(DAY, -1, GETDATE()), 2, 1, 'Shipped'),
(6, GETDATE(), 3, 2, 'Pending'),
(7, GETDATE(), 1, 1, 'Delivered'),
(8, GETDATE(), 2, 2, 'Delivered'),
(9, GETDATE(), 3, 1, 'Cancelled'),
(10, GETDATE(), 1, 1, 'Processing');
GO
```

100 %

Messages

```
(10 rows affected)

Completion time: 2025-09-30T15:29:29.3006068+01:00
```

Fig 3.4 Screenshot showing the insertion of Orders records into the Orders Table. It consists of 10 records.

Payments Table Insertion

```
USE [GlobalGadgetsDB]
GO

-- ============================================
-- INSERT INTO Payments Table-------------------------
-- ============================================
INSERT INTO Payments (payment_type) VALUES
('Credit Card'), --1
('PayPal'), --2
('Bank Transfer'), --3
('Mobile Money'), --4
('Cash on Delivery'), --5
('Cryptocurrency'), --6
('Gift Card'); --7
GO
```

100 %

Messages

```
(7 rows affected)

Completion time: 2025-09-30T14:41:23.1027387+01:00
```

Fig 3.5 Screenshot showing the insertion of Payment type records into the Payments Table. It consists of 7 records.

Products Table Insertion

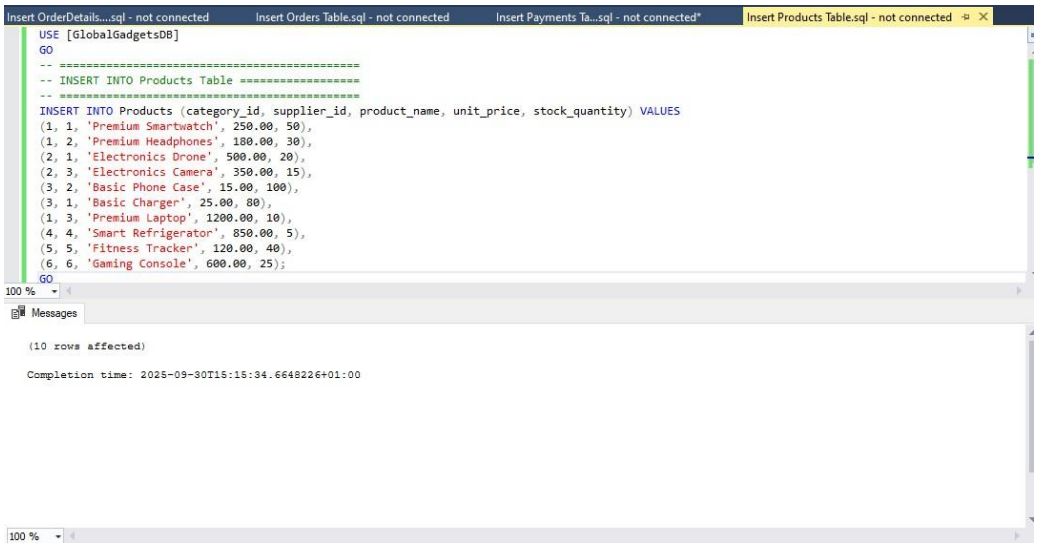


Fig 3.6 Screenshot showing the insertion of Products records into the Products Table. It consists of 10 records.

Reviews Table Insertion

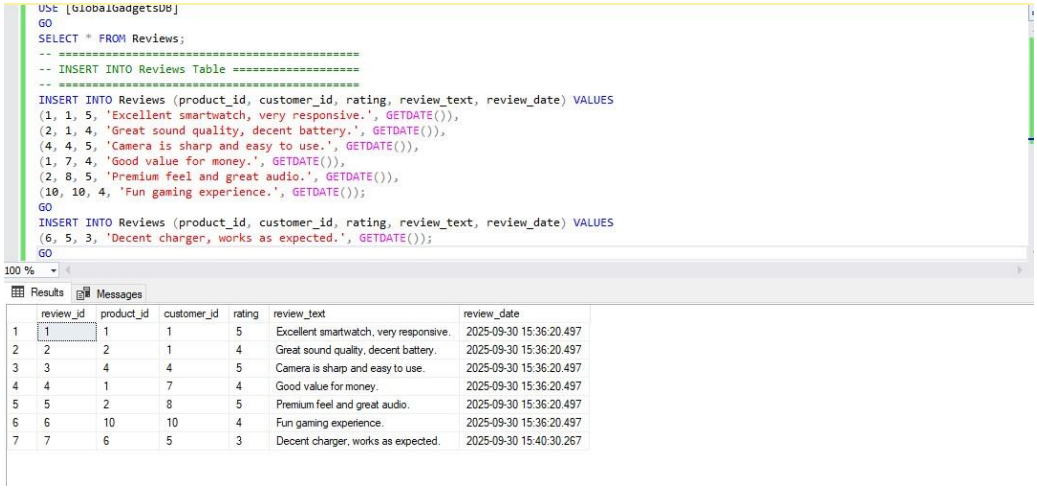


Fig 3.7 Screenshot showing the insertion of Reviews records into the Review Table. It consists of 7 records.

Roles Table Insertion

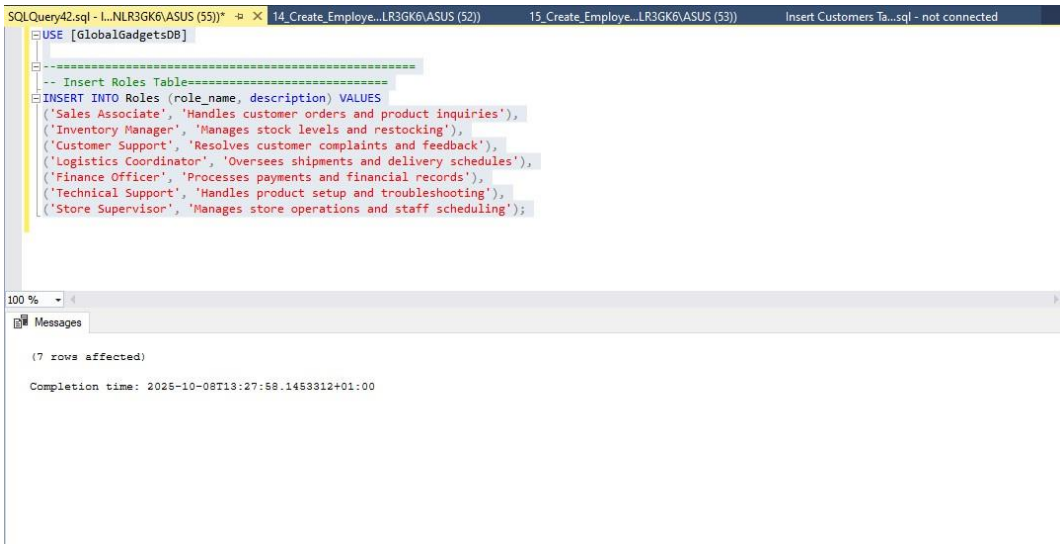


Fig 3.8 Screenshot showing the insertion of Roles of Employees records into the Roles Table. It consists of 7 records.

Shipping Table Insertion

Fig 3.9 Screenshot showing the insertion of Shipping records into the Shipping Table. It consists of 7 records.

Suppliers Table Insertion

```
USE [GlobalGadgetsDB]
GO

-- ============================================
-- INSERT INTO Suppliers Table ================
-- ============================================
INSERT INTO Suppliers (supplier_name, contact_email) VALUES
('TechNova Ltd.', 'contact@technova.com'),
('GadgetWorld Inc.', 'sales@gadgetworld.com'),
('SmartEdge Co.', 'info@smartedge.com'),
('ElectroHub', 'support@electrohub.com'),
('NovaParts', 'hello@novaparts.com'),
('DigitalCore', 'admin@digitalcore.com'),
('NextGen Supplies', 'orders@nextgen.com');
GO
```

```
100 %
Messages

(7 rows affected)

Completion time: 2025-09-30T15:11:19.3510780+01:00
```

Fig 4.1 Screenshot showing the insertion of Supplier records into the Suppliers Table. It consists of 7 records.

User Account Table Insertion

```
USE [GlobalGadgetsDB]
GO

-- ============================================
-- Inserting into UserAccount Table
-- ============================================
INSERT INTO [dbo].[UserAccount] (customer_id, username, password_hash)
VALUES
(1, 'johndoe', 'hashed_pw1'),
(2, 'marysmith', 'hashed_pw2'),
(3, 'ahmedali', 'hashed_pw3'),
(4, 'gracej', 'hashed_pw4'),
(5, 'davidok', 'hashed_pw5'),
(6, 'fatimab', 'hashed_pw6'),
(7, 'chineduo', 'hashed_pw7');
GO
```

```
100 %
Messages

(7 rows affected)

Completion time: 2025-10-01T18:32:01.4344031+01:00
```

Fig 4.2 Screenshot showing the insertion of User Login records into the User Account Table. It consists of 7 records.

Employees Table Insertion

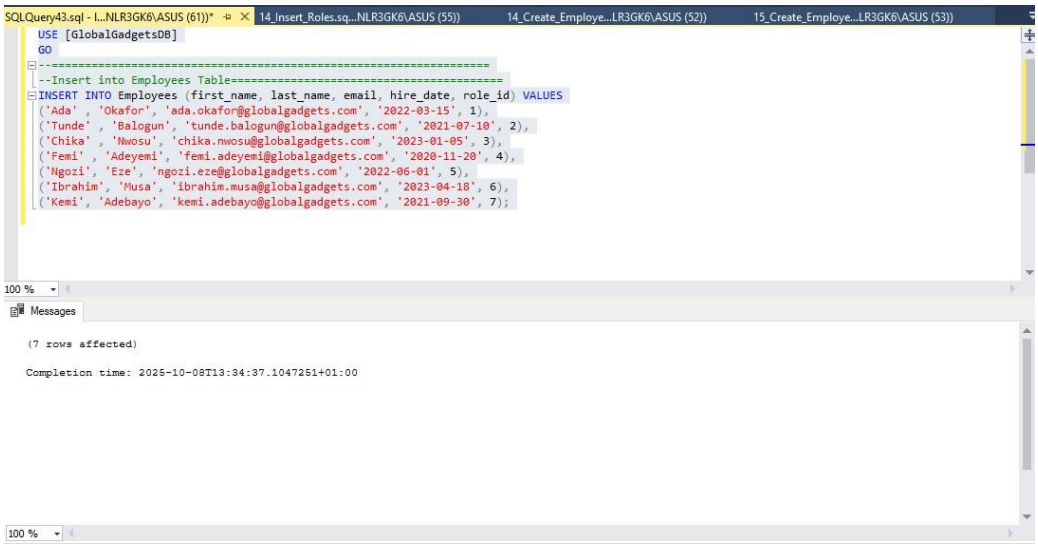


Fig 4.3 Screenshot showing the insertion of Employees records into the Employees Table. It consists of 7 records.

Section 3: T-SQL FUNCTIONAL IMPLEMENTATION

3.1 CONSTRAINT IMPLEMENTATION (Requirement 2)

Constraints were applied to enforce data integrity and business rules across the schema:

- Primary Keys: This uniquely identify records in each table (e.g., productid, orderid)
- Foreign Keys: This maintain referential integrity between related tables (e.g., supplier_id in Products)
- CHECK Constraints: This is to validate values such as payment type (Cash, Card, Transfer) and status of the orders (Pending, Shipped, Cancelled)
- NOT NULL Constraints: This is to ensure essential fields like product_name, customer_email, and order_date are always provided

A constraint was added **to check that the product price is always a positive value** which means it must always be greater than zero as required and it was executed perfectly below:

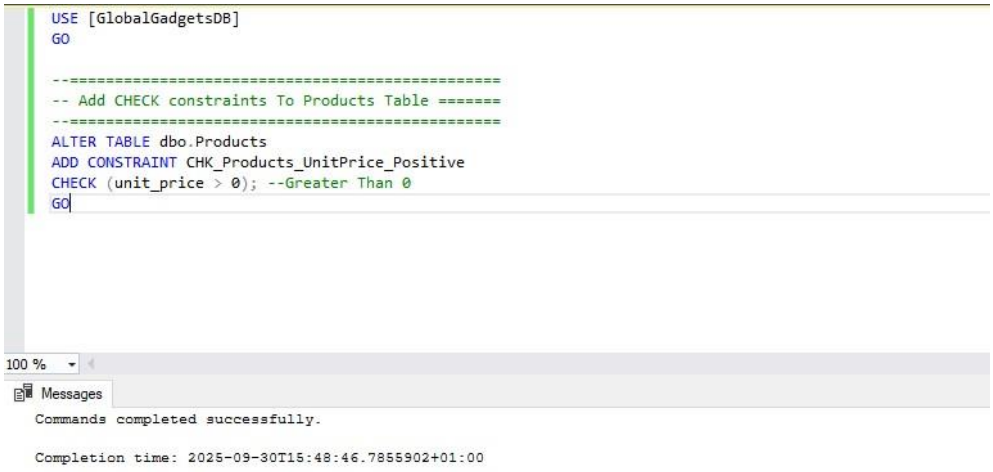


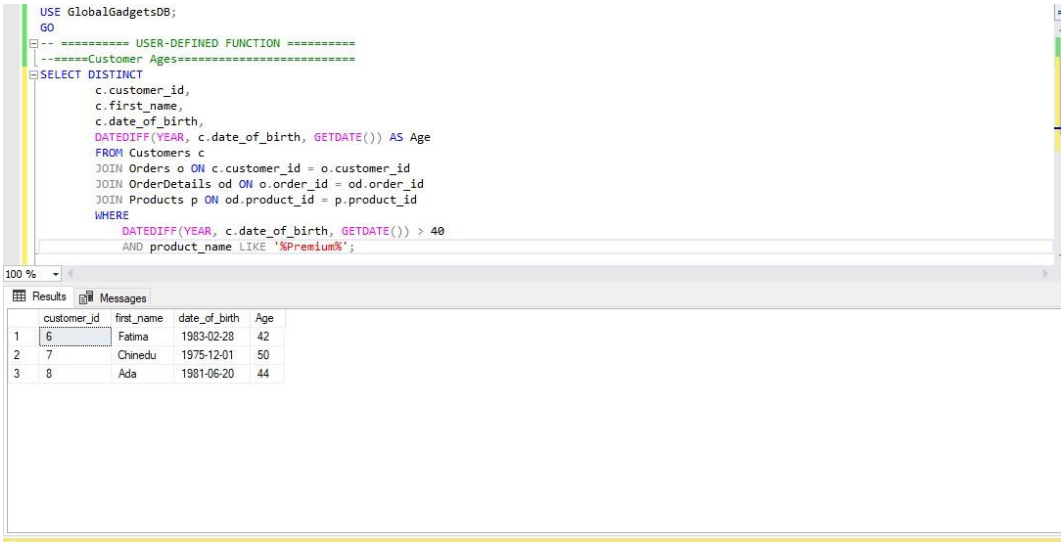
Fig 4.4 Screenshot of adding check constraint to the Products Table.

These constraints prevent invalid data entry and ensure consistent relationships between entities.

3.2 DATA ANALYSIS QUERY 1 (Requirement 3)

This says we should be a be able to list all the **Customers** who are older than 40 and have placed and order for a product in the 'Premium' product category.

We solved this below:



```
USE GlobalGadgetsDB;
GO
-- ========== USER-DEFINED FUNCTION ==========
--=====Customer Ages==========================
SELECT DISTINCT
        c.customer_id,
        c.first_name,
        c.date_of_birth,
        DATEDIFF(YEAR, c.date_of_birth, GETDATE()) AS Age
        FROM Customers c
        JOIN Orders o ON c.customer_id = o.customer_id
        JOIN OrderDetails od ON o.order_id = od.order_id
        JOIN Products p ON od.product_id = p.product_id
        WHERE
            DATEDIFF(YEAR, c.date_of_birth, GETDATE()) > 40
            AND product_name LIKE '%Premium%';
```

100 %

Results | Messages

	customer_id	first_name	date_of_birth	Age
1	6	Fatima	1983-02-28	42
2	7	Chinedu	1975-12-01	50
3	8	Ada	1981-06-20	44

Fig4.5 Screenshot showing the list of customers who are above 40 and have placed and order for a premium product.

3.3 STORED PROCEDURE AND USER DEFINED FUNCTIONS (Requirement 4)

These are predefined blocks of SQL code that perform tasks. It can be executed with a single command than repeating the same code multiple time. The total number of 4 stored procedures were created and they are as follows:

SEARCH PRODUCTS BY NAME

This stored procedure allows the retailer to search for products by name using a character string. It returns matching products along with their order details, sorted by the most recent order date first. The Process is shown below:

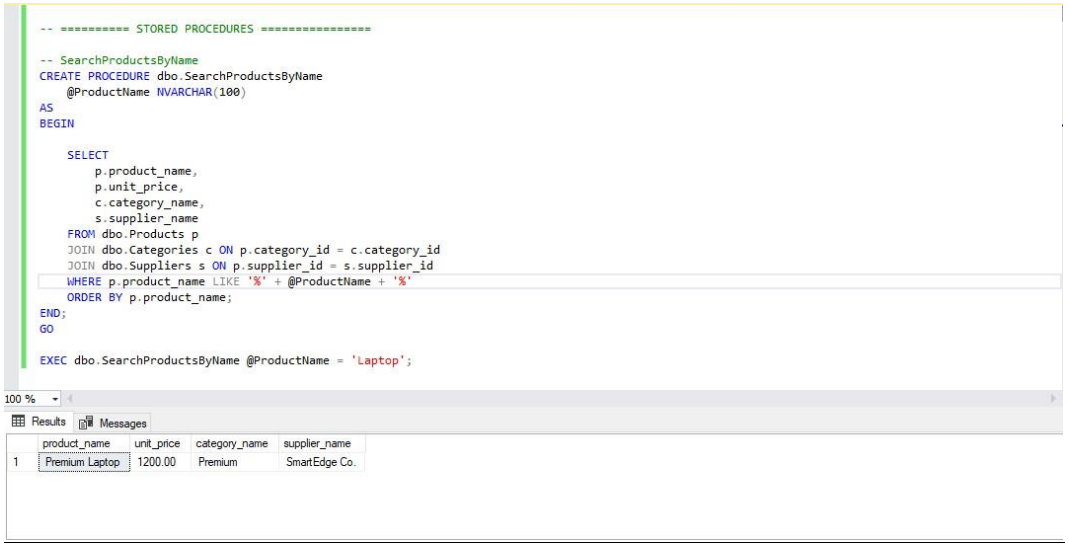


Fig4.6 Screenshot show the procedure for Search Products By Name.

GET TODAY ORDERS BY CUSTOMERS

This procedure retrieves a full list of products ordered by a specific customer on the current system date. It also includes supplier information for each product. It is useful for tracking daily customer activity and verifying supplier to product relationships for same day orders.

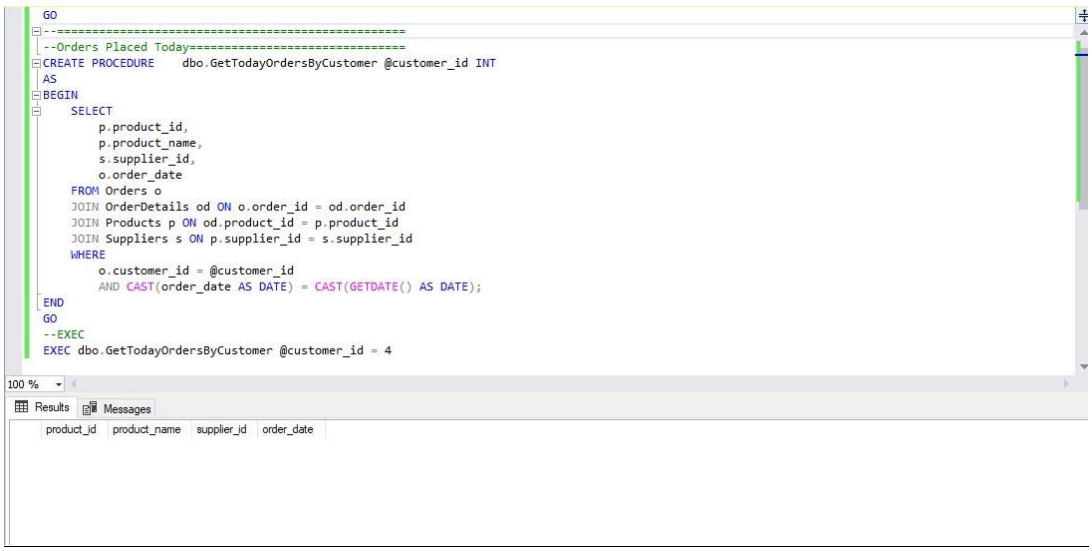


Fig 4.7 Screenshot showing the procedure for getting orders by customer and shipping details.

UPDATE SUPPLIER DETAILS

This procedure helps the retailer to update the details of an existing supplier, including name and email. It ensures the supplier records are exact and up to date for communication and other purposes. The Execution of the procedure is shown below:

```
-- UpdateSupplierDetails
CREATE PROCEDURE dbo.UpdateSupplierDetails
    @SupplierID INT,
    @NewName NVARCHAR(100),
    @NewEmail NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE dbo.Suppliers
    SET supplier_name = @NewName,
        contact_email = @NewEmail
    WHERE supplier_id = @SupplierID;
END;
GO

SELECT * FROM Suppliers
EXEC dbo.UpdateSupplierDetails
    @SupplierID = 2,
    @NewName = 'TechSource Ltd',
    @NewEmail = 'contact@techsource.com';
```

100 %

Results | Messages

	supplier_id	supplier_name	contact_email
1	1	TechNova Ltd.	contact@technova.com
2	2	TechSource Ltd	contact@techsource.com
3	3	SmartEdge Co.	info@smartedge.com
4	4	ElectroHub	support@electrohub.com
5	5	NovaParts	hello@novaparts.com

Fig4.8 Screenshot showing the procedure to update supplier details

DELETE DELIVERED ORDER

This procedure deletes an order from the database only if its order status is termed 'Delivered'. It helps to maintain a clean order history by removing completed transactions that no longer need to be retained.

```
--====================================
-- DeleteDeliveredOrder===============
CREATE PROCEDURE dbo.DeleteDeliveredOrder
    @OrderID INT
AS
BEGIN
    DELETE FROM dbo.Orders
    WHERE order_id = @OrderID
      AND status = 'Delivered';
END;
GO

EXEC dbo.DeleteDeliveredOrder @OrderID = 1;
```

100 %

Results | Messages

	order_id	customer_id	order_date	payment_id	shipping_id	status
1	2	2	2025-09-30 15:29:27.773	2	2	Processing
2	3	3	2025-09-30 15:29:27.773	3	1	Cancelled
3	4	4	2025-09-28 15:29:27.773	1	2	Delivered
4	5	5	2025-09-29 15:29:27.773	2	1	Shipped
5	6	6	2025-09-30 15:29:27.773	3	2	Pending
6	7	7	2025-09-30 15:29:27.773	1	1	Delivered
7	8	8	2025-09-30 15:29:27.773	2	2	Delivered
8	9	9	2025-09-30 15:29:27.773	3	1	Cancelled
9	10	10	2025-09-30 15:29:27.773	1	1	Processing

Fig4.9 Screenshot showing the procedure of the order deleting when order id = 1.

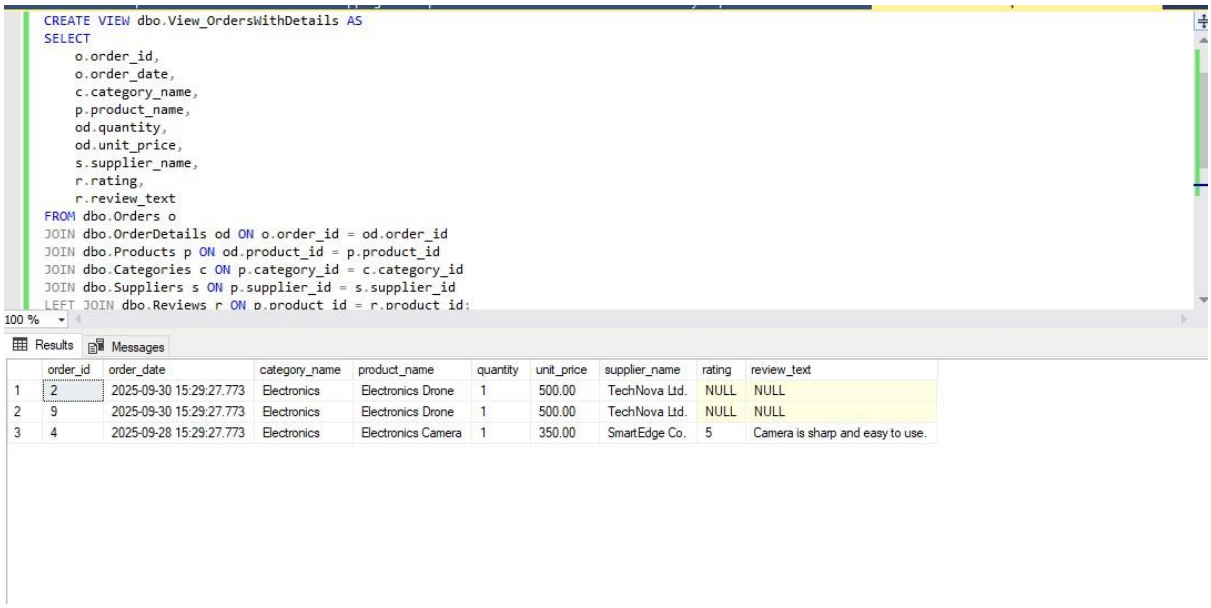
These functions and procedures encapsulate reusable logic and simplify complex operations.

3.4 VIEW CREATION (Requirement 5)

These are virtual tables based on the result of a query done in SQL. They make a way to access complex queries easier to get by. A view was created and it is as follows:

ORDERS WITH DETAILS

This **View** shows all previous and current orders for all customers, and including details of the category, the supplier name and any associated Review/Rating given for a product as required.



```
CREATE VIEW dbo.View_OrdersWithDetails AS
SELECT
    o.order_id,
    o.order_date,
    c.category_name,
    p.product_name,
    od.quantity,
    od.unit_price,
    s.supplier_name,
    r.rating,
    r.review_text
FROM dbo.Orders o
JOIN dbo.OrderDetails od ON o.order_id = od.order_id
JOIN dbo.Products p ON od.product_id = p.product_id
JOIN dbo.Categories c ON p.category_id = c.category_id
JOIN dbo.Suppliers s ON p.supplier_id = s.supplier_id
LEFT JOIN dbo.Reviews r ON p.product_id = r.product_id;
```

100 %

Results | Messages

	order_id	order_date	category_name	product_name	quantity	unit_price	supplier_name	rating	review_text
1	2	2025-09-30 15:29:27.773	Electronics	Electronics Drone	1	500.00	TechNova Ltd.	NULL	NULL
2	9	2025-09-30 15:29:27.773	Electronics	Electronics Drone	1	500.00	TechNova Ltd.	NULL	NULL
3	4	2025-09-28 15:29:27.773	Electronics	Electronics Camera	1	350.00	SmartEdge Co.	5	Camera is sharp and easy to use.

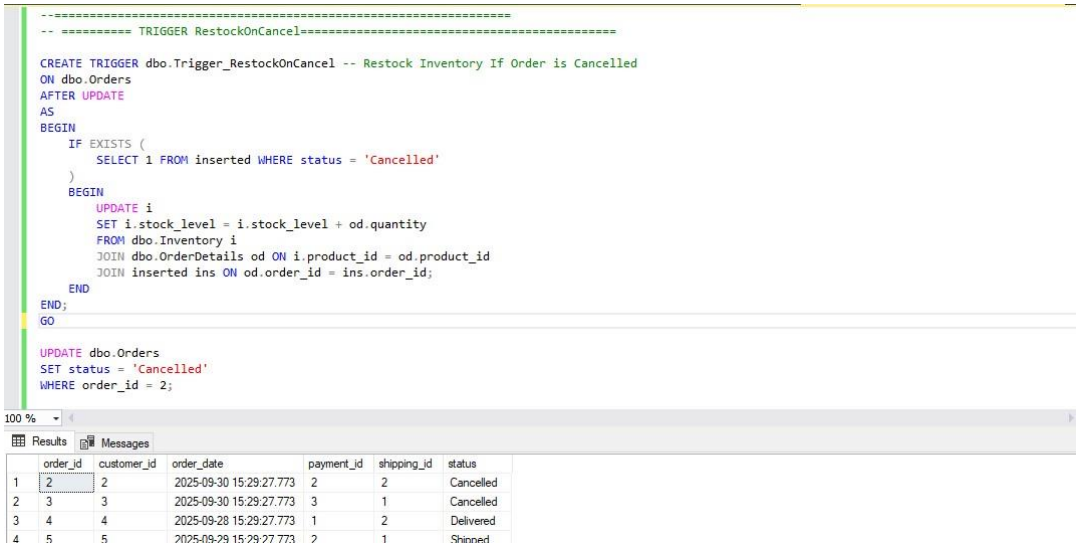
Fig 5.1 Screenshot showing the view that shows all previous and current orders.

3.5 TRIGGER IMPLEMENTATION (Requirement 6)

Triggers were implemented to automate business logic. They are automatic actions that run in response to events like **INSERT, UPDATE** OR **DELETE** on a table. The total number of 2 trigger cases were created and they are as follows:

TRIGGER RESTOCK ON CANCEL

This is a trigger called RestockOnCancel which shows that if the order status is labelled cancelled, the inventory gets restocked automatically and I confirmed by changing the status where order id is 2 to canceled from initial processing then the stock level went back to normal. The process and execution is shown below:



```
--=================================================================
-- ========== TRIGGER RestockOnCancel=================================================

CREATE TRIGGER dbo.Trigger_RestockOnCancel -- Restock Inventory If Order is Cancelled
ON dbo.Orders
AFTER UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1 FROM inserted WHERE status = 'Cancelled'
    )
    BEGIN
        UPDATE i
        SET i.stock_level = i.stock_level + od.quantity
        FROM dbo.Inventory i
        JOIN dbo.OrderDetails od ON i.product_id = od.product_id
        JOIN inserted ins ON od.order_id = ins.order_id;
    END
END;
GO

UPDATE dbo.Orders
SET status = 'Cancelled'
WHERE order_id = 2;
```

100 %

Results | Messages

	order_id	customer_id	order_date	payment_id	shipping_id	status
1	2	2	2025-09-30 15:29:27.773	2	2	Cancelled
2	3	3	2025-09-30 15:29:27.773	3	1	Cancelled
3	4	4	2025-09-28 15:29:27.773	1	2	Delivered
4	5	5	2025-09-29 15:29:27.773	2	1	Shipped

Fig5.2 Screenshot showing triggers 'Restock On Cancel' and how it works.

TRIGGER_PREVENT NEGATIVE STOCK

This is a trigger process called PreventNegativeStock. This Prevents stock levels from dropping below zero and I confirmed it. When I set the stock level to be negative, it didn't do successfully but when I set the stock level to a positive number, It updated which confirms it works. The process and execution are shown below:

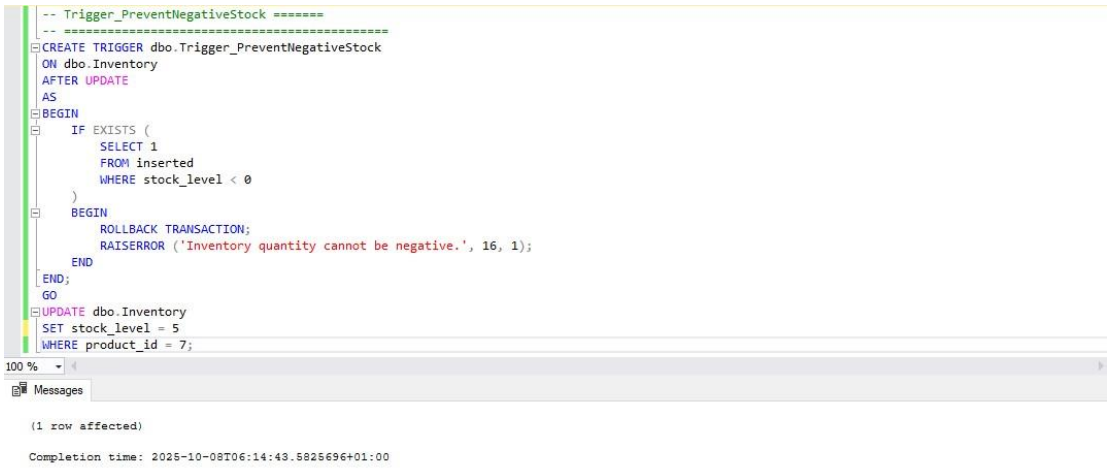


Fig 5.3 Screenshot showing triggers 'Prevent Negative stock' and how it works.

These triggers ensure real-time enforcement of rules and reduce manual intervention.

3.6 DATA ANALYSIS QUERY 2 (Requirement 7)

This sample query below allows the retailer to identify the number of 'Delivered Orders' with the category Electronics as required.

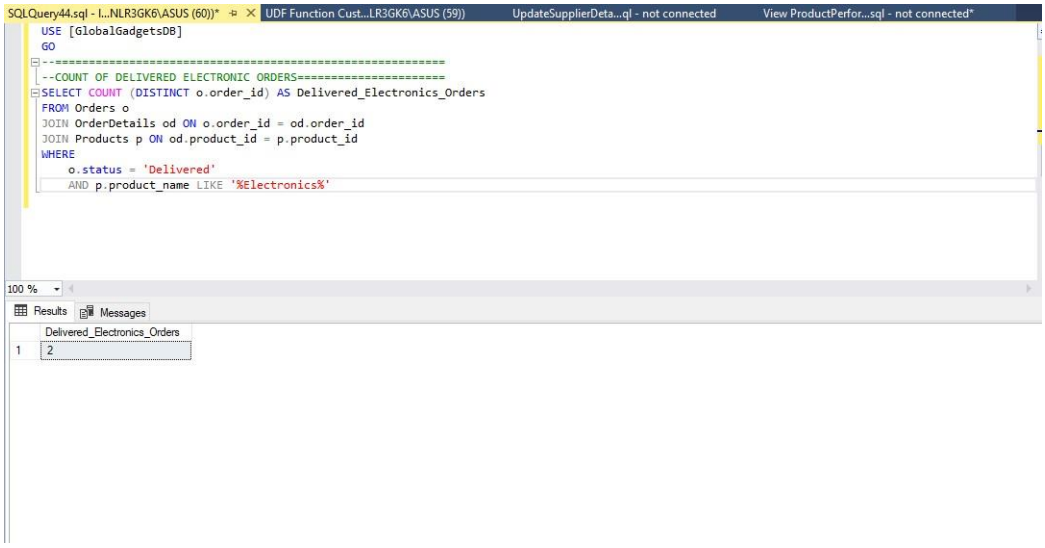


Fig5.4 Screenshot showing the query that identifies number of delivered electronic orders.

Section 4: STRATEGIC DATABASE ADVICE AND GUIDANCE

4.1 DATA INTEGRITY AND CONCURRENCY

GlobalGadgetsDB enforces data integrity through a combination of constraints, triggers, and transaction control:

- Constraints: Primary keys, foreign keys, and CHECK constraints prevent invalid or orphaned data.
- Stored Procedures: This makes the queries in the database faster and more efficient.By executing the stored procedure, queries won't be re-written from scratch.
- Triggers: Automatically enforce business rules (e.g., restocking on cancellation, preventing negative stock).

These mechanisms ensure that multiple users can interact with the database without compromising accuracy or consistency.

4.2 DATABASE SECURITY

Security measures were considered to protect sensitive data and restrict unauthorized access:

- Role-Based Access Control (RBAC): Employees are assigned roles that determine their access level
- Authentication: SQL Server authentication is used to manage user logins and permissions
- Data Masking: Sensitive fields such as customer emails and payment details can be masked in views or queries
- Audit Logging: Triggers log inventory changes and order cancellations to maintain traceability
- Backup Encryption: Backup files can be encrypted to prevent unauthorized restoration

These practices help safeguard customer data, financial records, and operational integrity.

4.3 DATABASE BACKUP AND RECOVERY

A backup and recovery strategy was implemented to ensure business continuity:

- Full Backups: Scheduled daily using SQL Server Agent or manual BACKUP DATABASE commands
- Differential Backups: Captured between full backups to reduce recovery time
- Transaction Log Backups: Enabled for point-in-time recovery in case of failure
- Backup File: The .bak file is included in the /backup folder for restoration

- Recovery Testing: Backups were restored in a test environment to validate integrity and completeness

This strategy ensures that GlobalGadgets can recover from data loss, corruption, or system failure with minimal downtime.

Section 5: CONCLUSION

The GlobalGadgetsDB project successfully delivers a robust, scalable, and fully normalized SQL Server database tailored to the requirement of a consumer electronics retailer. Through careful schema design, functional T-SQL implementation, and strategic management practices, the system addresses key business challenges including inventory control, order tracking, payment validation, and employee accountability.

The use of stored procedures, functions, views, and triggers ensures that business logic is enforced consistently and efficiently. Features like automatic restocking on order cancellation and role-based employee assignment demonstrate the system's ability to automate workflows and reduce manual overhead.

The database is designed with long-term sustainability in mind, incorporating best practices in data integrity, security, and backup. It is ready for deployment in a production environment and can be extended to support future enhancements such as mobile integration, analytics dashboards, and multi-branch operations.

This project reflects a deep understanding of relational database principles and showcases the ability to translate business requirements into a technically sound and maintainable solution.

REFERENCES

The following resources were consulted during the design and implementation of GlobalGadgetsDB:

- **Database Systems: A practical Approach to Design, Implementation, and Management** by Thomas Connolly and Carolyn Begg: A comprehensive text for practical design and management.
- **Principles of Database Systems** by J.D. Ullman: A foundational text on the principles of database systems
- **SQL Performance Explained** by Markus Winand: A simply explained text for sql performance optimization.
- **Other articles** like Google, GeeksForGeeks, Datacamp, Github, Copilot e.t.c

These references supported decisions around schema normalization, T-SQL syntax, and best practices in database management.

APPENDICES

Appendix A:

Akitoye Michael.sql.zip

This file Includes:

- CREATE TABLE statements for all entities
- INSERT INTO statements for sample data
- Stored procedures, functions, views, and triggers

Appendix B

GlobalGadgetsDB.bak

This file Contains:

- Full backup of the deployed database
- Ready for restoration in SQL Server Management Studio