RetailDB

T-SQL Project

Author:

Moaaz Abu-Saif Megahed

CONTENT

DATABASE CREATION

Requirements Gathering & Analysis

Develop a database for a fictional retail company that manage and track:

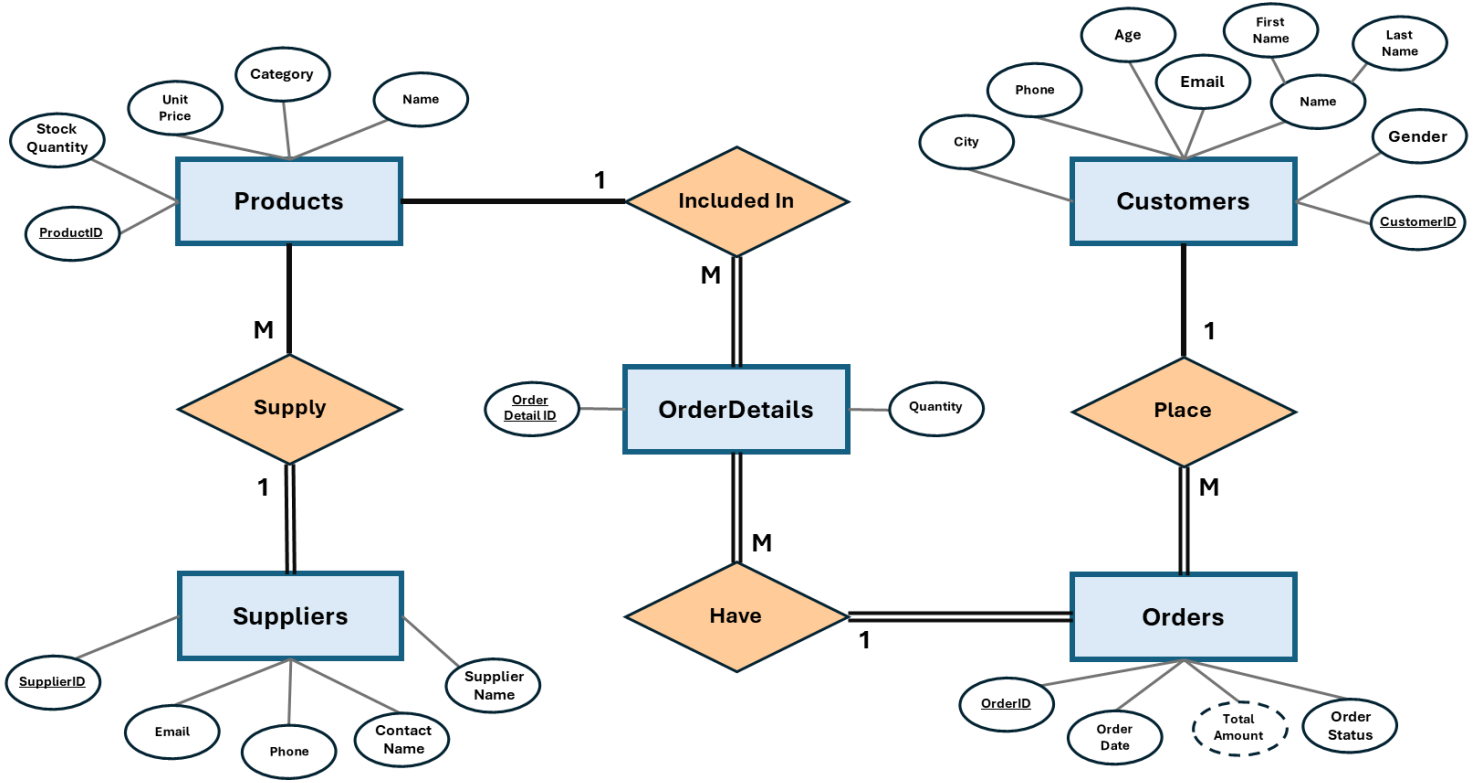
Products	Customers	Orders	Suppliers
Product details, Categories, Stock levels.	Customer information, Order history.	Order details, Order Status, Shipping information.	Supplier details, Product Supply information.

Entities and Attributes:

- **Products:** Product ID, Product Name, Category, Unit Price, Stock Quantity.
- **Customers:** Customer ID, First Name, Last Name, Age, Email, Phone , Gender ,City.
- **Orders:** OrderID , CustomerID , OrderDate, OrderStatus.
- **OrderDetails:** OrderDetailID, OrderID, ProductID, Quantity.
- **Suppliers:** SupplierID, SupplierName, ContactName, Phone, Email.

Objectives: Mastery of joins, views, stored procedures, basic CRUD operations, advanced queries, and fundamental database design.

Conceptual Design (ER Diagram)



Logical Design (Mapping)

Suppliers

SupplierID	SupplierName	ContactName	Phone	Email

Products

ProductID	ProductName	Category	UnitPrice	StockQuantity	SupplierID

OrderDetails

OrderDetailID	Quantity	ProductID	OrderID

Orders

OrderID	OrderDate	OrderStatus	CustomerID

Customers

CustomerID	FirstName	LastName	Age	Phone	Email	Gender	City

Implementation (Create Database)

- Checks if a database named 'RetailDB' exists. If it doesn't, it creates 'RetailDB'; otherwise, it prints a message stating that the database already exists.

- Checks if 'RetailDB' exists. If it does, it drops 'RetailDB'; otherwise, it prints a message stating that the database does not exist.

```sql
USE MASTER;
GO

-- Create database if it not exists
IF DB_ID('RetailDB') IS NULL
BEGIN
    CREATE DATABASE RetailDB;
END
ELSE
BEGIN
    PRINT 'Database "RetailDB" already exists.';
END;
GO

-- Drop database if it exists
IF DB_ID('RetailDB') IS NOT NULL
BEGIN
  USE MASTER;
    DROP DATABASE RetailDB;
END
ELSE
BEGIN
    PRINT 'Database "RetailDB" does not exist, so it cannot be dropped.';
END;
GO
```

Implementation (Create Schemas & Sequences)

- Creates two schemas within this database:
  - *Sales Schema for [Customers , Orders , OrderDetails] Tables.*
  - *Production Schema for [Suppliers , Products] Tables.*

```sql
USE RetailDB;
GO

-- Create Schemas
CREATE SCHEMA Sales;
GO
CREATE SCHEMA Production;
GO
```

- Creates 5 sequences in the 'RetailDB' database, each starting with 1 and incrementing by 1. These sequences are:
  - *Production.SuppliersIDs for supplier IDs.*
  - *Production.ProductsIDs for product IDs.*
  - *Sales.CustomersIDs for customer IDs.*
  - *Sales.OrdersIDs for order IDs.*
  - *Sales.OrderDetailsIDs for order detail IDs.*

```sql
-- Create Sequences
CREATE SEQUENCE Production.SuppliersIDs START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE Production.ProductsIDs START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE Sales.CustomersIDs START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE Sales.OrdersIDs START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE Sales.OrderDetailsIDs START WITH 1 INCREMENT BY 1;
GO
```

Implementation (Create Tables)

1. Suppliers Table:

- SupplierID: Primary key, auto-generated from *SuppliersIDs* Sequence.
- SupplierName: Non-null VARCHAR(100).
- ContactName, Phone, Email: Nullable VARCHAR columns.

```sql
-- Suppliers Table
CREATE TABLE Production.Suppliers (
    SupplierID INT PRIMARY KEY DEFAULT NEXT VALUE FOR Production.SuppliersIDs,
    SupplierName VARCHAR(100) NOT NULL,
    ContactName VARCHAR(100),
    Phone VARCHAR(20),
    Email VARCHAR(100)
);
GO
```

2. Products Table:

- ProductID: Primary key, auto-generated from *ProductsIDs* Sequence.
- ProductName: Non-null VARCHAR(100).
- Category: Nullable VARCHAR(50).
- UnitPrice: Non-null DECIMAL(10, 2) with a check constraint > 0.
- StockQuantity: Non-null INT with a check constraint > 0.
- SupplierID: Foreign key referencing *Production.Suppliers.SupplierID*.

```sql
-- Products Table
CREATE TABLE Production.Products (
    ProductID INT PRIMARY KEY DEFAULT NEXT VALUE FOR Production.ProductsIDs,
    ProductName VARCHAR(100) NOT NULL,
    Category VARCHAR(50),
    UnitPrice DECIMAL(10, 2) NOT NULL CHECK( UnitPrice > 0 ),
    StockQuantity INT NOT NULL CHECK( StockQuantity > 0 ),
  SupplierID INT,
  CONSTRAINT fk_supplier FOREIGN KEY (SupplierID) REFERENCES Production.Suppliers(SupplierID)
);
GO
```

Implementation (Create Tables)

3. Customers Table:

- CustomerID: Primary key, auto-generated from *CustomersIDs* Sequence.
- FirstName, LastName: Non-null VARCHAR(50).
- Age: INT with a check constraint ≥ 16.
- Phone, City: Nullable VARCHAR columns.
- Email: Non-null, unique VARCHAR(100).
- Gender: Non-null VARCHAR(10) with a check constraint for 'Male' or 'Female'.

```sql
-- Customers Table
CREATE TABLE Sales.Customers (
    CustomerID INT PRIMARY KEY DEFAULT NEXT VALUE FOR Sales.CustomersIDs,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
  Age INT CHECK( AGE >= 16 ),
  Phone VARCHAR(20),
    Email VARCHAR(100) NOT NULL UNIQUE,
  Gender VARCHAR(10) NOT NULL CHECK(Gender IN ('Male', 'Female')),
  City VARCHAR(50),
);
```

Implementation (Create Tables)

4. Orders Table:

- OrderID: Primary key, auto-generated from *OrdersIDs* Sequence.
- OrderDate: Non-null DATE with a check constraint < *GETDATE()*.
- OrderStatus: VARCHAR(20) with a check constraint for 'Processing', 'Shipped', 'Delivered', or 'Cancelled'.
- CustomerID: Non-null INT with a foreign key constraint referencing *Sales.Customers.CustomerID*.

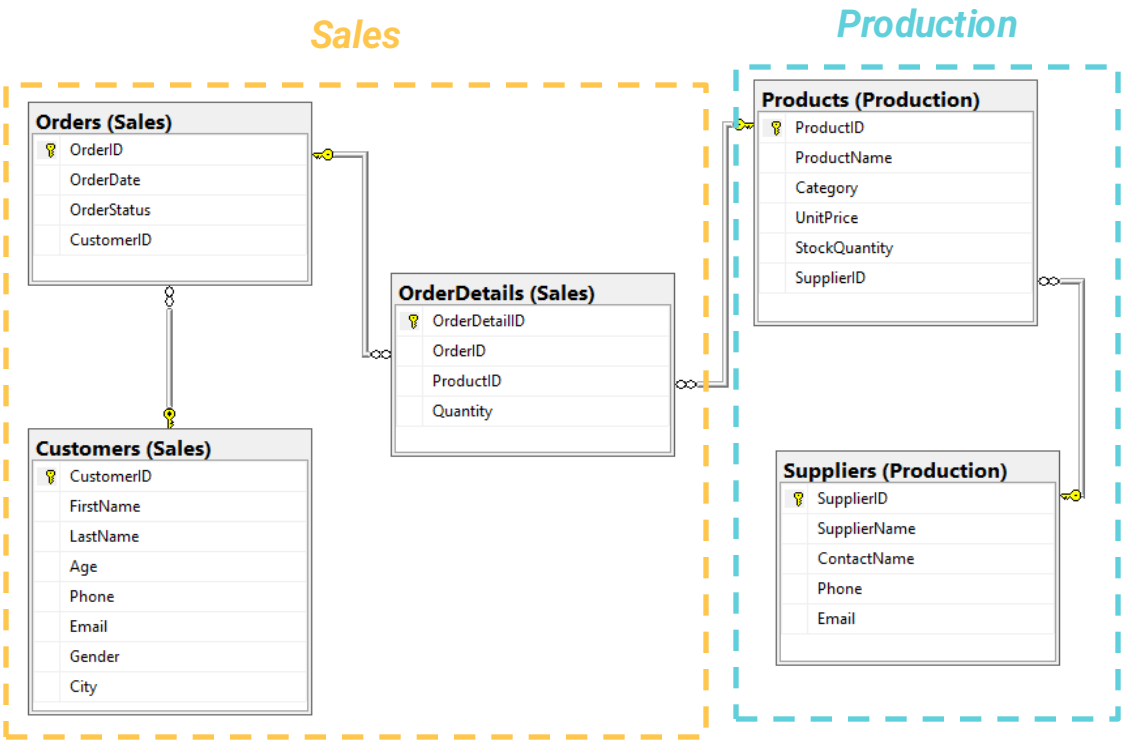
```sql
-- Orders Table
CREATE TABLE Sales.Orders (
  OrderID INT PRIMARY KEY DEFAULT NEXT VALUE FOR Sales.OrdersIDs,
  OrderDate DATE NOT NULL CHECK( OrderDate < GETDATE() ),
  OrderStatus VARCHAR(20) CHECK( OrderStatus IN ('Processing','Shipped','Delivered','Cancelled') ),
  CustomerID INT NOT NULL,
  CONSTRAINT fk_customer FOREIGN KEY (CustomerID) REFERENCES Sales.Customers(CustomerID)
);
GO
```

5. Order Details Table:

- OrderDetailID: Primary key, auto-generated from *OrderDetailsIDs* Sequence.
- OrderID: Non-null INT with a foreign key constraint referencing *Sales.Orders.OrderID*.
- ProductID: Non-null INT with a foreign key constraint referencing *Production.Products.ProductID*.
- Quantity: Non-null INT with a check constraint > 0.

```sql
-- OrderDetails Table
CREATE TABLE Sales.OrderDetails (
  OrderDetailID INT PRIMARY KEY DEFAULT NEXT VALUE FOR Sales.OrderDetailsIDs,
  OrderID INT NOT NULL,
  ProductID INT NOT NULL,
  Quantity INT NOT NULL CHECK( Quantity > 0 ),
  CONSTRAINT fk_order FOREIGN KEY (OrderID) REFERENCES Sales.Orders(OrderID),
  CONSTRAINT fk_product FOREIGN KEY (ProductID) REFERENCES Production.Products(ProductID)
);
GO
```

Database Diagram



DATA INERTION

Insert Sample Data (Customers)

```sql
INSERT INTO Sales.Customers (FirstName, LastName, Age, Phone, Email, Gender, City)
VALUES
('Omar', 'Abdel Rahman', 28, '01134455667', 'omar.rahman@gmail.com', 'Male', 'Gharbia'),
('Sara', 'Mohamed', 20, '01045566778', 'sara.mohamed@yahoo.com', 'Female', 'Alexandria'),
('Hassan', 'Ali', 35, '01156677889', 'hassan.ali@hotmail.com', 'Male', 'Giza'),
-- 145 Row ...
('Saad', 'Tarek', 36, '01011009988', 'saad.tarek@outlook.com', 'Male', 'Cairo' ),
('Rania', 'Hesham', 29, '01100998877', 'rania.hesham@outlook.com', 'Female', 'Giza' );
GO
```

Messages

```
(150 rows affected)

Completion time: 2024-07-20T05:01:27.0195143+03:00
```

Insert Sample Data (Orders)

```sql
INSERT INTO Sales.Orders (OrderDate, OrderStatus, CustomerID) VALUES
('2024-01-03', 'Delivered', 56),
('2024-03-12', 'Shipped', 112),
('2024-05-21', 'Delivered', 23),
-- 245 Row ...
('2024-06-13', 'Shipped', 109),
('2024-04-04', 'Delivered', 37);
GO
```

Messages

```
(250 rows affected)

Completion time: 2024-07-20T05:06:10.9591672+03:00
```

Insert Sample Data (Suppliers)

```sql
INSERT INTO Production.Suppliers (SupplierName, ContactName, Phone, Email) VALUES
('El Sewedy Electric', 'Ahmed El Sewedy', '01234567890', 'info@elsewedy.com'),
('Bahgat Group', 'Mostafa Bahgat', '01098765432', 'contact@bahgatgroup.com'),
('Raya Distribution', 'Mohamed Abdallah', '01223344556', 'sales@rayadistribution.com'),
-- 20 Row ...
('Electro Trade', 'Amr El Feky', '01156677889', 'info@electrotrade.com'),
('Nile Electronics', 'Hesham Gamal', '01267788990', 'info@nileelectronics.com');
GO
```

Messages

```
(25 rows affected)

Completion time: 2024-07-20T05:09:57.5501505+03:00
```

Insert Sample Data (Products)

```sql
INSERT INTO Production.Products (ProductName, Category, UnitPrice, StockQuantity, SupplierID) VALUES
('LG OLED TV', 'Television', 15000.00, 50, 1),
('Samsung Galaxy S21', 'Mobile Phone', 12000.00, 100, 2),
('Sony PlayStation 5', 'Gaming Console', 8000.00, 30, 3),
-- 45 Row ...
('Amazon Echo Dot', 'Speaker', 2000.00, 150, 20),
('Google Nest Hub', 'Smart Home', 5000.00, 60, 19);
GO
```

Messages

```
(50 rows affected)

Completion time: 2024-07-20T05:13:00.1691672+03:00
```

Insert Sample Data (OrderDetails)

```sql
INSERT INTO Sales.OrderDetails (OrderID, ProductID, Quantity) VALUES
(1, 12, 1),
(1, 5, 1),
(2, 35, 2),
-- 315 Row ...
(249, 19, 1),
(250, 50, 2);
GO
```

Messages

```
(320 rows affected)

Completion time: 2024-07-20T05:15:53.6885122+03:00
```

DATABASE TESTING

CRUD Operations (Customers)

Insert

Delete

```sql
INSERT INTO Sales.Customers (FirstName, LastName, Age, Phone, Email, Gender, City) VALUES
('Moaaz', 'Saif', 20, '01156196874', 'moaaz.saif@gmail.com', 'Male', 'Giza');
SELECT * FROM Sales.Customers;
GO

UPDATE Sales.Customers SET Age = 21 WHERE CustomerID = 151;
SELECT * FROM Sales.Customers;
GO

DELETE FROM Sales.Customers WHERE CustomerID = 151;
SELECT * FROM Sales.Customers;
GO
```

Update

Read

Results | Messages

	CustomerID	FirstName	LastName	Age	Phone	Email	Gender	City
149	149	Saad	Tarek	36	01011009988	saad.tarek@outlook.com	Male	Cairo
150	150	Rania	Hesham	29	01100998877	rania.hesham@outlook.com	Female	Giza
151	151	Moaaz	Saif	20	01156196874	moaaz.saif@gmail.com	Male	Giza

	CustomerID	FirstName	LastName	Age	Phone	Email	Gender	City
149	149	Saad	Tarek	36	01011009988	saad.tarek@outlook.com	Male	Cairo
150	150	Rania	Hesham	29	01100998877	rania.hesham@outlook.com	Female	Giza
151	151	Moaaz	Saif	21	01156196874	moaaz.saif@gmail.com	Male	Giza

	CustomerID	FirstName	LastName	Age	Phone	Email	Gender	City
147	147	Gamal	Omar	53	01133221100	gamal.omar@outlook.com	Male	Luxor
148	148	Reem	Fathy	25	01222110099	reem.fathy@outlook.com	Female	Aswan
149	149	Saad	Tarek	36	01011009988	saad.tarek@outlook.com	Male	Cairo
150	150	Rania	Hesham	29	01100998877	rania.hesham@outlook.com	Female	Giza

Query executed successfully.

CRUD Operations (Orders)

Insert

Delete

```sql
INSERT INTO Sales.Orders (OrderDate, OrderStatus, CustomerID) VALUES
('2024-07-10', 'Processing', 18);
SELECT * FROM Sales.Orders;
GO

UPDATE Sales.Orders SET OrderStatus = 'Cancelled' WHERE OrderID = 251;
SELECT * FROM Sales.Orders;
GO

DELETE FROM Sales.Orders WHERE OrderID = 251;
SELECT * FROM Sales.Orders;
GO
```

Update

Results | Messages

	OrderID	OrderDate	OrderStatus	CustomerID
249	249	2024-06-13	Shipped	109
250	250	2024-04-04	Delivered	37
251	251	2024-07-10	Processing	18

	OrderID	OrderDate	OrderStatus	CustomerID
249	249	2024-06-13	Shipped	109
250	250	2024-04-04	Delivered	37
251	251	2024-07-10	Cancelled	18

	OrderID	OrderDate	OrderStatus	CustomerID
248	248	2024-01-17	Processing	79
249	249	2024-06-13	Shipped	109
250	250	2024-04-04	Delivered	37

Query executed successfully.

Read

CRUD Operations (OrderDetails)

Insert

Delete

```sql
INSERT INTO Sales.OrderDetails (OrderID, ProductID, Quantity) VALUES
(85, 16, 2);
SELECT * FROM Sales.OrderDetails;
GO

UPDATE Sales.OrderDetails SET Quantity = 1 WHERE OrderDetailID = 321;
SELECT * FROM Sales.OrderDetails;
GO

DELETE FROM Sales.OrderDetails WHERE OrderDetailID = 321;
SELECT * FROM Sales.OrderDetails;
GO
```

Update

Results | Messages

	OrderDetailID	OrderID	ProductID	Quantity
319	319	249	19	1
320	320	250	50	2
321	321	85	16	2

	OrderDetailID	OrderID	ProductID	Quantity
319	319	249	19	1
320	320	250	50	2
321	321	85	16	1

	OrderDetailID	OrderID	ProductID	Quantity
318	318	249	49	1
319	319	249	19	1
320	320	250	50	2

Query executed successfully.

Read

CRUD Operations (Products)

Insert

Delete

```sql
INSERT INTO Production.Products (ProductName,Category,UnitPrice,StockQuantity,SupplierID) VALUES
('Dell G3 15-3500', 'Laptop', 23000.00,10,12);
SELECT * FROM Production.Products;
GO

UPDATE Production.Products SET UnitPrice = 25000.00 WHERE ProductID = 51;
SELECT * FROM Production.Products;
GO

DELETE FROM Production.Products WHERE ProductID = 51;
SELECT * FROM Production.Products;
GO
```

Update

Results | Messages

	ProductID	ProductName	Category	UnitPrice	StockQuantity	SupplierID
49	49	Amazon Echo Dot	Speaker	2000.00	150	20
50	50	Google Nest Hub	Smart Home	5000.00	60	19
51	51	Dell G3 15-3500	Laptop	23000.00	10	12

	ProductID	ProductName	Category	UnitPrice	StockQuantity	SupplierID
49	49	Amazon Echo Dot	Speaker	2000.00	150	20
50	50	Google Nest Hub	Smart Home	5000.00	60	19
51	51	Dell G3 15-3500	Laptop	25000.00	10	12

	ProductID	ProductName	Category	UnitPrice	StockQuantity	SupplierID
48	48	JBL Flip 5	Speaker	3000.00	100	23
49	49	Amazon Echo Dot	Speaker	2000.00	150	20
50	50	Google Nest Hub	Smart Home	5000.00	60	19

Query executed successfully.

Read

CRUD Operations (Suppliers)

Insert

Delete

```sql
INSERT INTO Production.Suppliers(SupplierName,ContactName,Phone,Email) VALUES
('Giza Group', 'Moaaz Saif', '01156196874','info@gizagroup.com');
SELECT * FROM Production.Suppliers;
GO

UPDATE Production.Suppliers SET SupplierName = 'Giza Trading' WHERE SupplierID = 26;
SELECT * FROM Production.Suppliers;
GO

DELETE FROM Production.Suppliers WHERE SupplierID = 26;
SELECT * FROM Production.Suppliers;
GO
```

Update

Results | Messages

	SupplierID	SupplierName	ContactName	Phone	Email
24	24	Electro Trade	Amr El Feky	01156677889	info@electrotrade.com
25	25	Nile Electronics	Hesham Gamal	01267788990	info@nileelectronics.com
26	26	Giza Group	Moaaz Saif	01156196874	info@gizagroup.com

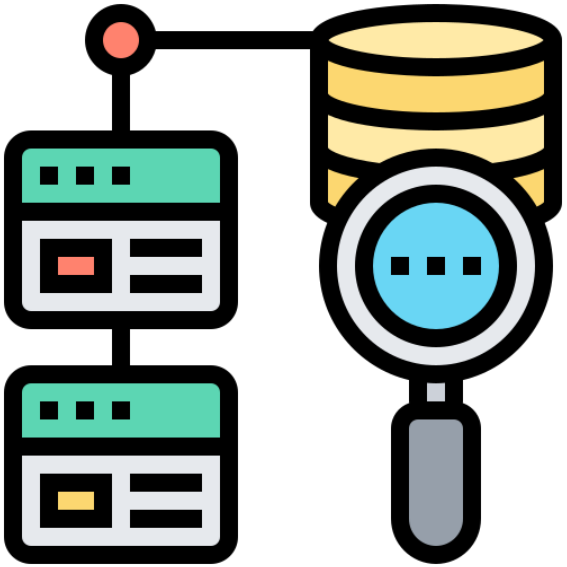
	SupplierID	SupplierName	ContactName	Phone	Email
24	24	Electro Trade	Amr El Feky	01156677889	info@electrotrade.com
25	25	Nile Electronics	Hesham Gamal	01267788990	info@nileelectronics.com
26	26	Giza Trading	Moaaz Saif	01156196874	info@gizagroup.com

	SupplierID	SupplierName	ContactName	Phone	Email
22	22	Power Electronics	Ahmed Samir	01134455667	info@powerelectronics.com
23	23	Delta Trading	Mostafa Naguib	01145566778	info@deltatrading.com
24	24	Electro Trade	Amr El Feky	01156677889	info@electrotrade.com
25	25	Nile Electronics	Hesham Gamal	01267788990	info@nileelectronics.com

Query executed successfully.

Read

ADVANCED QUERIES



Joins Queries (Inner , Full Outer)

```sql
-- INNER JOIN
SELECT o.OrderID, o.OrderDate, o.OrderStatus, c.CustomerID,
     c.FirstName + ' ' + c.LastName AS CustomerName
FROM Sales.Orders o INNER JOIN Sales.Customers c
ON o.CustomerID = c.CustomerID;
```

Combines rows from *Orders* and *Customers* where CustomerID matches, resulting in a dataset showing orders with customer names.

```sql
-- FULL OUTER JOIN
SELECT s.SupplierID , s.SupplierName , s.ContactName,
     p.ProductID , p.ProductName , p.StockQuantity
FROM Production.Suppliers s FULL OUTER JOIN Production.Products p
ON s.SupplierID = p.SupplierID;
```

Retrieves all rows from both *Suppliers* and *Products*, combining rows with matching SupplierID and including NULLs where there is no match.

Joins Queries (Left , Right)

```sql
-- LEFT JOIN
SELECT s.SupplierID , s.SupplierName , s.ContactName,
     p.ProductID , p.ProductName , p.StockQuantity
FROM Production.Suppliers s LEFT JOIN Production.Products p
ON s.SupplierID = p.SupplierID;
```

Selects all rows from *Suppliers* and the matching rows from *Products* based on SupplierID, with NULLs for products that don't match.

```sql
-- RIGHT JOIN
SELECT s.SupplierID , s.SupplierName , s.ContactName,
     p.ProductID , p.ProductName , p.StockQuantity
FROM Production.Suppliers s RIGHT JOIN Production.Products p
ON s.SupplierID = p.SupplierID;
```

Selects all rows from *Products* and the matching rows from *Suppliers* based on SupplierID, with NULLs for suppliers that don't match.

Joins Queries (Self , Cross)

```sql
-- SELF JOIN
SELECT
    c1.CustomerID AS CustomerID1, c1.FirstName + ' ' + c1.LastName AS Customer1,
  c2.CustomerID AS CustomerID2, c2.FirstName + ' ' + c2.LastName AS Customer2,
  c1.Age, c1.Gender, c1.City
FROM  Sales.Customers c1 INNER JOIN Sales.Customers c2
ON c1.Age = c2.Age AND c1.Gender = c2.Gender AND
   c1.City = c2.City AND c1.CustomerID <> c2.CustomerID;
```

Joins the *Customers* table with itself on Age, Gender, and City to find customers with the same information but different IDs.

```sql
-- CROSS JOIN
SELECT p.ProductID, p.ProductName,
       s.SupplierID, s.SupplierName
FROM Production.Products p
CROSS JOIN  Production.Suppliers s;
```

Generates a Cartesian product of *Products* and *Suppliers*, pairing every product with every supplier.

Subqueries

```sql
-- Find the Most Expensive Product Ordered by Each Customer
SELECT o.CustomerID, p.ProductName, MAX(p.UnitPrice) AS MaxPrice
FROM Sales.Orders o
INNER JOIN Sales.OrderDetails od ON o.OrderID = od.OrderID
INNER JOIN Production.Products p ON od.ProductID = p.ProductID
WHERE p.UnitPrice = (
        SELECT MAX(p2.UnitPrice)
        FROM Sales.OrderDetails od2
        INNER JOIN Production.Products p2
    ON od2.ProductID = p2.ProductID
        WHERE od2.OrderID IN (
      SELECT OrderID FROM Sales.Orders
      WHERE CustomerID = o.CustomerID
    )
  )
GROUP BY o.CustomerID, p.ProductName
ORDER BY MaxPrice DESC;
```

This query retrieves the most expensive product ordered by each customer. It uses a correlated subquery to find the maximum unit price per customer.

```sql
-- Find Suppliers Who Supply Products Not Ordered in the Last 2 Months
SELECT s.SupplierID, s.SupplierName , s.ContactName
FROM Production.Suppliers s
WHERE s.SupplierID IN ((
        SELECT p.SupplierID
        FROM  Production.Products p
        WHERE p.ProductID NOT IN (
                SELECT od.ProductID
                FROM Sales.OrderDetails od
                INNER JOIN Sales.Orders o
                ON od.OrderID = o.OrderID
                WHERE o.OrderDate >= DATEADD(MONTH, -2, GETDATE())
        )
    )
);
```

This query identifies suppliers who supply products that have not been ordered in the last 6 months.

Common Table Expressions (CTEs)

```sql
-- CTE to calculate total amount for each order
WITH OrderTotalAmountCTE AS (
    SELECT o.OrderID, o.OrderDate , o.OrderStatus , o.CustomerID,
           SUM(od.Quantity * p.UnitPrice) AS TotalAmount
    FROM Sales.Orders o
    INNER JOIN Sales.OrderDetails od ON o.OrderID = od.OrderID
    INNER JOIN Production.Products p ON od.ProductID = p.ProductID
    GROUP BY o.OrderID ,o.OrderDate , o.OrderStatus , o.CustomerID
)
SELECT * FROM OrderTotalAmountCTE;
```

Calculate the total amount for each order by summing the product of quantity and unit price for each order.

```sql
-- CTE to calculate total sales per category
WITH CategorySalesCTE AS (
    SELECT p.Category , SUM(od.Quantity * p.UnitPrice) AS TotalSales
    FROM Sales.OrderDetails od
    INNER JOIN Production.Products p ON od.ProductID = p.ProductID
    GROUP BY p.Category
)
SELECT * FROM CategorySalesCTE ORDER BY TotalSales DESC;
```

Compute the total sales for each product category by summing the product of quantity and unit price and orders the results by total sales in descending order.

Window Functions

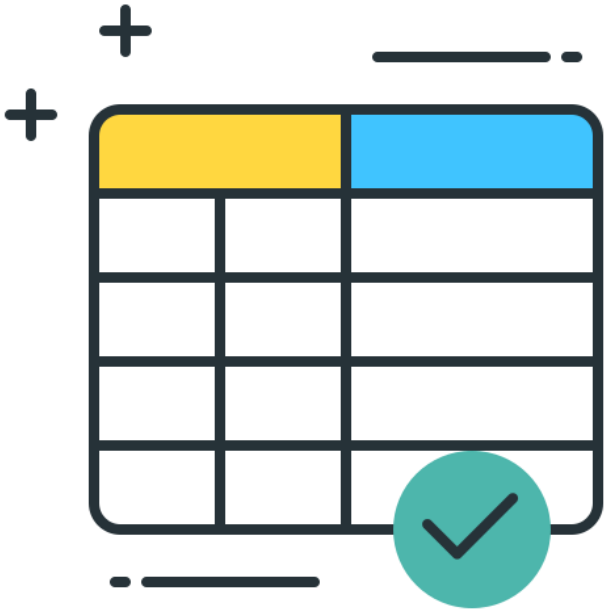
```sql
-- Ranking Products by Total Sales
SELECT
    p.ProductID,
    p.ProductName,
    SUM(od.Quantity * p.UnitPrice) AS TotalSales,
    DENSE_RANK() OVER (ORDER BY SUM(od.Quantity * p.UnitPrice) DESC) AS SalesRank
FROM Sales.OrderDetails od
INNER JOIN Production.Products p ON od.ProductID = p.ProductID
GROUP BY p.ProductID, p.ProductName;
```

Ranks products based on their total sales amount, with the highest-selling product receiving the top rank.

```sql
-- Determining Monthly Sales Trends
SELECT
    YEAR(o.OrderDate) AS Year,
    MONTH(o.OrderDate) AS Month,
    SUM(od.Quantity * p.UnitPrice) AS TotalSales,
    COALESCE(LAG(SUM(od.Quantity * p.UnitPrice)) OVER (ORDER BY YEAR(o.OrderDate), MONTH(o.OrderDate)),0)
  AS PreviousMonthSales,
    SUM(od.Quantity * p.UnitPrice)
  - COALESCE(LAG(SUM(od.Quantity * p.UnitPrice)) OVER (ORDER BY YEAR(o.OrderDate), MONTH(o.OrderDate)), 0)
  AS SalesDifference
FROM Sales.Orders o
INNER JOIN Sales.OrderDetails od ON o.OrderID = od.OrderID
INNER JOIN Production.Products p ON od.ProductID = p.ProductID
GROUP BY YEAR(o.OrderDate), MONTH(o.OrderDate);
```

Determining monthly sales trends by calculates monthly sales amounts and compares each month’s sales to the previous month’s sales.

VIEWS



Slow Sales Products View

The view shows products with fewer than 15 units sold in the past year, joining the Products, OrderDetails, and Orders tables to aggregate and filter sales data.

```sql
CREATE VIEW vw_SlowSalesProducts AS
SELECT
    p.ProductID,
    p.ProductName,
    p.Category,
    p.UnitPrice,
    p.StockQuantity,
    COALESCE(SUM(od.Quantity), 0) AS TotalSoldLastYear
FROM
    Production.Products p
LEFT JOIN
    Sales.OrderDetails od ON p.ProductID = od.ProductID
LEFT JOIN
    Sales.Orders o ON od.OrderID = o.OrderID AND o.OrderDate >= DATEADD(YEAR, -1, GETDATE())
GROUP BY
    p.ProductID, p.ProductName, p.Category, p.UnitPrice, p.StockQuantity
HAVING
    COALESCE(SUM(od.Quantity), 0) < 15;
```

Monthly Sales Summary View

This view provides a summary of monthly sales including total sales, average order value, and the number of orders and customers.

```sql
CREATE VIEW vw_MonthlySalesSummary AS
SELECT
    DATEPART(MONTH, o.OrderDate) AS Month,
    DATEPART(YEAR, o.OrderDate) AS Year,
    COUNT(DISTINCT o.OrderID) AS TotalOrders,
    COUNT(DISTINCT c.CustomerID) AS CustomersNum,
    CAST(SUM(od.Quantity * p.UnitPrice) AS DECIMAL(10,2)) AS TotalSales,
    CAST(AVG(od.Quantity * p.UnitPrice) AS DECIMAL(10,2)) AS AverageOrderValue
FROM
    Sales.Orders o
JOIN
    Sales.OrderDetails od ON o.OrderID = od.OrderID
JOIN
    Production.Products p ON od.ProductID = p.ProductID
JOIN
  Sales.Customers c ON c.CustomerID = o.CustomerID
GROUP BY
    DATEPART(YEAR, o.OrderDate), DATEPART(MONTH, o.OrderDate);
```

Actual Profit Suppliers View

The view calculates the total profit for each supplier by summing the product of quantity sold and unit price, joining Suppliers, Products, and OrderDetails tables.

```sql
CREATE VIEW vw_ActualProfitSuppliers AS
SELECT
  s.SupplierID,
  s.SupplierName,
  s.ContactName,
  SUM(od.Quantity*p.UnitPrice) AS TotalProfit
FROM
  Production.Suppliers s
LEFT JOIN
  Production.Products p ON s.SupplierID = p.SupplierID
INNER JOIN
  Sales.OrderDetails od ON od.ProductID = p.ProductID
GROUP BY s.SupplierID,
     s.SupplierName,
     s.ContactName;
```

Low Stock Products View

The view lists products with stock quantities less than 10, showing their ID, name, category, unit price, and current stock level.

```sql
CREATE VIEW vw_LowStockProducts AS
SELECT
    p.ProductID,
    p.ProductName,
    p.Category,
    p.UnitPrice,
    p.StockQuantity
FROM
    Production.Products p
WHERE
    p.StockQuantity < 10;
```

Active Orders View

This view summarizes active orders by listing the order ID, date, status, customer ID, customer name, and the total amount for orders that are 'Processing' or 'Shipped'.

```sql
CREATE VIEW vw_ActiveOrders AS
SELECT
    o.OrderID,
    o.OrderDate,
    o.OrderStatus,
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    SUM(od.Quantity * p.UnitPrice) AS TotalAmount
FROM
    Sales.Orders o
JOIN
    Sales.Customers c ON o.CustomerID = c.CustomerID
JOIN
    Sales.OrderDetails od ON o.OrderID = od.OrderID
JOIN
    Production.Products p ON od.ProductID = p.ProductID
GROUP BY o.OrderID,
     o.OrderDate,
     o.OrderStatus,
     c.CustomerID,
         c.FirstName + ' ' + c.LastName
HAVING o.OrderStatus IN ('Processing', 'Shipped');
```

STORED PROSDURES

Insert New Order

Creates a stored procedure to insert a new order and update inventory. Handles errors and insufficient stock by rolling back the transaction and printing an error message.

```sql
CREATE PROCEDURE sp_InsertNewOrder
    (@OrderDate DATE, @CustomerID INT,@OrderDetails OrderDtl READONLY)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @OrderID INT = NEXT VALUE FOR Sales.OrdersIDs;

        INSERT INTO Sales.Orders (OrderID, OrderDate, OrderStatus, CustomerID)
        VALUES (@OrderID, @OrderDate, 'Processing', @CustomerID);

        INSERT INTO Sales.OrderDetails (OrderID, ProductID, Quantity)
        SELECT @OrderID, ProductID, Quantity
        FROM @OrderDetails;

        UPDATE p
        SET StockQuantity = StockQuantity - od.Quantity
        FROM Production.Products p
        INNER JOIN @OrderDetails od ON p.ProductID = od.ProductID
        WHERE p.ProductID = od.ProductID AND StockQuantity >= od.Quantity;

        IF EXISTS (
            SELECT 1 FROM Production.Products p
            INNER JOIN @OrderDetails od
      ON p.ProductID = od.ProductID
            WHERE StockQuantity < od.Quantity
        )
        BEGIN
            PRINT('Insufficient stock for one or more products.');
            ROLLBACK TRANSACTION;
            RETURN;
        END

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
    PRINT('Something is wrong!.');
    END CATCH
END;
```

Calculate Total Sales For Customer

Calculate and return the total quantity of products and the total sales amount for a specific customer, identified by @CustID, and grouping results by customer and order status.

```sql
CREATE PROCEDURE sp_CalculateTotalSalesForCustomer(@CustID AS INT)
AS
BEGIN
    SELECT
      c.CustomerID,
      c.FirstName,
      c.LastName,
      o.OrderStatus,
      SUM(od.Quantity) AS TotalProducts,
      SUM(p.UnitPrice * od.Quantity) AS TotalSales
    FROM
      Sales.Customers c
    JOIN
      Sales.Orders o ON c.CustomerID = o.CustomerID
    JOIN
      Sales.OrderDetails od ON o.OrderID = od.OrderID
    JOIN
      Production.Products p ON od.ProductID = p.ProductID
    WHERE
      c.CustomerID = @CustID
    GROUP BY
      c.CustomerID, c.FirstName, c.LastName , o.OrderStatus;
END;
```

Predict Next Purchase Date For Customer

Predict the next purchase date for a customer by calculating the average interval between their past purchases and adding it to the date of their last purchase.

```sql
CREATE PROCEDURE sp_PredictNextPurchaseDate (@CustID AS INT)
AS
BEGIN
  WITH PurchaseIntervals AS (
        SELECT
            CustomerID,
            DATEDIFF(DAY,
                LAG(OrderDate) OVER (PARTITION BY CustomerID ORDER BY OrderDate),
                OrderDate) AS Interval
        FROM
            Sales.Orders
        WHERE
            CustomerID = @CustID
    ),

    AverageIntervals AS (
        SELECT
            AVG(Interval) AS AvgInterval
        FROM
            PurchaseIntervals
        WHERE
            Interval IS NOT NULL
    )

    SELECT
        c.CustomerID,
        c.FirstName,
        c.LastName,
        MAX(o.OrderDate) AS LastOrderDate,
        DATEADD(DAY, ai.AvgInterval, MAX(o.OrderDate)) AS PredictedNextPurchaseDate
    FROM
        Sales.Customers c
    JOIN
        Sales.Orders o ON c.CustomerID = o.CustomerID
    CROSS JOIN
        AverageIntervals ai
    WHERE
        c.CustomerID = @CustID
    GROUP BY
        c.CustomerID, c.FirstName, c.LastName, ai.AvgInterval;
END;
```

Restock Low Inventory Products

Restock products with inventory below a specified level by updating their stock quantity to the restock level and then returning the updated product details.

```sql
CREATE PROCEDURE sp_RestockLowInventoryProducts (@RestockLevel AS INT)
AS
BEGIN
    UPDATE Production.Products
    SET StockQuantity = @RestockLevel
    WHERE StockQuantity < @RestockLevel;

    SELECT ProductID, ProductName, StockQuantity
    FROM Production.Products
    WHERE StockQuantity = @RestockLevel;
END;
GO
```

THANKS