APUNTES DATA - ERI

Recuerden que pueden existir múltiples formas de resolver. (Lo dejo como funciona en WS3SCHOOL)

🔵 20 Ejercicios SQL - Nivel Inicial (Base: Northwind)

1-Mostrar a todos los clientes.
SELECT *
FROM Customers;

2-Listar los nombres de los productos.

```
SELECT ProductName
FROM Products;
```

3-Mostrar los primeros 10 empleados.

```
SELECT *
FROM Employees
LIMIT 10;
```

4-Listar las órdenes realizadas por clientes de Alemania.

```
SELECT Orders.*
FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
WHERE Customers.Country = 'Germany';
```

5-Obtener todos los productos con precio mayor a 20.

```
SELECT *
FROM Products
WHERE UnitPrice > 20;
```

6-Mostrar los nombres y cargos de los empleados.

```
SELECT FirstName + ' ' + LastName AS FullName, Notes
FROM Employees;
```

(Acá también puede aparecer como title, desplieguen la tabla primero?

7-Listar los nombres de los clientes y su país.

```
SELECT CustomerName, Country
FROM Customers;
```

8-Contar cuántos productos hay en total.

```
SELECT COUNT(*) AS TotalProducts
FROM Products;
```

9-Mostrar los productos ordenados por precio de mayor a menor.

```
SELECT ProductName, UnitPrice
FROM Products
```

```
ORDER BY UnitPrice DESC;
```

10-Mostrar las categorías de productos.

```
SELECT *
FROM Categories;
```

11-Listar los productos y el nombre de su categoría.

```
SELECT Products.ProductName, Categories.CategoryName
FROM Products, Categories
WHERE Products.CategoryID = Categories.CategoryID;
```

12-Mostrar los productos cuyo nombre comience con la letra 'C'.

```
SELECT ProductName
FROM Products
WHERE ProductName LIKE 'C%';
```

13-Ver los proveedores de productos.

```
SELECT DISTINCT Suppliers.SupplierName
FROM Products, Suppliers
WHERE Products.SupplierID = Suppliers.SupplierID;
```

14-Ver todos los pedidos hechos en el año 1997.

```
SELECT *
FROM Orders
WHERE OrderDate LIKE '1997%';
```

15-Mostrar los países únicos de los clientes.

```
SELECT DISTINCT Country
FROM Customers;
```

16-Ver la cantidad de productos por categoría.

```
SELECT Categories.CategoryName, COUNT(Products.ProductID) AS ProductCount
FROM Products, Categories
WHERE Products.CategoryID = Categories.CategoryID
GROUP BY Categories.CategoryName;
```

17-Mostrar las órdenes junto con el nombre del empleado que las gestionó.

```
SELECT Orders.OrderID, Employees.FirstName + ' ' + Employees.LastName AS
EmployeeName
FROM Orders, Employees
WHERE Orders.EmployeeID = Employees.EmployeeID;
```

18-Ver qué productos tienen stock

```
SELECT ProductName, Unit
FROM Products;
```

19-Calcular el precio promedio de los productos.

```
SELECT AVG(Price) AS AveragePrice
FROM Products;
```

20-Mostrar los 5 productos más caros.

```
SELECT TOP 5 ProductName, Price
FROM Products
ORDER BY Price DESC;
```

🟠 20 Ejercicios SQL - Nivel Avanzado (Base: Northwind)

1-Listar los 5 productos más vendidos (por cantidad total en órdenes).

```
SELECT TOP 5 Products.ProductName, SUM(OrderDetails.Quantity) AS TotalSold
FROM Products, OrderDetails
WHERE Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductName
ORDER BY SUM(OrderDetails.Quantity) DESC;
```

2-Mostrar los clientes que realizaron más de 10 pedidos.

```
SELECT Customers.CustomerName, COUNT(Orders.OrderID) AS TotalOrders
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerName
HAVING COUNT(Orders.OrderID) > 10;
```

3-Calcular el ingreso total por cada producto vendido (precio × cantidad).

```
SELECT Products.ProductName, SUM(OrderDetails.Quantity) AS TotalSold
FROM Products, OrderDetails
WHERE Products.ProductID = OrderDetails.ProductID
GROUP BY Products.ProductName
ORDER BY SUM(OrderDetails.Quantity) DESC;
```

Esta solución es una aproximación. En una base real, deberíamos usar el precio registrado en cada orden, ya que puede haber descuentos, promociones o cambios a lo largo del tiempo.

4-Mostrar el nombre del empleado que gestionó más pedidos.

```
SELECT TOP 1 Employees.EmployeeID, COUNT(Orders.OrderID) AS TotalOrders
FROM Employees, Orders
WHERE Employees.EmployeeID = Orders.EmployeeID
GROUP BY Employees.EmployeeID
ORDER BY COUNT(Orders.OrderID) DESC;
```

5-Obtener el promedio de días que tarda un pedido entre la orden y la entrega (OrderDate y ShippedDate).
(En w3schools no está completa la tabla)

```
SELECT AVG(DATEDIFF('d', OrderDate, ShippedDate)) AS AvgDeliveryDays
FROM Orders
WHERE ShippedDate IS NOT NULL;
```

6-Mostrar los productos que nunca fueron pedidos.

```
SELECT ProductName
FROM Products
WHERE ProductID NOT IN (
  SELECT ProductID
  FROM OrderDetails
);
```

7-Encontrar los proveedores que ofrecen más de 3 productos.

```
SELECT Suppliers.SupplierName, COUNT(Products.ProductID) AS TotalProducts
FROM Suppliers, Products
WHERE Suppliers.SupplierID = Products.SupplierID
GROUP BY Suppliers.SupplierName
HAVING COUNT(Products.ProductID) > 3;
```

8-Mostrar el país con más clientes.

```
SELECT  Country, COUNT(*) AS TotalCustomers
FROM Customers
GROUP BY Country
ORDER BY COUNT(*) DESC;
```

9-Mostrar las 3 categorías con más productos registrados.

```
SELECT TOP 3 Categories.CategoryName, COUNT(Products.ProductID) AS ProductCount
FROM Categories, Products
WHERE Categories.CategoryID = Products.CategoryID
GROUP BY Categories.CategoryName
ORDER BY COUNT(Products.ProductID) DESC;
```

10-Calcular el total facturado por año.

```
SELECT YEAR(Orders.OrderDate) AS OrderYear,
      SUM(Products.Price * OrderDetails.Quantity) AS TotalRevenue
FROM Orders, OrderDetails, Products
WHERE Orders.OrderID = OrderDetails.OrderID
  AND OrderDetails.ProductID = Products.ProductID
GROUP BY YEAR(Orders.OrderDate)
ORDER BY YEAR(Orders.OrderDate);
```

11-Listar los productos más vendidos por categoría (uno por categoría).

```
SELECT CategoryName, ProductName, TotalSold
FROM (
  SELECT Categories.CategoryName,
        Products.ProductName,
        SUM(OrderDetails.Quantity) AS TotalSold,
        RANK() OVER (PARTITION BY Categories.CategoryID ORDER BY
SUM(OrderDetails.Quantity) DESC) AS rnk
```

```
  FROM Categories
  JOIN Products ON Categories.CategoryID = Products.CategoryID
  JOIN OrderDetails ON Products.ProductID = OrderDetails.ProductID
  GROUP BY Categories.CategoryName, Products.ProductName
) AS ranked
WHERE rnk = 1;
```

12-Mostrar para cada empleado su nombre y la cantidad de clientes que atendió.

```
SELECT Employees.FirstName + ' ' + Employees.LastName AS EmployeeName,
      COUNT(Orders.CustomerID) AS CustomersAttended
FROM Employees, Orders
WHERE Employees.EmployeeID = Orders.EmployeeID
GROUP BY Employees.FirstName, Employees.LastName;
```

13-Usar CASE para clasificar los productos según su precio (bajo, medio, alto).

```
SELECT ProductName,
      Price,
      CASE
         WHEN Price < 20 THEN 'Bajo'
         WHEN Price BETWEEN 20 AND 50 THEN 'Medio'
         ELSE 'Alto'
      END AS RangoPrecio
FROM Products;
```

14-Listar los pedidos que incluyen más de 3 productos diferentes.

```
SELECT OrderID,
      COUNT(ProductID) AS TotalItems
FROM OrderDetails
GROUP BY OrderID
HAVING COUNT(ProductID) > 3;
```

15-Obtener los clientes que han comprado productos de más de 3 categorías distintas.

```
SELECT Customers.CustomerName,
      Categories.CategoryName
FROM Customers, Orders, OrderDetails, Products, Categories
```

```
WHERE Customers.CustomerID = Orders.CustomerID
  AND Orders.OrderID = OrderDetails.OrderID
  AND OrderDetails.ProductID = Products.ProductID
  AND Products.CategoryID = Categories.CategoryID
ORDER BY Customers.CustomerName, Categories.CategoryName;
```

FUERA DE W3SCHOOL

```
SELECT Customers.CustomerName,
      COUNT(DISTINCT Categories.CategoryID) AS CategoryCount
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
JOIN Products ON OrderDetails.ProductID = Products.ProductID
JOIN Categories ON Products.CategoryID = Categories.CategoryID
GROUP BY Customers.CustomerName
HAVING COUNT(DISTINCT Categories.CategoryID) > 3;
```

16-Mostrar los empleados que nunca realizaron un pedido.

```
SELECT Employees.FirstName & ' ' & Employees.LastName AS EmployeeName
FROM Employees
LEFT JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID
WHERE Orders.OrderID IS NULL;
```

17-Determinar qué categoría genera más ingresos.

```
SELECT Categories.CategoryName,
      SUM(OrderDetails.Quantity * Products.Price) AS TotalRevenue
FROM Categories, Products, OrderDetails
WHERE Categories.CategoryID = Products.CategoryID
  AND Products.ProductID = OrderDetails.ProductID
GROUP BY Categories.CategoryName
ORDER BY TotalRevenue DESC;
```

18-Crear una subconsulta que muestre productos cuyo precio es mayor al precio promedio de su categoría.

```
SELECT CategoryID, AVG(Price) AS AvgPrice
FROM Products
GROUP BY CategoryID;
```

19-Mostrar los 5 clientes que más dinero han gastado (considerando cantidad × precio).

```
SELECT CategoryID, AVG(Price) AS AvgPrice
FROM Products
GROUP BY CategoryID;
```

20-Usar una CTE para calcular el total de ventas por empleado y filtrar los que superan los $20,000

```
SELECT Employees.FirstName & ' ' & Employees.LastName AS EmployeeName,
      SUM(OrderDetails.Quantity * Products.Price) AS TotalVentas
FROM Employees, Orders, OrderDetails, Products
WHERE Employees.EmployeeID = Orders.EmployeeID
  AND Orders.OrderID = OrderDetails.OrderID
  AND OrderDetails.ProductID = Products.ProductID
GROUP BY Employees.FirstName, Employees.LastName
HAVING SUM(OrderDetails.Quantity * Products.Price) > 20000;
```

De acá en adelante recomiendo usar una alternativa a W3SCHOOL. Las dejo para SQL SERVER

1-Obtener los 5 productos más vendidos en términos de cantidad total.

```
SELECT TOP 5 p.ProductName, SUM(od.Quantity) AS TotalSold
FROM [Order Details] od
JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY TotalSold DESC;
```

2-Calcular el ingreso total generado por cada producto (UnitPrice × Quantity).

```
SELECT p.ProductName, SUM(od.UnitPrice * od.Quantity) AS TotalRevenue
FROM [Order Details] od
JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY TotalRevenue DESC;
```

3-Listar los empleados que realizaron más de 100 pedidos.

```
SELECT e.FirstName + ' ' + e.LastName AS EmployeeName, COUNT(*) AS TotalOrders
FROM Orders o
JOIN Employees e ON o.EmployeeID = e.EmployeeID
GROUP BY e.FirstName, e.LastName
HAVING COUNT(*) > 100;
```

4-Mostrar los clientes que realizaron pedidos en más de 3 países diferentes.

```
SELECT c.CustomerID, c.CompanyName, COUNT(DISTINCT o.ShipCountry) AS
CountriesOrdered
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.CompanyName
HAVING COUNT(DISTINCT o.ShipCountry) > 3;
```

5-Mostrar para cada cliente cuántos pedidos hizo por año.

```
SELECT c.CustomerID, YEAR(o.OrderDate) AS Year, COUNT(*) AS OrdersCount
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
GROUP BY c.CustomerID, YEAR(o.OrderDate)
ORDER BY c.CustomerID, Year;
```

6-Usar un CTE para calcular el total de ventas por empleado y filtrar los que superaron los $20.000.

```
WITH VentasEmpleado AS (
  SELECT e.EmployeeID, e.FirstName + ' ' + e.LastName AS EmployeeName,
        SUM(od.Quantity * od.UnitPrice) AS TotalSales
  FROM Employees e
  JOIN Orders o ON e.EmployeeID = o.EmployeeID
  JOIN [Order Details] od ON o.OrderID = od.OrderID
  GROUP BY e.EmployeeID, e.FirstName, e.LastName
)
SELECT * FROM VentasEmpleado
WHERE TotalSales > 20000;
```

7-Determinar qué categoría de productos generó más ingresos totales.

```
SELECT TOP 1 c.CategoryName, SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
FROM Categories c
JOIN Products p ON c.CategoryID = p.CategoryID
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY c.CategoryName
ORDER BY TotalRevenue DESC;
```

8-Mostrar el promedio de unidades por pedido por producto, y ordenar de mayor a menor.

```
SELECT p.ProductName, AVG(od.Quantity * 1.0) AS AvgUnitsPerOrder
FROM Products p
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY p.ProductName
ORDER BY AvgUnitsPerOrder DESC;
```

9-Identificar los productos cuyo precio es mayor al promedio de su categoría.

```
SELECT p.ProductName, p.Price, p.CategoryID
FROM Products p
WHERE p.Price > (
  SELECT AVG(p2.Price)
  FROM Products p2
  WHERE p2.CategoryID = p.CategoryID
);
```

10-Encontrar los productos que nunca fueron incluidos en un pedido.

```
SELECT ProductName
FROM Products
WHERE ProductID NOT IN (
  SELECT DISTINCT ProductID FROM [Order Details]
);
```

11-Mostrar los clientes que más gastaron, considerando precio × cantidad por pedido.

```
SELECT TOP 5 c.CompanyName, SUM(od.Quantity * od.UnitPrice) AS TotalSpent
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.CompanyName
ORDER BY TotalSpent DESC;
```

12-Crear una consulta que muestre los productos clasificados como Bajo, Medio, Alto según su precio (usando CASE).

```
SELECT ProductName, Price,
  CASE
    WHEN Price < 20 THEN 'Bajo'
    WHEN Price BETWEEN 20 AND 50 THEN 'Medio'
    ELSE 'Alto'
  END AS PrecioCategoria
FROM Products;
```

13-Identificar el país que generó más ingresos en total.

```
SELECT TOP 1 o.ShipCountry, SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
FROM Orders o
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY o.ShipCountry
ORDER BY TotalRevenue DESC;
```

14-Mostrar el tiempo promedio (en días) entre la fecha del pedido y la fecha de envío.

```
SELECT AVG(DATEDIFF(DAY, OrderDate, ShippedDate)) AS AvgDeliveryDays
FROM Orders
WHERE ShippedDate IS NOT NULL;
```

15-Usar una subconsulta para listar productos cuyo precio es mayor al promedio global.

```
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

16-Mostrar las órdenes con más de 3 productos diferentes.

```
SELECT OrderID, COUNT(DISTINCT ProductID) AS ProductCount
FROM [Order Details]
GROUP BY OrderID
HAVING COUNT(DISTINCT ProductID) > 3;
```

17-Mostrar cuántos productos tiene cada proveedor, y filtrar los que tienen más de 5.

```
SELECT s.CompanyName, COUNT(*) AS TotalProducts
FROM Suppliers s
JOIN Products p ON s.SupplierID = p.SupplierID
GROUP BY s.CompanyName
HAVING COUNT(*) > 5;
```

18-Usar un JOIN entre empleados y sus territorios, agrupando por región.

```
SELECT r.RegionDescription, COUNT(DISTINCT et.EmployeeID) AS TotalEmployees
FROM Region r
JOIN Territories t ON r.RegionID = t.RegionID
JOIN EmployeeTerritories et ON t.TerritoryID = et.TerritoryID
GROUP BY r.RegionDescription;
```

19-Calcular el porcentaje de productos sin stock (0 unidades).

```
SELECT
  CAST(SUM(CASE WHEN UnitsInStock = 0 THEN 1 ELSE 0 END) * 100.0 / COUNT(*) AS
DECIMAL(5,2)) AS PercentOutOfStock
FROM Products;
```

20-Listar las 10 órdenes más valiosas (precio × cantidad), incluyendo el cliente y la fecha.

```
SELECT TOP 10 o.OrderID, c.CompanyName, o.OrderDate,
      SUM(od.Quantity * od.UnitPrice) AS TotalOrderValue
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY o.OrderID, c.CompanyName, o.OrderDate
ORDER BY TotalOrderValue DESC;
```

🥷 20 Desafíos SQL NIVEL NINJA

1-Top 3 productos más rentables (por unidad vendida) considerando el precio y el volumen total vendido.

```
SELECT TOP 3
    p.ProductName,
    SUM(od.Quantity) AS TotalSold,
    p.UnitPrice,
    SUM(od.UnitPrice * od.Quantity) AS TotalRevenue
FROM Products p
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY p.ProductName, p.UnitPrice
ORDER BY TotalRevenue DESC;
```

2-Clientes no rentables: identificar aquellos cuyos pedidos frecuentes son de bajo volumen total (costosos de mantener).

```
SELECT
    c.CustomerID,
    COUNT(o.OrderID) AS TotalOrders,
    SUM(od.Quantity * od.UnitPrice) AS TotalSpent
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.CustomerID
HAVING COUNT(o.OrderID) >= 5 AND SUM(od.Quantity * od.UnitPrice) < 1000;
```

3-Calcular el margen de ganancia promedio por categoría (si asumimos que el costo es el 60% del precio de lista).

```
SELECT
    c.CategoryName,
    AVG(p.UnitPrice * 0.4) AS AvgProfitMargin
FROM Products p
JOIN Categories c ON p.CategoryID = c.CategoryID
GROUP BY c.CategoryName;
```

4-Detectar clientes estacionales que solo compran en ciertas épocas del año.

```
SELECT
    c.CustomerID,
    DATENAME(QUARTER, o.OrderDate) AS Quarter,
    COUNT(DISTINCT DATENAME(QUARTER, o.OrderDate)) AS UniqueQuarters
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, DATENAME(QUARTER, o.OrderDate)
HAVING COUNT(DISTINCT DATENAME(QUARTER, o.OrderDate)) = 1;
```

5-Determinar qué productos deberían ser discontinuados por baja rotación y bajo valor de venta.

```
SELECT
    p.ProductName,
    SUM(od.Quantity) AS TotalSold,
    AVG(od.UnitPrice) AS AvgPrice
FROM Products p
LEFT JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY p.ProductName
HAVING SUM(od.Quantity) < 50 AND AVG(od.UnitPrice) < 10;
```

📦 LOGÍSTICA Y OPERACIONES

1-Calcular el promedio de demora entre el pedido y el envío, y detectar a los 3 empleados más eficientes.

```
SELECT TOP 3
    e.FirstName + ' ' + e.LastName AS EmployeeName,
    AVG(DATEDIFF(DAY, o.OrderDate, o.ShippedDate)) AS AvgDelayDays
FROM Orders o
JOIN Employees e ON o.EmployeeID = e.EmployeeID
WHERE o.ShippedDate IS NOT NULL
GROUP BY e.FirstName, e.LastName
ORDER BY AvgDelayDays ASC;
```

2-Identificar los proveedores que entregan productos más vendidos y proponer una estrategia de diversificación.

```
SELECT
    s.CompanyName AS Supplier,
    SUM(od.Quantity) AS TotalUnitsSold
FROM Suppliers s
JOIN Products p ON s.SupplierID = p.SupplierID
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY s.CompanyName
ORDER BY TotalUnitsSold DESC;
```

3-Detectar órdenes que llegaron tarde (por ejemplo, más de 5 días desde la fecha de orden), y agruparlas por empleado.

```
SELECT
    e.FirstName + ' ' + e.LastName AS EmployeeName,
    COUNT(o.OrderID) AS LateOrders
FROM Orders o
JOIN Employees e ON o.EmployeeID = e.EmployeeID
WHERE DATEDIFF(DAY, o.OrderDate, o.ShippedDate) > 5
GROUP BY e.FirstName, e.LastName
ORDER BY LateOrders DESC;
```

4-Evaluar la distribución geográfica de pedidos: ¿qué regiones generan más volumen logístico y requieren más inversión?

```
SELECT ShipCountry, COUNT(*) AS TotalOrders
FROM Orders
GROUP BY ShipCountry
ORDER BY TotalOrders DESC;
```

5-Identificar los productos más solicitados que podrían generar cuellos de botella logísticos por su alta demanda.

```
SELECT TOP 10
    p.ProductName,
    SUM(od.Quantity) AS TotalDemand
FROM Products p
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY p.ProductName
ORDER BY TotalDemand DESC;
```

📈 INTELIGENCIA COMERCIAL

1-Determinar los 5 clientes más valiosos por ingreso neto anual, incluyendo evolución año a año.

```
SELECT TOP 5 WITH TIES
    c.CompanyName,
    YEAR(o.OrderDate) AS Year,
    SUM(od.Quantity * od.UnitPrice) AS AnnualRevenue
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.CompanyName, YEAR(o.OrderDate)
ORDER BY SUM(od.Quantity * od.UnitPrice) DESC;
```

3-Identificar patrones de cross-selling: productos que suelen comprarse juntos en un mismo pedido.

```
SELECT
    od1.ProductID AS ProductA,
    od2.ProductID AS ProductB,
    COUNT(*) AS TimesBoughtTogether
FROM [Order Details] od1
JOIN [Order Details] od2 ON od1.OrderID = od2.OrderID AND od1.ProductID < od2.ProductID
GROUP BY od1.ProductID, od2.ProductID
ORDER BY TimesBoughtTogether DESC;
```

4-Evaluar si hay empleados con baja conversión de pedidos por cliente asignado, y sugerir entrenamiento.

```
SELECT
    e.EmployeeID,
    e.FirstName + ' ' + e.LastName AS EmployeeName,
    COUNT(DISTINCT o.CustomerID) AS ClientsManaged,
    COUNT(o.OrderID) AS TotalOrders,
    CAST(COUNT(o.OrderID) AS FLOAT) / NULLIF(COUNT(DISTINCT o.CustomerID), 0) AS
OrdersPerClient
FROM Employees e
JOIN Orders o ON e.EmployeeID = o.EmployeeID
GROUP BY e.EmployeeID, e.FirstName, e.LastName
HAVING COUNT(DISTINCT o.CustomerID) > 5
ORDER BY OrdersPerClient ASC;
```

5-Usar funciones de ventana para detectar el crecimiento de pedidos mensuales por cliente (tendencia positiva o negativa).

```
SELECT
    c.CustomerID,
    FORMAT(o.OrderDate, 'yyyy-MM') AS OrderMonth,
    COUNT(o.OrderID) AS MonthlyOrders,
    LAG(COUNT(o.OrderID)) OVER (PARTITION BY c.CustomerID ORDER BY
FORMAT(o.OrderDate, 'yyyy-MM')) AS PreviousMonthOrders
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, FORMAT(o.OrderDate, 'yyyy-MM')
ORDER BY c.CustomerID, OrderMonth;
```

6-Clasificar los clientes en segmentos: Top Tier, Medio, Bajo, según facturación total.

```
SELECT
   c.CustomerID,
   c.CompanyName,
   SUM(od.Quantity * od.UnitPrice) AS TotalSpent,
   CASE
      WHEN SUM(od.Quantity * od.UnitPrice) >= 10000 THEN 'Top Tier'
      WHEN SUM(od.Quantity * od.UnitPrice) BETWEEN 3000 AND 9999 THEN 'Medio'
      ELSE 'Bajo'
   END AS Segmento
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.CustomerID, c.CompanyName
ORDER BY TotalSpent DESC;
```

🤖 DECISIONES ESTRATÉGICAS

1-Proponer una política de precios diferenciados por categoría, con base en su volumen de ventas y valor promedio.

```
SELECT
   c.CategoryName,
   SUM(od.Quantity) AS TotalUnitsSold,
   AVG(p.UnitPrice) AS AvgUnitPrice
FROM Categories c
JOIN Products p ON c.CategoryID = p.CategoryID
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY c.CategoryName
ORDER BY TotalUnitsSold DESC;
```

2-Detectar productos con alta estacionalidad y margen bajo para redefinir estrategias de marketing o inventario.

```
SELECT
   p.ProductName,
   DATENAME(MONTH, o.OrderDate) AS OrderMonth,
   COUNT(*) AS MonthlyOrders,
```

```
    AVG(p.UnitPrice * 0.4) AS AvgProfit
FROM Products p
JOIN [Order Details] od ON p.ProductID = od.ProductID
JOIN Orders o ON od.OrderID = o.OrderID
GROUP BY p.ProductName, DATENAME(MONTH, o.OrderDate)
HAVING AVG(p.UnitPrice * 0.4) < 10
ORDER BY p.ProductName, MonthlyOrders DESC;
```

3-Evaluar qué categorías deberían recibir más presupuesto de promoción por su rentabilidad y rotación.

```
SELECT
    c.CategoryName,
    SUM(od.Quantity * od.UnitPrice) AS TotalRevenue,
    COUNT(od.OrderID) AS TotalOrders
FROM Categories c
JOIN Products p ON c.CategoryID = p.CategoryID
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY c.CategoryName
ORDER BY TotalRevenue DESC, TotalOrders DESC;
```

4-Simular una política de descuento por fidelidad: calcular cuánto se dejaría de ganar si a los top 10 clientes se les da un 10% de descuento.

```
WITH GastoClientes AS (
  SELECT
    c.CustomerID,
    SUM(od.Quantity * od.UnitPrice) AS TotalSpent
  FROM Customers c
  JOIN Orders o ON c.CustomerID = o.CustomerID
  JOIN [Order Details] od ON o.OrderID = od.OrderID
  GROUP BY c.CustomerID
)
SELECT TOP 10
    gc.CustomerID,
    gc.TotalSpent,
    gc.TotalSpent * 0.10 AS DiscountAmount,
    gc.TotalSpent - (gc.TotalSpent * 0.10) AS NewRevenue
FROM GastoClientes gc
ORDER BY gc.TotalSpent DESC;
```

5-Proponer una expansión territorial basada en los países con más clientes activos, comparado con ingresos por país.

PARTE | Países con más clientes activos

```
SELECT Country, COUNT(*) AS ActiveClients
FROM Customers
GROUP BY Country
ORDER BY ActiveClients DESC;
```

PARTE II Ingresos por país

```
SELECT
   o.ShipCountry,
   SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
FROM Orders o
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY o.ShipCountry
ORDER BY TotalRevenue DESC;
```

Consulta para priorizar expansión territorial (clientes activos + ingresos por país)

```
SELECT
   c.Country,
   COUNT(DISTINCT c.CustomerID) AS ActiveClients,
   SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.Country
ORDER BY TotalRevenue DESC, ActiveClients DESC;
```

1. Construcción de Variable de Recencia, Frecuencia y Monto (RFM) Para cada cliente, calcular:

La fecha de su último pedido (recencia)

Cantidad total de pedidos (frecuencia)

Total gastado (monto)

Objetivo: preparar una tabla base para modelos de segmentación de clientes.

```
import sqlite3
import pandas as pd

# Conectar a la base de datos SQLite
```

```
conn = sqlite3.connect("northwind.db")

# Traer datos necesarios: CustomerID, OrderID, OrderDate, TotalGastado
query = """
SELECT
    c.CustomerID,
    o.OrderID,
    o.OrderDate,
    SUM(od.UnitPrice * od.Quantity) AS OrderTotal
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY o.OrderID
"""

df_orders = pd.read_sql_query(query, conn)

# Asegurar que OrderDate esté en formato fecha
df_orders['OrderDate'] = pd.to_datetime(df_orders['OrderDate'])

# Fecha de referencia para recencia (puede ser el último día de pedidos +1)
reference_date = df_orders['OrderDate'].max() + pd.Timedelta(days=1)

# Agrupar por cliente y calcular R, F, M
rfm = df_orders.groupby('CustomerID').agg(
    Recency = ('OrderDate', lambda x: (reference_date - x.max()).days),
    Frequency = ('OrderID', 'nunique'),
    Monetary = ('OrderTotal', 'sum')
).reset_index()

# Mostrar RFM
print(rfm.head())
```

EXTRA - DESPUES PODES GUARDAR LA TABLA COMO DATAFRAME Y DE AHI USAR KMEANS POR EJEMPLO Y SEGMENTAR

```
rfm.to_sql("RFM_Table", conn, if_exists="replace", index=False)
```

EJERCICIOS II

2. Tendencia de ventas mensuales por categoría Mostrar el volumen de ventas (total y unidades) por categoría mes a mes en los últimos 2 años.

```
import sqlite3
import pandas as pd

# Conectar a la base Northwind
conn = sqlite3.connect("northwind.db")

# Consulta SQL: ventas por producto con fecha y categoría
query = """
SELECT
    c.CategoryName,
    strftime('%Y-%m', o.OrderDate) AS OrderMonth,
    SUM(od.Quantity) AS TotalUnits,
    SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
FROM [Order Details] od
JOIN Orders o ON od.OrderID = o.OrderID
JOIN Products p ON od.ProductID = p.ProductID
JOIN Categories c ON p.CategoryID = c.CategoryID
WHERE o.OrderDate >= date('now', '-2 years')
GROUP BY c.CategoryName, OrderMonth
ORDER BY OrderMonth ASC;
"""
df_monthly = pd.read_sql_query(query, conn)

# Mostrar resultados
print(df_monthly.head())
```

SI QUIEREN VISUALIZARLO -

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
pivot = df_monthly.pivot(index='OrderMonth', columns='CategoryName', values='TotalRevenue')
pivot.plot(kind='line', marker='o')
plt.title('Tendencia mensual de ventas por categoría')
plt.xlabel('Mes')
plt.ylabel('Ingresos ($)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

EJERCICIO III
3. Análisis de cohortes Agrupar clientes por mes de primer pedido y analizar su retención mensual durante 6 meses posteriores.

PRIMERO -EXTRAER DATOS CLIENTES Y SUS ORDENES

```
import sqlite3
import pandas as pd

# Conectar a la base
conn = sqlite3.connect("northwind.db")

# Extraer CustomerID y fecha de cada orden
query = """
SELECT
    CustomerID,
    DATE(OrderDate) AS OrderDate
FROM Orders
WHERE CustomerID IS NOT NULL
"""

df_orders = pd.read_sql_query(query, conn)
df_orders['OrderDate'] = pd.to_datetime(df_orders['OrderDate'])
```

SEGUNDO - COHORTE , CALCULAR PRIMER PEDIDO DE CADA CLIENTE

```
# Mes del primer pedido por cliente
df_orders['CohortMonth'] =
df_orders.groupby('CustomerID')['OrderDate'].transform('min').dt.to_period('M')
df_orders['OrderMonth'] = df_orders['OrderDate'].dt.to_period('M')
```

TERCERO - CALCULAR INDICE

```
df_orders['CohortIndex'] = (df_orders['OrderMonth'] - df_orders['CohortMonth']).apply(lambda x:
x.n)
```

CUARTO RETENCIÓN CANT CLIENTES ÚNICOS POR COHORTE

```
cohort_counts = df_orders.groupby(['CohortMonth',
'CohortIndex'])['CustomerID'].nunique().reset_index()

cohort_pivot = cohort_counts.pivot(index='CohortMonth', columns='CohortIndex',
values='CustomerID')
cohort_pivot = cohort_pivot.fillna(0).astype(int)
print(cohort_pivot)
```

SI QUIEREN VISUALIZAR

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
sns.heatmap(cohort_pivot, annot=True, fmt="d", cmap="YlGnBu")
plt.title('Análisis de Cohortes - Retención 6 meses')
plt.xlabel('Mes desde primer pedido')
plt.ylabel('Mes de cohorte')
plt.tight_layout()
plt.show()
```

4. Detección de productos “anómalos” Identificar productos con:

Alto precio pero baja venta

Bajo precio y alta rotación

Ideal para detección de outliers comerciales.

PASO 1 - TRAER PRECIOS Y CANT VENDIDA

```
query = """
SELECT
    p.ProductID,
    p.ProductName,
    p.UnitPrice,
    SUM(od.Quantity) AS TotalUnitsSold
FROM Products p
LEFT JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY p.ProductID, p.ProductName, p.UnitPrice
"""
df = pd.read_sql_query(query, conn)
```

PASO 2 - CLASIFICAR BAJOS Y ALTOS

```
# Calcular los percentiles para UnitPrice y TotalUnitsSold
high_price = df['UnitPrice'].quantile(0.75)
low_price = df['UnitPrice'].quantile(0.25)

high_sales = df['TotalUnitsSold'].quantile(0.75)
low_sales = df['TotalUnitsSold'].quantile(0.25)

# Agregar etiquetas
```

```
def clasificar_producto(row):
    if row['UnitPrice'] >= high_price and row['TotalUnitsSold'] <= low_sales:
        return 'Alto precio / Baja venta'
    elif row['UnitPrice'] <= low_price and row['TotalUnitsSold'] >= high_sales:
        return 'Bajo precio / Alta rotación'
    else:
        return 'Normal'

df['Anomalia'] = df.apply(clasificar_producto, axis=1)
```

SI QUIEREN VISUALIZAR

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='UnitPrice', y='TotalUnitsSold', hue='Anomalia', style='Anomalia',
s=100)
plt.title('Detección de productos anómalos')
plt.xlabel('Precio unitario ($)')
plt.ylabel('Unidades vendidas')
plt.grid(True)
plt.tight_layout()
plt.show()
```

EJERCICIO #5

5. Construcción de red de productos frecuentemente vendidos juntos Obtener pares de productos que aparecen en la misma orden al menos 5 veces → base para análisis de mercado o recomendación.

PASO 1 - Generar par/ producto.

```
query = """
SELECT
    od1.ProductID AS ProductA,
    od2.ProductID AS ProductB,
    COUNT(*) AS TimesBoughtTogether
FROM [Order Details] od1
JOIN [Order Details] od2
    ON od1.OrderID = od2.OrderID
    AND od1.ProductID < od2.ProductID
GROUP BY ProductA, ProductB
```

```
HAVING TimesBoughtTogether >= 5
ORDER BY TimesBoughtTogether DESC;
"""

df_pairs = pd.read_sql_query(query, conn)
```

PASO 2 - AGREGAR NOMBRE

```
# Obtener diccionario de ID → Nombre
df_products = pd.read_sql_query("SELECT ProductID, ProductName FROM Products", conn)

# Merge para obtener nombres legibles
df_pairs = df_pairs.merge(df_products, left_on='ProductA', right_on='ProductID')
df_pairs = df_pairs.merge(df_products, left_on='ProductB', right_on='ProductID', suffixes=('_A',
'_B'))

# Selección final
df_pairs = df_pairs[['ProductName_A', 'ProductName_B', 'TimesBoughtTogether']]
df_pairs.rename(columns={
    'ProductName_A': 'Producto A',
    'ProductName_B': 'Producto B',
    'TimesBoughtTogether': 'Veces Comprados Juntos'
}, inplace=True)

print(df_pairs.head())
```

PASO 3 - VISUALIZAR
(OJO ES RED)

```
import networkx as nx
import matplotlib.pyplot as plt

# Crear grafo
G = nx.Graph()

for _, row in df_pairs.iterrows():
    G.add_edge(row['Producto A'], row['Producto B'], weight=row['Veces Comprados Juntos'])

# Dibujar red
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G, k=0.3)
edges = G.edges(data=True)
nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=1500, edge_color='gray')
nx.draw_networkx_edge_labels(G, pos, edge_labels={(u,v): d['weight'] for u,v,d in edges})
```

```
plt.title('Red de productos frecuentemente vendidos juntos (≥ 5 veces)')
plt.tight_layout()
plt.show()
```

EJERCICIO 6

6. Flujo de pedidos por región Determinar la progresión mensual de pedidos por región geográfica (usando Customers y Orders) → para visualización tipo heatmap o geolocalización

```
import pandas as pd
import sqlite3

conn = sqlite3.connect("northwind.db")

query = """
SELECT
   c.Country AS Region,
   strftime('%Y-%m', o.OrderDate) AS OrderMonth,
   COUNT(o.OrderID) AS TotalOrders
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.OrderDate IS NOT NULL
GROUP BY Region, OrderMonth
ORDER BY Region, OrderMonth;
"""

df_flow = pd.read_sql_query(query, conn)
```

PASO 2 - PIVOT PARA EL HEATMAP

```
heatmap_data = df_flow.pivot(index='Region', columns='OrderMonth',
values='TotalOrders').fillna(0).astype(int)
print(heatmap_data)
```

PASO 3- VISUALIZAR

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 8))
sns.heatmap(heatmap_data, annot=True, fmt='d', cmap="YlGnBu")
```

```
plt.title("Flujo mensual de pedidos por región")
plt.xlabel("Mes")
plt.ylabel("Región")
plt.tight_layout()
plt.show()
```

EJERCICIO 7

7. Tiempo promedio y desviación estándar de entrega por país Calcular la demora entre OrderDate y ShippedDate, agrupado por país → para análisis de eficiencia logística.

```
import pandas as pd
import sqlite3

conn = sqlite3.connect("northwind.db")

query = """
SELECT
   o.ShipCountry AS Country,
   AVG(julianday(o.ShippedDate) - julianday(o.OrderDate)) AS AvgDeliveryDays,
   ROUND(STDDEV(julianday(o.ShippedDate) - julianday(o.OrderDate)), 2) AS
StdDevDeliveryDays
FROM Orders o
WHERE o.ShippedDate IS NOT NULL AND o.OrderDate IS NOT NULL
GROUP BY o.ShipCountry
ORDER BY AvgDeliveryDays ASC;
"""

df_logistics = pd.read_sql_query(query, conn)
print(df_logistics)
```

EJERCICIO 8

8. Segmentación de clientes por comportamiento de compra Generar clústers de clientes basados en:
Frecuencia
Monto total
Categorías más compradas
Preparar el dataset en SQL antes de pasarlo a Python para clustering.

PASO 1 - FRECUENCIA Y MONTO

```
SELECT
   c.CustomerID,
   COUNT(DISTINCT o.OrderID) AS Frequency,
```

```
    SUM(od.Quantity * od.UnitPrice) AS Monetary
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN [Order Details] od ON o.OrderID = od.OrderID
GROUP BY c.CustomerID;
```

PASO 2 AGREGAR CAT MAS COMPRADA

```
WITH CategoriaPorCliente AS (
    SELECT
        c.CustomerID,
        cat.CategoryName,
        SUM(od.Quantity) AS TotalUnits
    FROM Customers c
    JOIN Orders o ON c.CustomerID = o.CustomerID
    JOIN [Order Details] od ON o.OrderID = od.OrderID
    JOIN Products p ON od.ProductID = p.ProductID
    JOIN Categories cat ON p.CategoryID = cat.CategoryID
    GROUP BY c.CustomerID, cat.CategoryName
),
TopCategoria AS (
    SELECT CustomerID, CategoryName
    FROM (
        SELECT *,
            ROW_NUMBER() OVER (PARTITION BY CustomerID ORDER BY TotalUnits DESC)
AS rn
        FROM CategoriaPorCliente
    )
    WHERE rn = 1
)

SELECT
    base.CustomerID,
    base.Frequency,
    base.Monetary,
    tc.CategoryName AS TopCategory
FROM (
    SELECT
        c.CustomerID,
        COUNT(DISTINCT o.OrderID) AS Frequency,
        SUM(od.Quantity * od.UnitPrice) AS Monetary
    FROM Customers c
    JOIN Orders o ON c.CustomerID = o.CustomerID
    JOIN [Order Details] od ON o.OrderID = od.OrderID
```

```
    GROUP BY c.CustomerID
) AS base
LEFT JOIN TopCategoria tc ON base.CustomerID = tc.CustomerID;
```

PASO 3 - PYTHON PARA CLUSTERIZAR

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

# Cargar el dataset
df = pd.read_sql_query("SELECT * FROM ClientesSegmentacion", conn)

# One-hot para la categoría
df_encoded = pd.get_dummies(df, columns=["TopCategory"], drop_first=True)

# Escalado y clustering
scaler = StandardScaler()
kmeans = KMeans(n_clusters=4, random_state=42)
pipeline = make_pipeline(scaler, kmeans)
pipeline.fit(df_encoded.drop(columns=['CustomerID']))

# Agregar cluster resultante
df['Cluster'] = pipeline.named_steps['kmeans'].labels_
print(df.head())
```

EJERCICIO #9

9. Normalización de variables Generar una tabla con:

UnitPrice, UnitsInStock, UnitsOnOrder

Estandarizadas (z-score) → usando subconsultas para media y desvío.

PASO 1 - LA CONSULTA Y SUB PARA MEDIA Y DESVIO

```
SELECT
    ProductID,
    ProductName,
    UnitPrice,
    UnitsInStock,
```

```
    UnitsOnOrder,

    -- Z-score UnitPrice
    (UnitPrice - (SELECT AVG(UnitPrice) FROM Products)) /
    (SELECT STDDEV(UnitPrice) FROM Products) AS z_UnitPrice,

    -- Z-score UnitsInStock
    (UnitsInStock - (SELECT AVG(UnitsInStock) FROM Products)) /
    (SELECT STDDEV(UnitsInStock) FROM Products) AS z_UnitsInStock,

    -- Z-score UnitsOnOrder
    (UnitsOnOrder - (SELECT AVG(UnitsOnOrder) FROM Products)) /
    (SELECT STDDEV(UnitsOnOrder) FROM Products) AS z_UnitsOnOrder

FROM Products;
```

USANDO Z-SCORE

```
from scipy.stats import zscore
import pandas as pd
import sqlite3

conn = sqlite3.connect("northwind.db")

df = pd.read_sql_query("""
SELECT ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder
FROM Products
""", conn)

# Calcular z-score
cols = ['UnitPrice', 'UnitsInStock', 'UnitsOnOrder']
df_z = df.copy()
df_z[[f'z_{col}' for col in cols]] = df[cols].apply(zscore)

print(df_z.head())
```

EJERCICIO 1O

10. Variables para modelo de predicción de abandono Crear dataset de clientes activos/inactivos con variables como:

Días desde último pedido

N° de pedidos en los últimos 90 días

% de pedidos cancelados

Para usar como input de un modelo predictivo.

```
import pandas as pd
import sqlite3
from datetime import datetime

conn = sqlite3.connect("northwind.db")

# Tomamos todos los pedidos con fecha
query = """
SELECT
    CustomerID,
    OrderID,
    OrderDate,
    ShippedDate
FROM Orders
WHERE CustomerID IS NOT NULL
"""

df = pd.read_sql_query(query, conn)
df['OrderDate'] = pd.to_datetime(df['OrderDate'])
df['ShippedDate'] = pd.to_datetime(df['ShippedDate'])

# Fecha de análisis (hoy o corte arbitrario)
reference_date = df['OrderDate'].max() + pd.Timedelta(days=1)
```