INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Proyecto Fragmentación.

Alumnos:

Bazaldua Montoya Jose Juan

Benites Garcilazo Luis Fernando

Gallegos Lozano Karen Scarlett

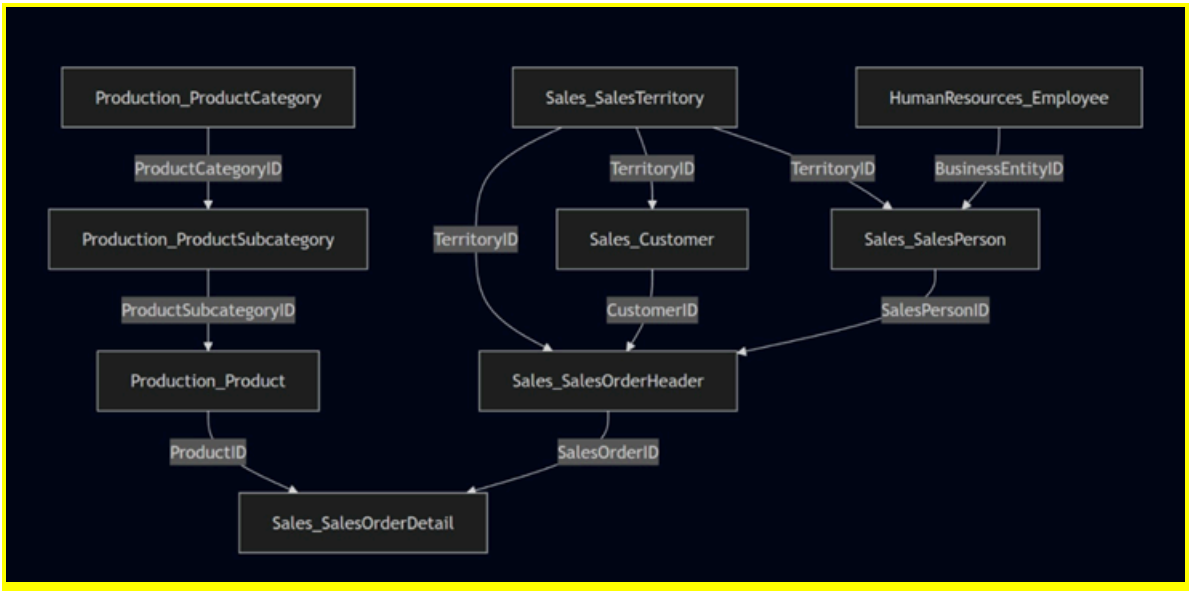
Sanchez Gallardo Janeth

Unidad de Aprendizaje: Base de Datos Distribuidas

Profesor: De la Cruz Sosa Carlos

Grafo relacional:..... 3
Distribución de Fragmentos:..... 4
Tablas para fragmentar:..... 5
Conexión remota física.....7
SQL.....7
Base coordinadora..... 7
Diccionario de fragmentación..... 8
Inserción de fragmentación..... 8
Norte América..... 9
Europa.....12
Consultas remotas:.....14
EXEC de los SP..... 42
API:.....47
API AWS:..... 60
Preguntas realizadas por el profesor..... 71
¿Qué ventaja tiene el uso de una tabla en la consulta?. 71
¿Qué desventaja tiene el uso de una tabla en la consulta?..... 71
¿Por qué no utilizar una variable de tabla a nivel memoria? ¿se puede?.....71
Ejercicios resueltos por cada integrante y Conclusiones individuales:..... 72

Grafo relacional:



Nuestra fragmentación tiene como punto de partida SalesTerritory, que fragmenta:

NAàSalesTerritory = ‘North America’

EUàSalesTerritory = ‘Europe’

El grafo cuenta con 4 fragmentos de SalesTerritory los cuales son:

- Sales_SalesTerrritory
- Sales_SalesCustomer
- Sales_SalesPerson
- Sales_SalesOrderHeader

Fragmentos desde Sales_SalesTerritory:

Desde la tabla Sales_SalesTerritory, se generan cuatro fragmentos que se conectan a diferentes tablas. Estos fragmentos están indicados de la siguiente manera:

- **Fragmento 1:** Sales_SalesTerritory. Es la tabla principal que contiene información sobre las regiones de ventas. Esta tabla se fragmenta para manejar los datos distribuidos según el territorio (por ejemplo, Norteamérica y Europa).
- **Fragmento 2:** Sales_Customer. Esta tabla contiene datos sobre los clientes. Está relacionada con Sales_SalesTerritory a través del atributo TerritoryID. Esto significa que los clientes se distribuyen según el territorio en el que operan.

- **Fragmento 3:** Sales_SalesPerson. Esta tabla contiene información sobre los empleados de ventas. También se conecta con Sales_SalesTerritory mediante el TerritoryID. De esta manera, los empleados de ventas se asocian con sus respectivas regiones.
- **Fragmento 4:** Sales_SalesOrderHeader Esta tabla contiene los encabezados de los pedidos de ventas. Se conecta con Sales_SalesTerritory a través del TerritoryID, lo que permite que los pedidos se asignen a las regiones correspondientes.

Relaciones Clave en el Modelo:

- Sales_SalesTerritory: Es la tabla base desde donde se generan las fragmentaciones y relaciones. Tiene conexiones directas con varias tablas, lo que indica que está involucrada en la distribución de datos entre diferentes regiones (territorios).
- Sales_Customer y Sales_SalesPerson: Ambas tablas están relacionadas con Sales_SalesTerritory a través del campo TerritoryID, lo que permite asignar a los clientes y empleados de ventas a territorios específicos.
- Sales_SalesOrderHeader: Se conecta con Sales_SalesTerritory y, a su vez, con la tabla Sales_SalesOrderDetail a través del SalesOrderID y ProductID. Esto indica que los pedidos de ventas se gestionan y distribuyen según las regiones de ventas.
- Production_ProductCategory y Production_ProductSubcategory: Estas tablas están relacionadas con los productos y sus categorías. Aunque no están directamente conectadas con la tabla principal Sales_SalesTerritory, se conectan con Sales_SalesOrderDetail a través del ProductID, lo que permite rastrear qué productos se han vendido en cada territorio.
- HumanResources_Employee: Está vinculada a la tabla Sales_SalesPerson mediante el BusinessEntityID. Esto establece una relación entre los empleados de ventas y sus datos de recursos humanos.

Distribución de Fragmentos:

Cada uno de estos fragmentos puede ser gestionado por diferentes servidores y bases de datos, según el sistema distribuido. Por ejemplo:

- Sales_SalesTerritory se fragmenta según la región.
- Sales_Customer y Sales_SalesPerson se agrupan por territorio, de modo que la información de clientes y empleados está disponible localmente en el fragmento correspondiente a su territorio.
- Sales_SalesOrderHeader se distribuye también por territorio, lo que asegura que las órdenes de ventas se gestionen según su región.

Beneficios de la Fragmentación:

- Optimización del rendimiento: Al distribuir los datos en fragmentos según las regiones, las consultas sobre clientes, ventas, empleados y pedidos se realizan de forma más eficiente, ya que los datos relevantes están más cerca de la ubicación de uso.

- Escalabilidad: El sistema puede crecer sin perder rendimiento al añadir más regiones o fragmentos, simplemente agregando más servidores o bases de datos.

Tablas para fragmentar:

Las tablas que se fragmentarían, basadas en el modelo y las relaciones, son las siguientes:

1. Sales_SalesTerritory:

- Esta es la tabla principal que define los territorios de ventas. Es el punto de partida para la fragmentación horizontal.
- Fragmentación: Sales_SalesTerritory se fragmenta horizontalmente por región.

2. Sales_SalesOrderHeader:

- Esta tabla contiene la cabecera de las órdenes de venta y está directamente relacionada con Sales_SalesTerritory a través del TerritoryID.
- Fragmentación: Al estar asociada con Sales_SalesTerritory, puede fragmentarse en función del TerritoryID, es decir, los registros de pedidos se distribuirían entre los fragmentos correspondientes a cada región.

3. Sales_Customer:

- Esta tabla contiene los datos de los clientes y está relacionada con Sales_SalesTerritory a través del TerritoryID.
- Fragmentación: Se fragmenta según el TerritoryID ,asignando a los clientes a los fragmentos correspondientes a su territorio.

4. Sales_SalesPerson:

- Esta tabla contiene información sobre los vendedores y está relacionada con Sales_SalesTerritory mediante el TerritoryID.
- Fragmentación: Se fragmenta por TerritoryID, asegurando que los datos de los empleados de ventas estén organizados según la región a la que pertenecen.

5. Sales_SalesOrderDetail:

- Esta tabla contiene los detalles de las órdenes de ventas, como los productos comprados y las cantidades. Está vinculada a Sales_SalesOrderHeader mediante el SalesOrderID.
- Fragmentación: Al estar asociada con Sales_SalesOrderHeader, esta tabla podría seguir la misma estrategia de fragmentación, distribuyéndose por territorio de ventas (es decir, fragmentarse en función de los registros de Sales_SalesOrderHeader, que a su vez se fragmentan por TerritoryID).

6. Production_ProductCategory:

- Contiene la categoría de los productos.
- Fragmentación: Aunque no se menciona explícitamente en el diagrama, podría fragmentarse en función de la categoría de productos, especialmente si se requiere acceder a categorías específicas por territorio.

7. Production_ProductSubcategory:

- Contiene la subcategoría de los productos.
- Fragmentación: Al igual que Production_ProductCategory, esta tabla podría fragmentarse según las subcategorías de productos, aunque su fragmentación dependerá del acceso necesario a los datos y si está directamente relacionado con los territorios de ventas.

8. Production_Product:

- Contiene los productos que están disponibles para la venta.
- Fragmentación: Production_Product puede fragmentarse en función de las subcategorías o categorías de productos o, en algunos casos, puede replicarse si se necesita acceso frecuente a la información de productos en todas las regiones.

9. HumanResources_Employee:

- Contiene información sobre los empleados de la empresa.
- Fragmentación: Esta tabla podría no necesitar fragmentación, pero si se deseara organizar la información por territorio, se podría asociar a Sales_SalesPerson y fragmentarse por TerritoryID, aunque esto dependería del acceso a los datos de los empleados por territorio.

Fragmentario horizontal:

- Sales_SalesTerritory se fragmenta horizontalmente en función del atributo CountryRegionCode (código de la región del país), lo que crea fragmentos para Norteamérica y Europa.
    - Fragmento Norteamérica: Los registros donde CountryRegionCode = 'North America' se almacenan en el fragmento SRV_NA.AW_NA.
    - Fragmento Europa: Los registros donde CountryRegionCode = 'Europe' se almacenan en el fragmento SRV_NA.AW_EU.

Esta fragmentación horizontal distribuye los datos de Sales_SalesTerritory en función de la región geográfica, lo cual es útil para mejorar la consulta y la eficiencia en el manejo de los datos basados en la región.

Primaria (sobre que atributos esta fragmentada):

Sales_SalesOrderHeader como Fragmentación Primaria:

- Sales_SalesOrderHeader se considera como una fragmentación primaria porque tiene fragmentaciones adicionales sobre sus propios datos, y está estrechamente relacionada con las tablas de pedidos y ventas.
- La fragmentación primaria de Sales_SalesOrderHeader se puede realizar en función de ciertos atributos clave, como el SalesOrderID y posiblemente el TerritoryID o el CustomerID.
    - Dado que Sales_SalesOrderHeader está relacionado con Sales_SalesTerritory a través del TerritoryID, la fragmentación de Sales_SalesOrderHeader puede seguir la misma lógica de fragmentación por territorio.
    - Fragmentos de Sales_SalesOrderHeader:
        - Los registros de pedidos que pertenecen a Norteamérica se almacenan en el fragmento SRV_NA.AW_NA.
        - Los registros de pedidos que pertenecen a Europa se almacenan en el fragmento SRV_NA.AW_EU.

Este enfoque asegura que las órdenes de ventas se distribuyan de manera consistente con los territorios correspondientes, facilitando el acceso a los datos de las órdenes según la región.

Conexión remota física

De acuerdo a lo solicitado para el proyecto, realizamos una conexión remota de dos servidores y dos clientes dentro del laboratorio de telemática II.

Conexión remota para los servidores, es donde se encuentra toda la información ordenada

- La máquina 15 es uno de nuestros servidores, el cual contiene las BD del diccionario y AW_NA
- La máquina 19 es el segundo servidor, el cual contiene las BD del diccionario y AW_EU

Conexión remota para los clientes, siendo el medio que usamos para acceder, seleccionar y filtrar la información que se encuentra en el servidor

- Las máquinas 6 y 17 hacen función de clientes, ejecutando los procedure y las ejecuciones de los mismos

SQL

Base coordinadora

```
IF DB_ID('diccionario_sPS') IS NOT NULL
```

```
BEGIN
        ALTER DATABASE diccionario_sPS SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
        DROP DATABASE diccionario_sPS;
END;
GO
CREATE DATABASE diccionario_sPS;
GO
USE diccionario_sPS;
GO
```

Diccionario de fragmentación. (Luis)

```
IF OBJECT_ID('dbo.diccionariodist', 'U') IS NOT NULL
        DROP TABLE dbo.diccionariodist;
GO
CREATE TABLE dbo.diccionariodist (
        id INT IDENTITY(1,1) PRIMARY KEY,
        servidor NVARCHAR(255),
        bd NVARCHAR(255),
        tabla NVARCHAR(255),
        predicado NVARCHAR(255)
);
GO
```

Inserción de fragmentación

-- AW_NA = North America

-- Europe = Europe

```
INSERT INTO dbo.diccionariodist (servidor, bd, tabla, predicado)
VALUES
        -- Región: AW_NA (North America)
```

```
        ('SRV_NA', 'AW_NA', 'SalesOrderHeader',  'AW_NA'),
        ('SRV_NA', 'AW_NA', 'SalesOrderDetail',  'AW_NA'),
        ('SRV_NA', 'AW_NA', 'Customer',          'AW_NA'),
        ('SRV_NA', 'AW_NA', 'SalesPerson',   'AW_NA'),
        ('SRV_NA', 'AW_NA', 'Employee',          'AW_NA'),
        ('SRV_NA', 'AW_NA', 'Person',      'AW_NA'),
        ('SRV_NA', 'AW_NA', 'Product',     'AW_NA'),
        -- Región: Europe
        ('SRV_EU', 'AW_EU', 'SalesOrderHeader',  'AW_EU'),
        ('SRV_EU', 'AW_EU', 'SalesOrderDetail',  'AW_EU'),
        ('SRV_EU', 'AW_EU', 'Customer',          'AW_EU'),
        ('SRV_EU', 'AW_EU', 'SalesPerson',   'AW_EU'),
        ('SRV_EU', 'AW_EU', 'Employee',          'AW_EU'),
        ('SRV_EU', 'AW_EU', 'Person',      'AW_EU'),
        ('SRV_EU', 'AW_EU', 'Product',     'AW_EU');
GO
SELECT * FROM diccionariodist;
GO
```

Norte América

```
-- CREAR BASE DE DATOS AW_NA (North America)
IF DB_ID('AW_NA') IS NOT NULL
BEGIN
        ALTER DATABASE AW_NA SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
        DROP DATABASE AW_NA;
END;
GO
CREATE DATABASE AW_NA;
GO
```

```sql
USE AW_NA;
GO

--Solo ponemos los esquemas que pide el proyecto:
-- sales.person, production, humanresources
--Territory
SELECT *
INTO Territory
FROM AdventureWorks2022.Sales.SalesTerritory;

 -- SALES.Customer
SELECT *
INTO Customer
FROM AdventureWorks2022.Sales.Customer
WHERE TerritoryID IN (
        SELECT TerritoryID
        FROM AdventureWorks2022.Sales.SalesTerritory
        WHERE [Group] = 'North America'
);
-- SALES.SalesOrderHeader
SELECT *
INTO SalesOrderHeader
FROM AdventureWorks2022.Sales.SalesOrderHeader
WHERE TerritoryID IN (
        SELECT TerritoryID
        FROM AdventureWorks2022.Sales.SalesTerritory
        WHERE [Group] = 'North America'
);
-- SALES.SalesOrderDetail
```

```
SELECT sod.*
INTO SalesOrderDetail
FROM AdventureWorks2022.Sales.SalesOrderDetail sod
JOIN AdventureWorks2022.Sales.SalesOrderHeader soh
        ON sod.SalesOrderID = soh.SalesOrderID
WHERE soh.TerritoryID IN (
        SELECT TerritoryID
        FROM AdventureWorks2022.Sales.SalesTerritory
        WHERE [Group] = 'North America'
);
-- SALES.SalesPerson
SELECT *
INTO SalesPerson
FROM AdventureWorks2022.Sales.SalesPerson sp
WHERE sp.BusinessEntityID IN (
        SELECT SalesPersonID
        FROM AdventureWorks2022.Sales.SalesOrderHeader soh
        JOIN AdventureWorks2022.Sales.SalesTerritory st
          ON soh.TerritoryID = st.TerritoryID
        WHERE st.[Group] = 'North America'
);
-- PRODUCTION.Product
SELECT *
INTO Product
FROM AdventureWorks2022.Production.Product;
-- PRODUCTION.ProductSubcategory
SELECT *
INTO ProductSubcategory
```

```
FROM AdventureWorks2022.Production.ProductSubcategory;

-- PRODUCTION.ProductCategory

SELECT *

INTO ProductCategory

FROM AdventureWorks2022.Production.ProductCategory;

-- HR.Employee

SELECT *

INTO Employee

FROM AdventureWorks2022.HumanResources.Employee;
```

Europa

```
-- CREAR BASE DE DATOS AW_EU (Europe)--

IF DB_ID('AW_EU') IS NOT NULL

BEGIN

ALTER DATABASE AW_EU SET SINGLE_USER WITH ROLLBACK IMMEDIATE;

        DROP DATABASE AW_EU;

END;

GO

CREATE DATABASE AW_EU;

GO

USE AW_EU;

GO

-- CREAR TABLAS NECESARIAS (Europe)


--Territory

SELECT *

INTO Territory

FROM AdventureWorks2022.Sales.SalesTerritory;
```

```sql
-- SALES.Customer
SELECT *
INTO CustomerAAE
FROM AdventureWorks2022.Sales.Customer
WHERE TerritoryID IN (
        SELECT TerritoryID
        FROM AdventureWorks2022.Sales.SalesTerritory
        WHERE [Group] = 'Europe'
);
-- SALES.SalesOrderHeader
SELECT *
INTO SalesOrderHeader
FROM AdventureWorks2022.Sales.SalesOrderHeader
WHERE TerritoryID IN (
        SELECT TerritoryID
        FROM AdventureWorks2022.Sales.SalesTerritory
        WHERE [Group] = 'Europe'
);
-- SALES.SalesOrderDetail
SELECT sod.*
INTO SalesOrderDetail
FROM AdventureWorks2022.Sales.SalesOrderDetail sod
JOIN AdventureWorks2022.Sales.SalesOrderHeader soh
        ON sod.SalesOrderID = soh.SalesOrderID
WHERE soh.TerritoryID IN (
        SELECT TerritoryID
        FROM AdventureWorks2022.Sales.SalesTerritory
        WHERE [Group] = 'Europe'
```

```
);

-- SALES.SalesPerson

SELECT *

INTO SalesPerson

FROM AdventureWorks2022.Sales.SalesPerson sp

WHERE sp.BusinessEntityID IN (

        SELECT SalesPersonID

        FROM AdventureWorks2022.Sales.SalesOrderHeader soh

        JOIN AdventureWorks2022.Sales.SalesTerritory st

          ON soh.TerritoryID = st.TerritoryID

        WHERE st.[Group] = 'Europe'

);

-- PRODUCTION.Product

SELECT *

INTO Product

FROM AdventureWorks2022.Production.Product;

-- HR.Employee

SELECT *

INTO Employee

FROM AdventureWorks2022.HumanResources.Employee;

-- PRODUCTION.ProductSubcategory

SELECT *

INTO ProductSubcategory

FROM AdventureWorks2022.Production.ProductSubcategory;

-- PRODUCTION.ProductCategory

SELECT *

INTO ProductCategory
FROM AdventureWorks2022.Production.ProductCategory;
```

Consultas remotas:

```
------------------------------------------------------------
-- 1. SP CONSULTAR LINKED SERVER
------------------------------------------------------------
IF OBJECT_ID('dbo.usp_ConsultarTablaLinkedServer', 'P') IS NOT NULL
	DROP PROCEDURE dbo.usp_ConsultarTablaLinkedServer;
GO

CREATE PROCEDURE dbo.usp_ConsultarTablaLinkedServer
@ServidorVinculado NVARCHAR(128),
@BaseDatos NVARCHAR(128),
@Tabla NVARCHAR(128)
AS
BEGIN
	DECLARE @SQL NVARCHAR(MAX);
	SET @SQL =
	'SELECT * FROM ' + QUOTENAME(@ServidorVinculado) + '.'
		+ QUOTENAME(@BaseDatos) + '.dbo.'
		+ QUOTENAME(@Tabla) + ';';
	EXEC sp_executesql @SQL;
END;
GO

------------------------------------------------------------
-- 2. SP DE RUTEO GENERAL
------------------------------------------------------------
IF OBJECT_ID('dbo.execquery', 'P') IS NOT NULL
	DROP PROCEDURE dbo.execquery;
```

```
GO

CREATE PROCEDURE dbo.execquery
@region NVARCHAR(50),
@tabla NVARCHAR(255),
@query NVARCHAR(MAX)
AS
BEGIN
        DECLARE @servidor NVARCHAR(255), @bd NVARCHAR(255), @SQL NVARCHAR(MAX);

        SELECT TOP 1 @servidor = servidor, @bd = bd
        FROM diccionariodist
        WHERE predicado = @region AND tabla = @tabla;

        IF @servidor IS NULL
        BEGIN
        RAISERROR('No existe esa región o tabla en el diccionario.', 16, 1);
        RETURN;
        END;

        SET @SQL =
        'SELECT * FROM ' + QUOTENAME(@servidor) + '.'
                + QUOTENAME(@bd) + '.dbo.'
                + QUOTENAME(@tabla) + ' '
                + ISNULL(@query, '') + ';';

        EXEC sp_executesql @SQL;
END;
```

```
GO

/*-----------------------------------------------------------
a) Registrar venta por Area
-----------------------------------------------------------*/

IF OBJECT_ID('dbo.usp_RegistrarVentaRegion', 'P') IS NOT NULL
        DROP PROCEDURE dbo.usp_RegistrarVentaRegion;
GO

GO

CREATE OR ALTER PROCEDURE dbo.usp_RegistrarVentaRegion
        @Region         NVARCHAR(50),   -- 'AW_NA' o 'AW_EU' (acepto 'Europe' y lo normalizo)
        @CustomerID     INT,
    @SalesPersonID INT,
    @TerritoryID    INT,
    @ProductID      INT,
    @OrderQty       INT,
    @UnitPrice      MONEY,
        @FechaOrden     DATETIME
AS
BEGIN
        SET NOCOUNT ON;   --Evita que devuelvan mesajes sobre el # de filas afectadas
        SET XACT_ABORT ON; --Si ocurre un error durante la transaccion se aborta

        -- Normalizar región
        SET @Region = CASE
```

```
            WHEN @Region IN (N'Europe', N'EU', N'Europa') THEN N'AW_EU'
            WHEN @Region IN (N'North America', N'NA', N'Norte America', N'Norte América') THEN
N'AW_NA'
            ELSE @Region
                END;
        --Busca en el diccionario el fragmento
        DECLARE @servidor SYSNAME, @bd SYSNAME;

        SELECT TOP (1)
    @servidor = servidor,
    @bd       = bd
        FROM dbo.diccionariodist
        WHERE tabla = N'SalesOrderHeader'
        AND predicado = @Region;

        IF @servidor IS NULL
        THROW 50000, 'Región no registrada en el diccionario.', 1;

        -- Asegurar colocalización Detail/Header (misma región, mismo fragmento)
        IF NOT EXISTS (
        SELECT 1
        FROM dbo.diccionariodist
        WHERE tabla = N'SalesOrderDetail'
        AND predicado = @Region
        AND servidor = @servidor
        AND bd = @bd
        )
        THROW 50001, 'SalesOrderDetail no está co-localizado con SalesOrderHeader para esa región.',
1;
```

```
        -- Derivados
        begin
        --para generar un ID
        DECLARE @SalesOrderNumber NVARCHAR(50) =
        N'SO-' + REPLACE(CONVERT(NVARCHAR(36), NEWID()), N'-', N'');
        -- Conveirte mis cantidades
        DECLARE @LineTotal NUMERIC(38,6) =
        CONVERT(NUMERIC(38,6), @OrderQty) * CONVERT(NUMERIC(38,6), @UnitPrice);

        DECLARE @OrderDateIso NVARCHAR(30) = CONVERT(NVARCHAR(30), @FechaOrden, 126);
        DECLARE @SalesOrderNumberEsc NVARCHAR(100) = REPLACE(@SalesOrderNumber, N'',
N''''); -- escape comillas

        -- Captura del ID devuelto por remoto
        IF OBJECT_ID('tempdb..#NewSO', 'U') IS NOT NULL DROP TABLE #NewSO;
        CREATE TABLE #NewSO (SalesOrderID INT NOT NULL);

  ----------------------------------------------------------------

        -- Batch que se ejecuta EN el remoto (un solo string, sin anidación)
        ----------------------------------------------------------------
DECLARE @RemoteBatch NVARCHAR(MAX) = N'
SET NOCOUNT ON
USE ' + QUOTENAME (@bd) + N';

        INSERT INTO dbo.SalesOrderHeader
        (
    RevisionNumber, OrderDate, DueDate, ShipDate, Status,
    OnlineOrderFlag, SalesOrderNumber, PurchaseOrderNumber,
```

```
AccountNumber, CustomerID, SalesPersonID, TerritoryID
    )
    VALUES
    (
    0,
CONVERT(datetime, ''' + @OrderDateIso + N''', 126),
DATEADD(DAY, 7, CONVERT(datetime, ''' + @OrderDateIso + N''', 126)),
CONVERT(datetime, ''' + @OrderDateIso + N''', 126),
    5,
    0,
    N''' + @SalesOrderNumberEsc + N''',
NULL,
NULL,
    ' + CAST(@CustomerID AS NVARCHAR(20)) + N',
    ' + CAST(@SalesPersonID AS NVARCHAR(20)) + N',
    ' + CAST(@TerritoryID AS NVARCHAR(20)) + N'
    );

    DECLARE @NewSalesOrderID INT = CAST(SCOPE_IDENTITY() AS INT);

    -- Generar SalesOrderDetailID sin colisiones
    DECLARE @NewDetailID INT;
    SELECT @NewDetailID = ISNULL(MAX(SalesOrderDetailID), 0) + 1
    FROM dbo.SalesOrderDetail WITH (UPDLOCK, HOLDLOCK);

    INSERT INTO dbo.SalesOrderDetail
    (
SalesOrderID, SalesOrderDetailID, CarrierTrackingNumber,
```

```
        OrderQty, ProductID, SpecialOfferID,
        UnitPrice, UnitPriceDiscount,
        LineTotal, rowguid, ModifiedDate
            )
            VALUES
            (
        @NewSalesOrderID,
        @NewDetailID,
        NULL,
        CAST(' + CAST(@OrderQty AS NVARCHAR(20)) + N' AS smallint),
            ' + CAST(@ProductID AS NVARCHAR(20)) + N',
            1,
        CAST(' + CONVERT(VARCHAR(50), @UnitPrice) + N' AS money),
            0,
        CAST(' + CONVERT(VARCHAR(100), @LineTotal) + N' AS numeric(38,6)),
        NEWID(),
        GETDATE()
            );

            SELECT @NewSalesOrderID AS SalesOrderID;
';

            -- Como AT necesita el nombre del linked server “hardcodeado”, armamos 1 nivel de SQL dinámico
            DECLARE @Cmd NVARCHAR(MAX) =
            N'INSERT INTO #NewSO(SalesOrderID)
          EXEC (N''' + REPLACE(@RemoteBatch, N'''', N'''''') + N''') AT ' + QUOTENAME(@servidor) + N';';

            -- DEBUG: descomenta si vuelve a fallar para ver el SQL exacto que se ejecuta
```

```
        -- SELECT @Cmd AS DebugCmd;

        EXEC (@Cmd);

        DECLARE @SalesOrderID INT = (SELECT TOP 1 SalesOrderID FROM #NewSO);

        SELECT
    @SalesOrderID AS SalesOrderID,
    @servidor   AS LinkedServer,
        @bd                 AS BaseDatos;
END;
GO


------------------------------------------------------------
--. SP CONSULTA DISTRIBUIDA DE SALESORDERHEADER
------------------------------------------------------------
IF OBJECT_ID('dbo.usp_ConsultaOrderHeaderDistribuida', 'P') IS NOT NULL
        DROP PROCEDURE dbo.usp_ConsultaOrderHeaderDistribuida;
GO

CREATE PROCEDURE dbo.usp_ConsultaOrderHeaderDistribuida
@consulta NVARCHAR(MAX)
AS
BEGIN
        DECLARE @TerritoryID INT,
        @region NVARCHAR(50),
        @query NVARCHAR(MAX);
```

```
-- Extraer TerritoryID del WHERE
SELECT @TerritoryID = TRY_CAST(
LTRIM(RTRIM(
REPLACE(SUBSTRING(@consulta, CHARINDEX('=', @consulta)+1, 20), ';','')
)) AS INT);

IF @TerritoryID IS NULL
BEGIN
RAISERROR('No se pudo extraer TerritoryID.', 16, 1);
RETURN;
END;

-- Buscar región
SELECT @region =
CASE [Group]
WHEN 'North America' THEN 'AW_NA'
ELSE [Group]
END
FROM AdventureWorks2022.Sales.SalesTerritory
WHERE TerritoryID = @TerritoryID;

IF @region IS NULL
BEGIN
RAISERROR('TerritoryID no existe en ninguna región.', 16, 1);
RETURN;
END;
```

```
        SET @query = 'WHERE TerritoryID = ' + CAST(@TerritoryID AS NVARCHAR);

        EXEC dbo.execquery @region, 'SalesOrderHeader', @query;
END;
GO

/*-----------------------------------------------------------
b) Total de ventas por territorio
-----------------------------------------------------------*/
 -- INCISO B JUAN

 IF OBJECT_ID('dbo.sp_TotalVentasPorTerritorio', 'P') IS NOT NULL
        DROP PROCEDURE dbo.sp_TotalVentasPorTerritorio;
GO
CREATE OR ALTER PROCEDURE dbo.sp_TotalVentasPorTerritorio
    @FechaInicio DATE,
    @FechaFin    DATE,
    @Region      NVARCHAR(50) = NULL
AS
BEGIN
        SET NOCOUNT ON;

        -- Validaciones básicas
        IF @FechaInicio IS NULL OR @FechaFin IS NULL
        THROW 51001, 'Debe proporcionar @FechaInicio y @FechaFin.', 1;

        IF @FechaFin < @FechaInicio
```

```
        THROW 51002, '@FechaFin no puede ser menor que @FechaInicio.', 1;

        DECLARE @sql NVARCHAR(MAX) = N'';

        ;WITH Mapeo AS (
        SELECT DISTINCT servidor, bd, predicado
        FROM dbo.diccionariodist
        WHERE tabla = N'SalesOrderHeader'
        AND (@Region IS NULL OR predicado = @Region)
        )
        SELECT @sql = STRING_AGG(
        N'SELECT N''' + predicado + N''' AS Region,
           st.TerritoryID,
           st.Name AS Territorio,
           COUNT(*) AS NumOrdenes,
           SUM(soh.TotalDue) AS TotalVentas
    FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader AS soh
    JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesTerritory   AS st
      ON soh.TerritoryID = st.TerritoryID
   WHERE soh.OrderDate >= @pInicio
     AND soh.OrderDate  < DATEADD(day, 1, @pFin)
   GROUP BY st.TerritoryID, st.Name',
        N' UNION ALL '
        )
        FROM Mapeo;

        IF @sql IS NULL OR LEN(@sql) = 0
        THROW 51003, 'No hay mapeos en diccionariodist para SalesOrderHeader con la región
especificada.', 1;
```

```
        -- Orden final
        SET @sql = @sql + N' ORDER BY Region, Territorio;';

        -- Ejecutar la consulta distribuida con parámetros
        EXEC sp_executesql
        @sql,
        N'@pInicio DATE, @pFin DATE',
    @pInicio = @FechaInicio,
    @pFin    = @FechaFin;
END;
GO


------------------------------------------------------
IF OBJECT_ID('dbo.usp_TotalVentasPorTerritorio', 'P') IS NOT NULL
        DROP PROCEDURE dbo.usp_TotalVentasPorTerritorio;
GO

CREATE PROCEDURE dbo.usp_TotalVentasPorTerritorio
   @TerritoryID INT,
   @FechaInicio DATE,
   @FechaFin    DATE
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @region   NVARCHAR(50),
```

```
@servidor NVARCHAR(255),
@bd      NVARCHAR(255),
@sql     NVARCHAR(MAX),
@Total   MONEY;

-- Determinar región a partir del TerritoryID usando la BD central
SELECT @region =
CASE [Group]
WHEN 'North America' THEN 'AW_NA'
ELSE [Group]
END
FROM AdventureWorks2022.Sales.SalesTerritory
WHERE TerritoryID = @TerritoryID;

IF @region IS NULL
BEGIN
RAISERROR('TerritoryID no existe.', 16, 1);
RETURN;
END;

-- Ubicar el fragmento correspondiente a esa región
SELECT TOP 1 @servidor = servidor,
   @bd             = bd
FROM diccionariodist
WHERE tabla = 'SalesOrderHeader'
AND predicado = @region;

IF @servidor IS NULL
```

```
        BEGIN
        RAISERROR('No se encontró fragmento para la región %s.', 16, 1, @region);
        RETURN;
        END;

        -- Consulta remota para sumar TotalDue
        SET @sql = N'
    SELECT @TotalOut = SUM(TotalDue)
        FROM ' + QUOTENAME(@servidor) + N'.' + QUOTENAME(@bd) + N'.dbo.SalesOrderHeader
    WHERE TerritoryID = @TerritoryID
     AND OrderDate >= @FechaInicio
     AND OrderDate < DATEADD(DAY, 1, @FechaFin);
        ';

        EXEC sp_executesql
        @sql,
        N'@TerritoryID INT, @FechaInicio DATE, @FechaFin DATE, @TotalOut MONEY OUTPUT',
    @TerritoryID = @TerritoryID,
    @FechaInicio = @FechaInicio,
    @FechaFin           = @FechaFin,
    @TotalOut = @Total OUTPUT;

        SELECT @Total AS TotalVentas;
END;
GO

/*-------------------------------------------------------
c)  Consultar el total de ventas de toda la base de datos.
```

```
            SP TOTAL VENTAS GLOBAL
------------------------------------------------------------*/

-- Para guardar Total Venta Norte America
IF OBJECT_ID('dbo.usp_TotalVentas_NA', 'P') IS NOT NULL
            DROP PROCEDURE dbo.usp_TotalVentas_NA;
GO

CREATE PROCEDURE dbo.usp_TotalVentas_NA
    @FechaInicio DATE,
    @FechaFin    DATE,
    @TotalNA     MONEY OUTPUT
AS
BEGIN
            SET NOCOUNT ON;

            SELECT
        @TotalNA = SUM(TotalDue)
            FROM SRV_NA.AW_NA.dbo.SalesOrderHeader
            WHERE OrderDate >= @FechaInicio
            AND OrderDate <  DATEADD(DAY, 1, @FechaFin);
END;
GO

-- Para guardar TotalVneta de Europa
IF OBJECT_ID('dbo.usp_TotalVentas_EU', 'P') IS NOT NULL
            DROP PROCEDURE dbo.usp_TotalVentas_EU;
GO
```

```
CREATE PROCEDURE dbo.usp_TotalVentas_EU
   @FechaInicio DATE,
   @FechaFin    DATE,
   @TotalEU     MONEY OUTPUT
AS
BEGIN
        SET NOCOUNT ON;

        SELECT
      @TotalEU = SUM(TotalDue)
        FROM SRV_EU.AW_EU.dbo.SalesOrderHeader
        WHERE OrderDate >= @FechaInicio
        AND OrderDate <  DATEADD(DAY, 1, @FechaFin);
END;
GO

--Sumando ambas Ventas
IF OBJECT_ID('dbo.usp_TotalVentas_Global', 'P') IS NOT NULL
        DROP PROCEDURE dbo.usp_TotalVentas_Global;
GO

CREATE PROCEDURE dbo.usp_TotalVentas_Global
   @FechaInicio DATE,
   @FechaFin    DATE
AS
BEGIN
        SET NOCOUNT ON;
```

```
        DECLARE
    @TotalNA MONEY,
    @TotalEU MONEY,
        @TotalGlobal MONEY;

--Para norte america
        EXEC dbo.usp_TotalVentas_NA
        @FechaInicio = @FechaInicio,
    @FechaFin = @FechaFin,
    @TotalNA   = @TotalNA OUTPUT;

--Para europa
        EXEC dbo.usp_TotalVentas_EU
        @FechaInicio = @FechaInicio,
    @FechaFin = @FechaFin,
    @TotalEU   = @TotalEU OUTPUT;



        SET @TotalGlobal = ISNULL(@TotalNA, 0) + ISNULL(@TotalEU, 0);


        SELECT
    @TotalNA   AS TotalNorthAmerica,
    @TotalEU   AS TotalEurope,
    @TotalGlobal  AS TotalGlobal;
END;
GO
```

```
/*-------------------------------------------------------------------*/



/*-----------------------------------------------------
d)  Consulte el producto más vendido y menos vendido en un
        periodo especificado dada una categoría en un área
        especificada
        SP PRODUCTO MÁS Y MENOS VENDIDO POR REGIÓN
-----------------------------------------------------------*/


-- INCISO D JUAN
 IF OBJECT_ID('dbo.sp_ProductoMasYMenosVendidoPorCategoria', 'P') IS NOT NULL
        DROP PROCEDURE dbo.sp_ProductoMasYMenosVendidoPorCategoria;


GO
CREATE OR ALTER PROCEDURE dbo.sp_ProductoMasYMenosVendidoPorCategoria
    @FechaInicio DATE,
    @FechaFin    DATE,
    @Categoria   NVARCHAR(100),
    @Region      NVARCHAR(50) = NULL,          -- 'Europe' | 'North America' | NULL (ambas)
        @Metrica         NVARCHAR(20)  = N'Cantidad'   -- 'Cantidad' | 'Importe'
AS
BEGIN
        SET NOCOUNT ON;


        -- Validaciones básicas
        IF @FechaInicio IS NULL OR @FechaFin IS NULL
```

```
THROW 51001, 'Debe proporcionar @FechaInicio y @FechaFin.', 1;

IF @FechaFin < @FechaInicio
THROW 51002, '@FechaFin no puede ser menor que @FechaInicio.', 1;

IF @Metrica NOT IN (N'Cantidad', N'Importe')
THROW 51004, 'Parametro @Metrica inválido. Use "Cantidad" o "Importe".', 1;

/*
Construimos una consulta federada que agrega por producto en cada nodo,
filtrando por periodo y categoría; luego calculamos el máximo y mínimo
en CTEs separadas para evitar ORDER BY dentro del UNION ALL.
*/
DECLARE @sql NVARCHAR(MAX) = N'';

;WITH Mapeo AS (
SELECT DISTINCT servidor, bd, predicado AS Region
FROM dbo.diccionariodist
WHERE tabla = N'SalesOrderDetail'
AND (@Region IS NULL OR predicado = @Region)
)
SELECT @sql =
STRING_AGG(
N'SELECT N''' + Region + N''' AS Region,
            p.ProductID,
            p.Name AS Producto,
            ps.Name AS Subcategoria,
            pc.Name AS Categoria,
```

```
            SUM(sod.OrderQty)  AS Cantidad,
            SUM(sod.LineTotal) AS Importe
        FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderDetail AS sod
        JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader AS soh
          ON sod.SalesOrderID = soh.SalesOrderID
        JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.Product AS p
          ON sod.ProductID = p.ProductID
        JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.ProductSubcategory AS ps
          ON p.ProductSubcategoryID = ps.ProductSubcategoryID
        JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.ProductCategory AS pc
          ON ps.ProductCategoryID = pc.ProductCategoryID
       WHERE soh.OrderDate >= @pInicio
         AND soh.OrderDate  < DATEADD(day, 1, @pFin)
         AND pc.Name = @pCategoria
       GROUP BY p.ProductID, p.Name, ps.Name, pc.Name',
       N' UNION ALL '
       )
       FROM Mapeo;

       IF @sql IS NULL OR LEN(@sql) = 0
       THROW 51003, 'No hay mapeos en diccionariodist para SalesOrderDetail con la región
especificada.', 1;

       -- Ordenamientos según la métrica (se usarán dentro de CTEs separadas)
       DECLARE @OrdenMax NVARCHAR(200) =
       CASE WHEN @Metrica = N'Cantidad'
       THEN N'Cantidad DESC, Importe DESC, ProductID'
       ELSE N'Importe  DESC, Cantidad DESC, ProductID'
       END;
```

```
DECLARE @OrdenMin NVARCHAR(200) =
CASE WHEN @Metrica = N'Cantidad'
THEN N'Cantidad ASC, Importe ASC, ProductID'
ELSE N'Importe  ASC, Cantidad ASC, ProductID'
END;

/*
Armado final:
- Accedemos a los nodos donde estan los datos.
- cada uno hace TOP 1 con su ORDER BY, como CTEs separadas.
- Resultado: union de esas dos filas.
*/
SET @sql = N'WITH Datos AS (' + @sql + N'),
    Maximo AS (
        SELECT TOP 1 Region, Producto, ProductID, Subcategoria, Categoria, Cantidad, Importe
        FROM Datos
        ORDER BY ' + @OrdenMax + N'
    ),
    Minimo AS (
        SELECT TOP 1 Region, Producto, ProductID, Subcategoria, Categoria, Cantidad, Importe
        FROM Datos
        ORDER BY ' + @OrdenMin + N'
    )
  SELECT N''MasVendido'' AS Tipo, Region, Producto, ProductID, Subcategoria, Categoria,
Cantidad, Importe
  FROM Maximo
UNION ALL
```

```
        SELECT N"MenosVendido" AS Tipo, Region, Producto, ProductID, Subcategoria, Categoria,
Cantidad, Importe

        FROM Minimo;';


        -- Ejecutar con parámetros seguros

        EXEC sp_executesql

        @sql,

        N'@pInicio DATE, @pFin DATE, @pCategoria NVARCHAR(100)',

    @pInicio    = @FechaInicio,

    @pFin       = @FechaFin,

    @pCategoria = @Categoria;

END;

GO


/*------------------------------------------------------------

e) El empleado con más ventas regitradas por áreas

        SP EMPLEADO TOP VENTAS POR REGIÓN

------------------------------------------------------------*/

/*

-- Qué ventaja tiene el uso de una tabla en la consulta

        1. Acceso a datos distribuidos: Permite consultar la información que está en diferentes servidores

        o nodos como si fuera una sola fuente

        2. Escalabilidad :Pueden aprovechar la capacidad de multiples maquinas, lo que mejora

        el rendimiento en grandes columenes de datos

        3. Flexibilidad: Se pueden combinar datos de distintas bases sin necesidad de moverlos

        fisicamente, reduciendo costos

        4. Optimiza el rendimiento: es decir que optimiza la ejecución dividiendo

        la carga entre nodos, lo que puede acelerar la respuesta
```

-- Qué desventaja tiene el uso de una tabla en la consulta

1. Complejidad en la configuración: se requiere definir muy bien los enlaces, permisos entre los servidores, lo que lo hace tedioso

2.Mayor latencia: Al depender de redes y muchos nodos, las consultas se pueden hacer lentas en la base lcoal, en especial si hay grandes volumenes de datos

3. Problemas de consistencia: Si los datos cmabian en tiempo real, puede haber incosistencias entre nodos durante la ejecucion de la consulta

4. Dependencia de la red: Si falla la conexión entre servidores, la consulta puede fallar o tardar demasiado

5. Costos de mantenimiento: Monitorea y asegura la integridad de entornos distribuidos

-- Por que no utilizar una variable de tabla a nivel memoria, ¿se puede?

No se puede

1. Ámbito y visibilidad: Porque tienen un alcance estrictamente local, no tienen esquema ni un catálogo que pueda referenciar a otro servidor. En consultas distribuidas el otro servidor no puede ver ni acceder a tu variable de tabla local.

2. Serialización/Red: Para compartir algo con otro nodo, hay que serializarlo y enviarlo por la red. Las variables de tabla no estan diseñadas para eso. Lo que sí se puede enviar es el resultado de un SELECT no la reestructura interna de una varibale de tabla

3.Optimización y estadísticas: Estas suelen tener estadísticas limitadas, lo que afecta la calidad del plan de ejecucion. En un entrono distribuido, esto empeora el coste estimado y puede derivar en planes

4.Transaccionalidad y consistencia: Al estar encapsuladas en una sola instancia y sesion, no se puede garantizar aislamiento/consistencia dsitribuida ni coordinación de transacciones entre nodos

```
-- Reescibir la consulta con una consulta distribuida para evitar la tabla temporal
*/
IF OBJECT_ID('dbo.usp_EmpleadoTopVentasPorArea', 'P') IS NOT NULL
	DROP PROCEDURE dbo.usp_EmpleadoTopVentasPorArea;
GO
CREATE PROCEDURE dbo.usp_EmpleadoTopVentasPorArea
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @union NVARCHAR(MAX) = N'';
	DECLARE @sql   NVARCHAR(MAX) = N'';
	;WITH Dist AS  --El ; asegura que el WITH no choque con una instruccion previa
	(
	SELECT DISTINCT servidor, bd, predicado
	FROM dbo.diccionariodist
	WHERE tabla = 'SalesOrderHeader'
	)
	--Voy a concatenar Strings
	SELECT
	@union = @union  --va acumulando
	+ CASE WHEN @union = N'' THEN N'' ELSE N'
UNION ALL
' END
	 + N'SELECT '
	 + QUOTENAME(predicado,'') + N' AS Region,
      SalesPersonID,
```

```
        SUM(TotalDue) AS TotalVentas
 FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader
 WHERE SalesPersonID IS NOT NULL
 GROUP BY SalesPersonID'
            FROM Dist --Repite por cada ruta del diccionario
            ORDER BY predicado;

            IF @union = N''
            THROW 50001, 'No hay filas en diccionariodist para tabla = SalesOrderHeader.', 1;

            SET @sql = N';
WITH VentasUnion AS
(
' + @union + N'
),
VentasPorRegion AS
(
            -- Por si una región tuviera más de un fragmento
            SELECT Region, SalesPersonID, SUM(TotalVentas) AS TotalVentas
            FROM VentasUnion
            GROUP BY Region, SalesPersonID
),
Ranked AS --Asigna un rango por region
(
            SELECT Region,
          SalesPersonID,
          TotalVentas,
          RANK() OVER (PARTITION BY Region ORDER BY TotalVentas DESC) AS rn
```

```
        FROM VentasPorRegion
)
SELECT Region, SalesPersonID, TotalVentas
FROM Ranked
WHERE rn = 1
ORDER BY Region;
';

        EXEC sys.sp_executesql @sql;
END;
GO



/*-------------------------------------------------------
f)  El cliente ocom más órdenes solicitadas en casa área
        CLIENTE TOP ORDENES POR REGIÓN
-------------------------------------------------------*/
IF OBJECT_ID('dbo.usp_ClienteTopOrdenesPorArea', 'P') IS NOT NULL
        DROP PROCEDURE dbo.usp_ClienteTopOrdenesPorArea;
GO

CREATE PROCEDURE dbo.usp_ClienteTopOrdenesPorArea
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @union NVARCHAR(MAX) = N'';
```

```
        DECLARE @sql   NVARCHAR(MAX) = N'';

        ;WITH Dist AS
        (
        SELECT DISTINCT servidor, bd, predicado
        FROM dbo.diccionariodist
        WHERE tabla = 'SalesOrderHeader'
        )
        SELECT
      @union = @union
         + CASE WHEN @union = N'' THEN N'' ELSE N'
UNION ALL
' END
         + N'SELECT '
         + QUOTENAME(predicado,'''') + N' AS Region,
       CustomerID,
       COUNT(*) AS NumeroOrdenes
 FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader
 GROUP BY CustomerID'
        FROM Dist
        ORDER BY predicado;

        IF @union = N''
        THROW 50002, 'No hay filas en diccionariodist para tabla = SalesOrderHeader.', 1;

        SET @sql = N';
WITH OrdenesUnion AS
(
```

```
' + @union + N'
),
OrdenesPorRegion AS
(
        -- Por si una región tuviera más de un fragmento, consolida
    SELECT Region, CustomerID, SUM(NumeroOrdenes) AS NumeroOrdenes
        FROM OrdenesUnion
        GROUP BY Region, CustomerID
),
Ranked AS
(
        SELECT Region,
      CustomerID,
      NumeroOrdenes,
      RANK() OVER (PARTITION BY Region ORDER BY NumeroOrdenes DESC) AS rn
        FROM OrdenesPorRegion
)
SELECT Region, CustomerID, NumeroOrdenes
FROM Ranked
WHERE rn = 1
ORDER BY Region;
';

        EXEC sys.sp_executesql @sql;
END;
GO
```

EXEC de los SP

```
---- CONSULTAS REMOTAS--------

/*------------------------------

        Prueba conexion

--------------------------------*/

EXEC dbo.usp_ConsultarTablaLinkedServer

        @ServidorVinculado = 'SRV_NA',

   @BaseDatos = 'AW_NA',

        @Tabla = 'SalesTerritory';


        EXEC dbo.usp_ConsultarTablaLinkedServer

        @ServidorVinculado = 'SRV_EU',

   @BaseDatos = 'AW_EU',

        @Tabla = 'SalesTerritory';


/*------------------------------

a) Registrar venta por area

--------------------------------*/


EXEC dbo.usp_RegistrarVentaRegion

        @Region          = 'AW_NA',

   @CustomerID = 11000,

   @SalesPersonID = 280,

   @TerritoryID   = 1,

   @ProductID    = 776,

   @OrderQty     = 5,

   @UnitPrice     = 100.00,

   @FechaOrden          = '2013-01-15';
```

```
        EXEC dbo.usp_RegistrarVentaRegion
        @Region          = 'AW_EU',
  @CustomerID = 11000,
  @SalesPersonID = 280,
  @TerritoryID   = 4,
  @ProductID    = 776,
  @OrderQty     = 5,
        @UnitPrice       = 100.00,
        @FechaOrden    = '2013-01-15';

 --Prueba final final

 DECLARE @F DATETIME = GETDATE();
        EXEC dbo.usp_RegistrarVentaRegion
        @Region = N'AW_NA',
  @CustomerID = 11000,
  @SalesPersonID = 280,
  @TerritoryID = 1,
  @ProductID = 776,
  @OrderQty = 5,
  @UnitPrice = 100.00,
  @FechaOrden = @F;


/*----------------------------
b) Registrar venta por area
 Total Ventas por territorio por fechas
```

```
-------------------------------*/

select * from diccionariodist


EXEC dbo.sp_TotalVentasPorTerritorio

        @TerritoryID = 1,          --Europa = 4

        @FechaInicio = '2013-01-01',

        @FechaFin    = '2013-12-31';




/*-----------------------------

c) Total de Ventas global

-------------------------------*/


--Sumando ambas consultas

        EXEC dbo.usp_TotalVentas_Global

        @FechaInicio = '2011-01-01',

        @FechaFin       = '2011-07-31';



select * from SRV_NA.AW_NA.dbo.SalesOrderheader

select * from SRV_NA.AW_NA.dbo.SalesOrderDetail

select * from SRV_NA.AW_NA.dbo.product

select * from SRV_NA.AW_NA.dbo.productCategory

select * from SRV_NA.AW_NA.dbo.productsubCategory



/*-----------------------------
```

```
d) Producto más menos vendido por categoría y área

--------------------------------*/

EXEC dbo.sp_ProductoMasYMenosVendidoPorCategoria

        @FechaInicio = '2013-01-01',

        @FechaFin       = '2013-12-31',

   @Categoria   = N'Bikes',

        @Region = N'AW_NA',

        @Metrica     = N'Importe'; -- ¿Por qué se utiliza?¿Qué representa?¿Es necesario?


        EXEC dbo.sp_ProductoMasYMenosVendidoPorCategoria

        @FechaInicio = '2013-01-01',

        @FechaFin       = '2013-12-31',

   @Categoria   = N'Bikes',

        @Metrica          = N'Cantidad';


/*------------------------------

e) Empelado con más ventas por área

--------------------------------*/


EXEC dbo.usp_EmpleadoTopVentasPorArea;




/*------------------------------

f) Cliente con más órdenes solicitadas por área más ventas por área

--------------------------------*/


EXEC dbo.usp_ClienteTopOrdenesPorArea;
```

```sql
select salespersonid, sum(totaldue) from SRV_NA.AW_NA.dbo.SalesOrderheader

group by salespersonid

order by salespersonid
```

API(Luis):

```python
from flask import Flask, request, jsonify

import pyodbc

import os

from datetime import datetime

import traceback



app = Flask(__name__)



# ----------------------------------------

# 1) CONEXIÓN AL DICCIONARIO (LOCALHOST)

# ----------------------------------------

CONN_STR = os.getenv(
```

```
    "CONN_STR",
    "DRIVER={ODBC Driver 17 for SQL Server};"
    "SERVER=localhost;"
    "DATABASE=AW_CLIENTE_1;"
    "UID=Alumno;"
    "PWD=Estudiantes;"
)


def get_conn():
    return pyodbc.connect(CONN_STR)




# Modificación de la función para validar múltiples formatos de fecha
def validar_fecha(valor):
    formatos = ["%Y-%m-%d", "%Y/%m/%d"]  # Se añaden los dos formatos
    for formato in formatos:
        try:
            datetime.strptime(valor, formato)
            return True
        except Exception:
            continue
```

```
    print(f"ERROR FECHA: {valor} no es un formato válido")
    return False


@app.route("/", methods=["GET"])
def root():
    return jsonify({
        "status": "ok",
        "message": "API funcionando",
    })
# ============================================================
#  A) POST /registrar_venta  --> usa usp_RegistrarVentaRegion
# ============================================================
@app.route("/registrar_venta", methods=["POST"])
def registrar_venta():
    data = request.get_json()

    required = [
        "Region","CustomerID","SalesPersonID",
        "TerritoryID","ProductID","OrderQty",
        "UnitPrice","FechaOrden"
    ]
    if not all(k in data for k in required):
```

```
        return jsonify({"error": "Faltan parámetros"}), 400


    try:
        conn = get_conn()
        cursor = conn.cursor()
        cursor.execute(
            """
            EXEC dbo.usp_RegistrarVentaRegion
                @Region=?, @CustomerID=?, @SalesPersonID=?,
@TerritoryID=?,
                @ProductID=?, @OrderQty=?, @UnitPrice=?,
@FechaOrden=?
            """,
            data["Region"],
            data["CustomerID"],
            data["SalesPersonID"],
            data["TerritoryID"],
            data["ProductID"],
            data["OrderQty"],
            data["UnitPrice"],
            data["FechaOrden"]
        )
        row = cursor.fetchone()
```

```
        conn.commit()
        return jsonify({
            "SalesOrderID": row[0],
            "LinkedServer": row[1],
            "BaseDatos": row[2]
        })
    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500



# ============================================================
#  B) GET /ventas_territorio
#     -> llama sp_TotalVentasPorTerritorio (Juan)
# ============================================================
@app.route("/ventas_territorio", methods=["GET"])
def ventas_territorio():
    region = request.args.get("region")  # opcional
    inicio = request.args.get("inicio")
    fin = request.args.get("fin")

    if not inicio or not fin:
```

```
        return jsonify({"error": "inicio y fin requeridos"}), 400


    try:
        conn = get_conn()
        cursor = conn.cursor()


        cursor.execute(
            """
            EXEC dbo.sp_TotalVentasPorTerritorio
                @FechaInicio=?, @FechaFin=?, @Region=?
            """,
            inicio, fin, region
        )


        cols = [c[0] for c in cursor.description]
        rows = [dict(zip(cols, r)) for r in cursor.fetchall()]
        return jsonify(rows)


    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500
```

```python
# ============================================================
#  C) GET /ventas_global
#     -> llama usp_TotalVentas_Global
# ============================================================
@app.route("/ventas_global", methods=["GET"])
def ventas_global():
    inicio = request.args.get("inicio")
    fin = request.args.get("fin")

    if not inicio or not fin:
        return jsonify({"error": "inicio y fin requeridos"}), 400

    try:
        conn = get_conn()
        cursor = conn.cursor()
        cursor.execute(
            """
            EXEC dbo.usp_TotalVentas_Global
                @FechaInicio=?, @FechaFin=?
            """,
            inicio, fin
```

```
        )
        row = cursor.fetchone()
        return jsonify({
            "TotalNorthAmerica": row[0],
            "TotalEurope": row[1],
            "TotalGlobal": row[2]
        })

    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500



#=========================================================
#  D) GET /producto_mas_menos
#     Params:
#       categoria (nombre de la categoría, p.ej. 'Bikes')
#       area (opcional) -> 'NA'|'EU' o nombres 'North
America'|'Europe'
#       inicio  YYYY-MM-DD
#       fin     YYYY-MM-DD
#       metrica (opcional) -> 'Cantidad' (default) o 'Importe'
#     Llama: dbo.sp_ProductoMasYMenosVendidoPorCategoria
```

```python
# ============================================================




# Ruta con validación de fecha
@app.route("/producto_mas_menos", methods=["GET"])
def producto_mas_menos():
    categoria = request.args.get("categoria")
    area = request.args.get("area")  # opcional
    inicio = request.args.get("inicio")
    fin = request.args.get("fin")
    metrica = request.args.get("metrica", "Cantidad")  # default

    # Logs de depuración
    print(">>> categoria recibida:", categoria)
    print(">>> area recibida:", area)
    print(">>> inicio recibido:", repr(inicio))
    print(">>> fin recibido:", repr(fin))
    print(">>> metrica recibida:", metrica)

    # Validación de requeridos
    if not categoria or not inicio or not fin:
```

```
        return jsonify({"error": "Parametros requeridos:
categoria, inicio, fin"}), 400


    # Limpieza de espacios invisibles
    inicio = inicio.replace("\r", "").replace("\n", "").strip()
    fin = fin.replace("\r", "").replace("\n", "").strip()


    # Map short area codes a los nombres del diccionario
    if area:
        area_map = {
    "NA": "AW_NA",
    "NORTH": "AW_NA",
    "AW_NA": "AW_NA",
    "EU": "AW_EU",
    "EURO": "AW_EU",
    "AW_EU": "AW_EU"
}


        area = area_map.get(area.upper(), area)


    # Validar metrica
    if metrica not in ("Cantidad", "Importe"):
```

```
        return jsonify({"error": "Parametro 'metrica' inválido.
Use 'Cantidad' o 'Importe'."}), 400


    # Validación de fechas con la nueva función
    if not validar_fecha(inicio) or not validar_fecha(fin):
        return jsonify({"error": "Formato de fecha inválido. Use
YYYY-MM-DD o YYYY/MM/DD"}), 400


    try:
        conn = get_conn()
        cursor = conn.cursor()


        cursor.execute(
            """
            EXEC dbo.sp_ProductoMasYMenosVendidoPorCategoria
                @FechaInicio=?, @FechaFin=?, @Categoria=?,
@Region=?, @Metrica=?
            """,
            inicio, fin, categoria, area, metrica
        )


        cols = [c[0] for c in cursor.description] if
cursor.description else []
        rows = [dict(zip(cols, r)) for r in cursor.fetchall()]
```

```
        if not rows:
            return jsonify({"message": "Sin resultados"}), 200

        return jsonify(rows), 200

    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500




# ============================================================
#  E) GET /empleado_top
#     Llama: dbo.usp_EmpleadoTopVentasPorArea
# ============================================================
@app.route("/empleado_top", methods=["GET"])
def empleado_top():
    try:
        conn = get_conn()
        cursor = conn.cursor()
```

```
        cursor.execute("EXEC dbo.usp_EmpleadoTopVentasPorArea;")
        cols = [c[0] for c in cursor.description] if
cursor.description else []
        rows = [dict(zip(cols, r)) for r in cursor.fetchall()]
        if not rows:
            return jsonify({"message": "Sin resultados"}), 200
        return jsonify(rows), 200

    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500
# ============================================================
#  F) GET /cliente_top
#     Llama: dbo.usp_ClienteTopOrdenesPorArea
# ============================================================
@app.route("/cliente_top", methods=["GET"])
def cliente_top():
    try:
        conn = get_conn()
        cursor = conn.cursor()
        cursor.execute("EXEC dbo.usp_ClienteTopOrdenesPorArea;")
        cols = [c[0] for c in cursor.description] if
cursor.description else []
```

```
        rows = [dict(zip(cols, r)) for r in cursor.fetchall()]
        if not rows:
            return jsonify({"message": "Sin resultados"}), 200
        return jsonify(rows), 200

    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500

# ============================================================
# MAIN
# ============================================================
if __name__ == "__main__":
    print("API corriendo en http://127.0.0.1:5000")
    app.run(host="0.0.0.0", port=5000)
```

API AWS:

```
# app.py
from flask import Flask, request, jsonify
import pyodbc
import os
import traceback
from datetime import datetime
```

```
app = Flask(__name__)

# ---------------------
# CONFIG
# ---------------------
CONN_STR_NA = os.getenv(
    "CONN_STR_NA",
    "DRIVER={ODBC Driver 17 for SQL Server};"

"SERVER=database-1.c8z0w24sg3wo.us-east-1.rds.amazonaws.com,1433;"
    "DATABASE=AW_NA;"
    "UID=admin;"
    "PWD=basededatos;"
    "Encrypt=yes;"
    "TrustServerCertificate=yes"
)

CONN_STR_EU = os.getenv(
    "CONN_STR_EU",
    "DRIVER={ODBC Driver 17 for SQL Server};"

"SERVER=database-2.c8z0w24sg3wo.us-east-1.rds.amazonaws.com,1433;"
    "DATABASE=AW_EU;"
    "UID=admin;"
    "PWD=basededatos;"
    "Encrypt=yes;"
    "TrustServerCertificate=yes"
)

CONNS = {
    "NA": CONN_STR_NA,
    "EU": CONN_STR_EU
}

# ---------------------
```

```python
# HELPERS
# ---------------------
def get_conn(region):
    """Devuelve una conexión pyodbc abierta para una region 'NA' o
'EU'"""
    conn_str = CONNS.get(region)
    if not conn_str:
        raise ValueError("Región desconocida: " + str(region))
    # autocommit True para que SPs con TRAN funcionen sin bloquear
    return pyodbc.connect(conn_str, autocommit=True)

def sp_fetchall(conn, sp_name, params=()):
    """
    Ejecuta un stored procedure con parámetros posicionales y
devuelve lista de dicts.
    Construye EXEC sólo con placeholders cuando hay params.
    """
    cursor = conn.cursor()
    try:
        if params and len(params) > 0:
            placeholders = ", ".join("?" for _ in params)
            sql = f"EXEC {sp_name} {placeholders}"
            cursor.execute(sql, params)
        else:
            sql = f"EXEC {sp_name}"
            cursor.execute(sql)
        cols = [c[0] for c in cursor.description] if
cursor.description else []
        rows = [dict(zip(cols, row)) for row in cursor.fetchall()]
if cols else []
        return rows
    finally:
        try:
            cursor.close()
        except:
            pass
```

```python
def sp_fetchone_scalar(conn, sp_name, params=()):
    rows = sp_fetchall(conn, sp_name, params)
    if not rows:
        return None
    first_row = rows[0]
    if isinstance(first_row, dict):
        vals = list(first_row.values())
        return vals[0] if vals else None
    return first_row

# ---------------------
# Root / health
# ---------------------
@app.route("/", methods=["GET"])
def root():
    return jsonify({
        "status": "ok",
        "message": "API funcionando",
        "endpoints": [
            "/registrar_venta (POST)",
            "/consultaD (GET)",
            "/empleado_mas (GET)",
            "/cliente_mas (GET)",
            "/total_global (GET)",
            "/ventas_territorio (GET)"
        ]
    })

# ---------------------
# ENDPOINTS: Consultas (D, E, F) y consolidación
# ---------------------

@app.route("/consultaD", methods=["GET"])
def consultaD():
    region = (request.args.get("region") or "ALL").upper()
```

```
    try:
        categoria = int(request.args["categoria"])
    except Exception:
        return jsonify({"error":"categoria (int) es obligatorio"}),
400
    inicio = request.args.get("inicio")
    fin = request.args.get("fin")
    if not inicio or not fin:
        return jsonify({"error":"inicio y fin son obligatorios
(YYYY-MM-DD)"}), 400

    targets = ["NA","EU"] if region in ("ALL","*") else [region]
    result = {}
    for r in targets:
        conn = None
        try:
            conn = get_conn(r)
            rows = sp_fetchall(conn,
"dbo.usp_ConsultaProductoMasYMenosVendido", (inicio, fin,
categoria))
            result[r] = rows[0] if rows else {
                "ProductoMasVendido": None,
                "CantidadMasVendida": None,
                "ProductoMenosVendido": None,
                "CantidadMenosVendida": None
            }
        except Exception as e:
            result[r] = {"error": str(e), "trace":
traceback.format_exc()}
        finally:
            if conn:
                try: conn.close()
                except: pass
    return jsonify(result)

@app.route("/empleado_mas", methods=["GET"])
```

```
def empleado_mas():
    region = (request.args.get("region") or "ALL").upper()
    targets = ["NA","EU"] if region in ("ALL","*") else [region]
    result = {}
    for r in targets:
        conn = None
        try:
            conn = get_conn(r)
            rows = sp_fetchall(conn, "dbo.usp_EmpleadoMasVentas",
())
            result[r] = rows[0] if rows else None
        except Exception as e:
            result[r] = {"error": str(e), "trace":
traceback.format_exc()}
        finally:
            if conn:
                try: conn.close()
                except: pass
    return jsonify(result)

@app.route("/cliente_mas", methods=["GET"])
def cliente_mas():
    region = (request.args.get("region") or "ALL").upper()
    targets = ["NA","EU"] if region in ("ALL","*") else [region]
    result = {}
    for r in targets:
        conn = None
        try:
            conn = get_conn(r)
            rows = sp_fetchall(conn, "dbo.usp_ClienteMasOrdenes",
())
            result[r] = rows[0] if rows else None
        except Exception as e:
            result[r] = {"error": str(e), "trace":
traceback.format_exc()}
        finally:
```

```
            if conn:
                try: conn.close()
                except: pass
    return jsonify(result)

@app.route("/total_global", methods=["GET"])
def total_global():
    inicio = request.args.get("inicio")
    fin = request.args.get("fin")
    if not inicio or not fin:
        return jsonify({"error":"inicio y fin son obligatorios"}),
400

    totals = {}
    grand = 0.0
    for r in ("NA","EU"):
        conn = None
        try:
            conn = get_conn(r)
            rows = sp_fetchall(conn, "dbo.usp_TotalVentasLocal",
(None, inicio, fin))
            val = 0.0
            if rows and isinstance(rows[0], dict):
                first = rows[0]
                for v in first.values():
                    if v is not None:
                        try:
                            val = float(v)
                        except:
                            val = 0.0
                        break
            totals[r] = val
            grand += val
        except Exception as e:
            totals[r] = {"error": str(e), "trace":
traceback.format_exc()}
```

```
        finally:
            if conn:
                try: conn.close()
                except: pass
    return jsonify({"totales_por_region": totals, "total_global":
grand})

@app.route("/ventas_territorio", methods=["GET"])
def ventas_territorio():
    """
    Params:
      region (optional): 'NA', 'EU', or 'ALL'
      inicio (required) YYYY-MM-DD
      fin (required) YYYY-MM-DD
    """
    region = (request.args.get("region") or "ALL").upper()
    inicio = request.args.get("inicio")
    fin = request.args.get("fin")
    if not inicio or not fin:
        return jsonify({"error":"inicio y fin obligatorios
(YYYY-MM-DD)"}), 400

    targets = ["NA","EU"] if region in ("ALL","*") else [region]
    result = {}
    for r in targets:
        conn = None
        try:
            conn = get_conn(r)
            # usp_TotalVentasPorTerritorioLocal expects
@FechaInicio, @FechaFin
            rows = sp_fetchall(conn,
"dbo.usp_TotalVentasPorTerritorioLocal", (inicio, fin))
            result[r] = rows if rows else []
        except Exception as e:
            result[r] = {"error": str(e), "trace":
traceback.format_exc()}
```

```
        finally:
            if conn:
                try: conn.close()
                except: pass
    return jsonify(result)

# ---------------------
# ENDPOINT: Registrar venta (header + detalles) en una region
# ---------------------
@app.route("/registrar_venta", methods=["POST"])
def registrar_venta():
    body = request.get_json(force=True)
    region = body.get("region")
    if region is None or region.upper() not in ("NA","EU"):
        return jsonify({"error":"region debe ser 'NA' o 'EU'"}),
400
    region = region.upper()

    header = body.get("header") or {}
    required = ["CustomerID","OrderDate"]
    for f in required:
        if header.get(f) is None:
            return jsonify({"error": f"{f} es obligatorio en
header"}), 400

    details = body.get("details") or []
    if not isinstance(details, list) or len(details) == 0:
        return jsonify({"error":"details debe ser una lista con al
menos 1 item"}), 400

    conn = None
    cursor = None
    try:
        conn = get_conn(region)
        cursor = conn.cursor()
```

```
        sql_header = """
        DECLARE @NewID INT;
        EXEC dbo.usp_RegistrarVentaHeaderLocal
            @CustomerID = ?,
            @SalesPersonID = ?,
            @TerritoryID = ?,
            @OrderDate = ?,
            @ShipMethodID = ?,
            @ShipToAddressID = ?,
            @BillToAddressID = ?,
            @AccountNumber = ?,
            @Comment = ?,
            @NewSalesOrderID = @NewID OUTPUT;
        SELECT @NewID AS NewSalesOrderID;
        """
        header_params = (
            header.get("CustomerID"),
            header.get("SalesPersonID"),
            header.get("TerritoryID"),
            header.get("OrderDate"),
            header.get("ShipMethodID"),
            header.get("ShipToAddressID"),
            header.get("BillToAddressID"),
            header.get("AccountNumber"),
            header.get("Comment"),
        )
        cursor.execute(sql_header, header_params)
        new_row = cursor.fetchone()
        if not new_row:
            return jsonify({"error":"No se obtuvo
NewSalesOrderID"}), 500
        sales_order_id = new_row.NewSalesOrderID

        # detalles
        for d in details:
            params = (
```

```
                sales_order_id,
                d.get("ProductID"),
                d.get("OrderQty"),
                d.get("UnitPrice"),
                d.get("UnitPriceDiscount", 0.0),
                d.get("SpecialOfferID", 1)
            )
            cursor.execute(
                "EXEC dbo.usp_RegistrarVentaDetalleLocal ?, ?, ?,
?, ?, ?",
                params
            )

        return jsonify({"SalesOrderID": sales_order_id}), 201

    except Exception as e:
        return jsonify({"error": str(e), "trace":
traceback.format_exc()}), 500

    finally:
        try:
            if cursor: cursor.close()
        except: pass
        try:
            if conn: conn.close()
        except: pass

# ---------------------
# RUN
# ---------------------
if __name__ == "__main__":
    # config basic para debugging
    print("CONN_STR_NA startswith DRIVER:", CONN_STR_NA[:40])
    print("CONN_STR_EU startswith DRIVER:", CONN_STR_EU[:40])
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Preguntas realizadas por el profesor

¿Qué ventaja tiene el uso de una tabla en la consulta?

1. Acceso a datos distribuidos: Permite consultar la información que está en diferentes servidores o nodos como si fuera una sola fuente
2. Escalabilidad :Pueden aprovechar la capacidad de múltiples maquinas, lo que mejora el rendimiento en grandes volúmenes de datos
3. Flexibilidad: Se pueden combinar datos de distintas bases sin necesidad de moverlos físicamente, reduciendo costos
4. Optimiza el rendimiento: es decir que optimiza la ejecución dividiendo la carga entre nodos, lo que puede acelerar la respuesta

¿Qué desventaja tiene el uso de una tabla en la consulta?

1. Complejidad en la configuración: se requiere definir muy bien los enlaces, permisos entre los servidores, lo que lo hace tedioso

2. Mayor latencia: Al depender de redes y muchos nodos, las consultas se pueden hacer lentas en la base local, en especial si hay grandes volúmenes de datos

3. Problemas de consistencia: Si los datos cambian en tiempo real, puede haber inconsistencias entre nodos durante la ejecución de la consulta

4. Dependencia de la red: Si falla la conexión entre servidores, la consulta puede fallar o tardar demasiado

5. Costos de mantenimiento: Monitorea y asegura la integridad de entornos distribuidos

¿Por qué no utilizar una variable de tabla a nivel memoria? ¿se puede?

No se puede

1. Ámbito y visibilidad: Porque tienen un alcance estrictamente local, no tienen esquema ni un catálogo que pueda referenciar a otro servidor. En consultas distribuidas el otro servidor no puede ver ni acceder a tu variable de tabla local.

2. Serialización/Red: Para compartir algo con otro nodo, hay que serializarlo y enviarlo por la red. Las variables de tabla no están diseñadas para eso. Lo que sí se puede enviar es el resultado de un SELECT no la reestructura interna de una variable de tabla

3. Optimización y estadísticas: Estas suelen tener estadísticas limitadas, lo que afecta la calidad del plan de ejecución. En un entorno distribuido, esto empeora el coste estimado y puede derivar en planes

4. Transaccionalidad y consistencia: Al estar encapsuladas en una sola instancia y sesión, no se puede garantizar aislamiento/consistencia distribuida ni coordinación de transacciones entre nodos

Ejercicios resueltos por cada integrante y Conclusiones individuales:

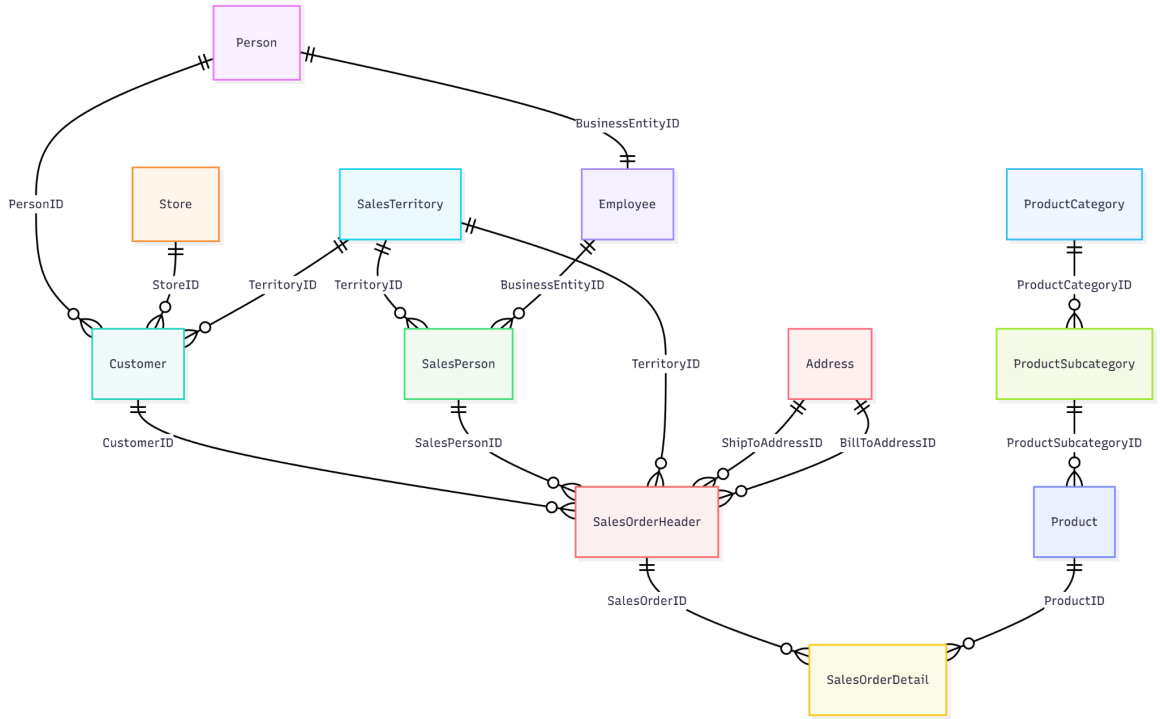
Gallegos Lozano Karen Scarlett:
Realización del **grafo** relacional:

Primero ordenar las entidades

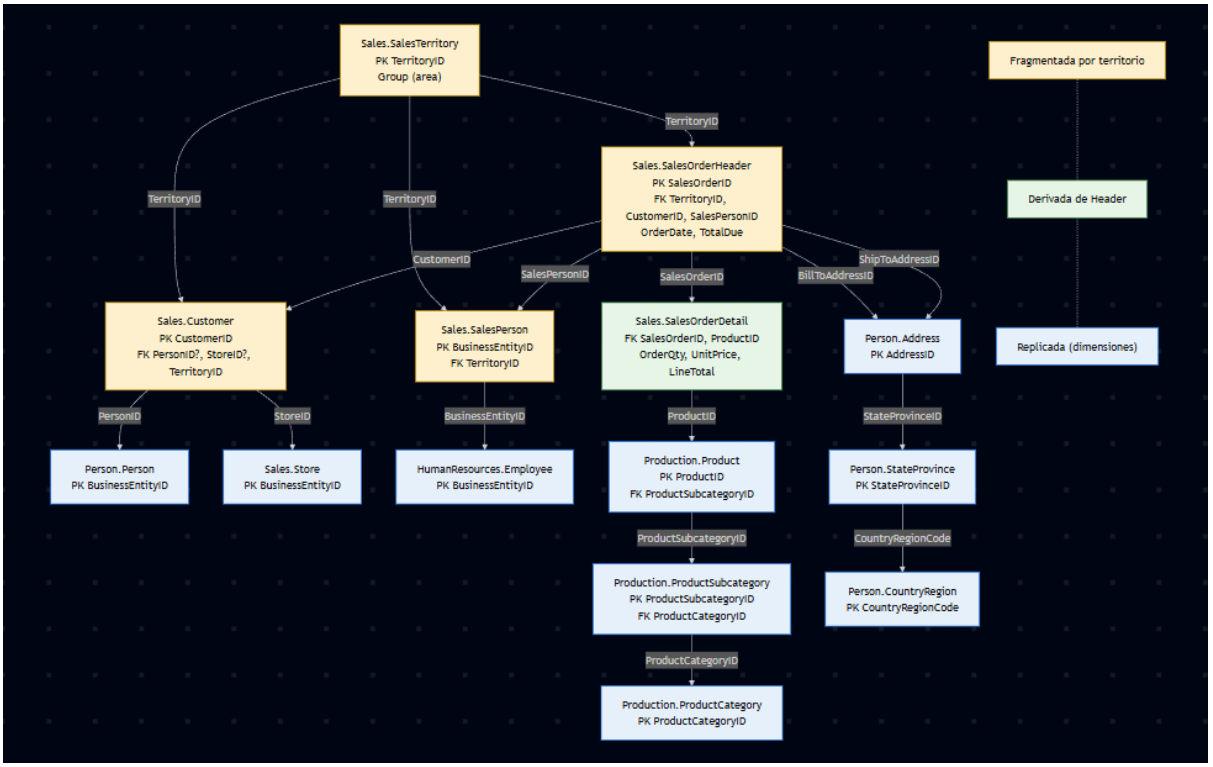
- Reconocer cuáles son las entidades principales
- Buscar las entidades que presenten una interrelación 1:N
- Buscar las interrelaciones N:M

Me dediqué a buscar las tablas relacionadas con TerritoryID y las tablas relacionadas con lo que nos pide el profesor.

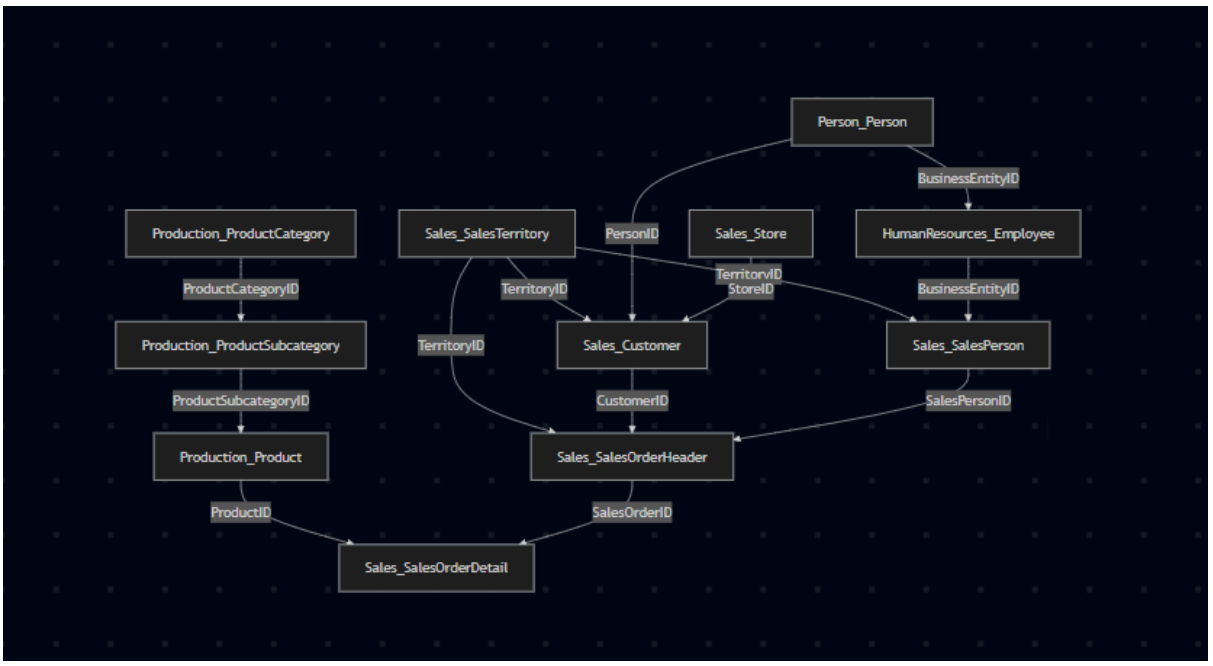
Este es el primer grafo creado, el problema de este es que estoy poniendo como principal a SalesOrderHeader pero todo empieza desde sales territory



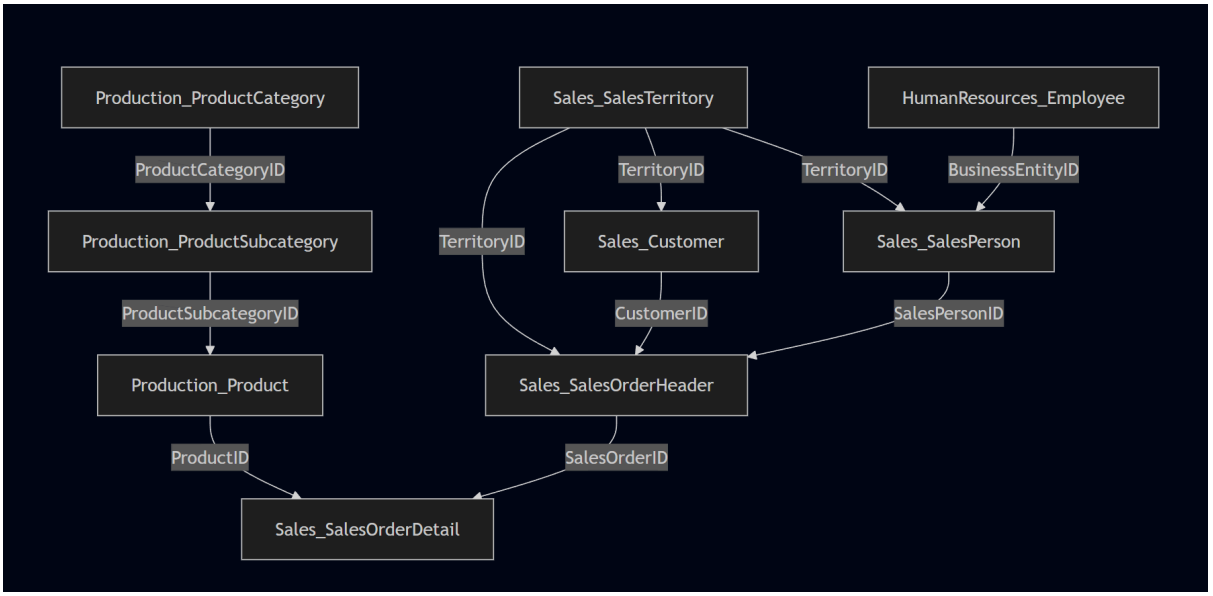
A Continuación realicé este grafo, no funcionó ya que las tablas no estaban ordenadas jerárquicamente y tenía tablas de sobra (no hacían absolutamente nada y no eran utilizadas).



Después de consultar al profesor este último grafo muestra la jerarquía de tablas.



Por último para el grafo definitivo, lo modifiqué porque estamos teniendo problemas para poder obtener información de las tablas person.person y sales. store. Solo las puse para tener como información más completa pero en si no son muy necesarias, por lo tanto el grafo ya quedaría de este modo de acuerdo a lo que haremos con nuestros diccionarios y fragmentación.



A partir de esto se fragmentaron las tablas y la fragmentación que se aplicó a cada una.

Gallegos Lozano Karen Scarlett, Bazaldua Montoya Jose Juan

Para el **inciso A**, tuvimos demasiados problemas para la realización de esta consulta, realicé un total de tres consultas tratando de que el resultado se ejecutará de forma correcta. Fue un fracaso, por lo tanto adjunto las tres consultas probadas. Nuestro mayor problema en cada una

de las consultas fue que el parámetro de @FechaOrden, nos arrojaba un error al pasar a DATETIME, la solución a esto fue poner un GETDATE().
Otro de nuestros problemas fue el remotebatch, esto lo usamos para ejecutar los procesos por lotes en el servidor de manera remota.

- Para esta consulta estoy insertando una venta en Header y Detail y que esto me devuelva el SalesOrderId, este procedure va a la venta por región y asegura que caigan juntos, ejecutando los INSERT y regresa un ID automáticamente.

```
CREATE OR ALTER PROCEDURE dbo.usp_RegistrarVentaRegion
   @Region        NVARCHAR(50),   -- 'AW_NA' o 'AW_EU' (acepto 'Europe' y lo normalizo)
   @CustomerID    INT,
   @SalesPersonID INT,
   @TerritoryID   INT,
   @ProductID     INT,
   @OrderQty      INT,
   @UnitPrice     MONEY,
   @FechaOrden    DATETIME
AS
BEGIN
   SET NOCOUNT ON;   --Evita que devuelvan mesajes sobre el # de filas afectadas
   SET XACT_ABORT ON; --Si ocurre un error durante la transaccion se aborta

   -- Normalizar región
   SET @Region = CASE
                 WHEN @Region IN (N'Europe', N'EU', N'Europa') THEN N'AW_EU'
                       WHEN @Region IN (N'North America', N'NA', N'Norte America', N'Norte
América') THEN N'AW_NA'
                 ELSE @Region
               END;
   --Busca en el diccionario el fragmento
   DECLARE @servidor SYSNAME, @bd SYSNAME;

   SELECT TOP (1)
      @servidor = servidor,
      @bd       = bd
   FROM dbo.diccionariodist
   WHERE tabla = N'SalesOrderHeader'
     AND predicado = @Region;

   IF @servidor IS NULL
      THROW 50000, 'Región no registrada en el diccionario.', 1;

   -- Asegurar colocalización Detail/Header (misma región, mismo fragmento)
   IF NOT EXISTS (
      SELECT 1
      FROM dbo.diccionariodist
      WHERE tabla = N'SalesOrderDetail'
        AND predicado = @Region
        AND servidor = @servidor
        AND bd = @bd
   )
          THROW 50001, 'SalesOrderDetail no está co-localizado con SalesOrderHeader para
esa región.', 1;

   -- Derivados
```

```
 begin
--para generar un ID
DECLARE @SalesOrderNumber NVARCHAR(50) =
    N'SO-' + REPLACE(CONVERT(NVARCHAR(36), NEWID()), N'-', N'');
-- Conveirte mis cantidades
DECLARE @LineTotal NUMERIC(38,6) =
    CONVERT(NUMERIC(38,6), @OrderQty) * CONVERT(NUMERIC(38,6), @UnitPrice);

        DECLARE @OrderDateIso NVARCHAR(30) = CONVERT(NVARCHAR(30),
@FechaOrden, 126);
                    DECLARE @SalesOrderNumberEsc NVARCHAR(100) =
REPLACE(@SalesOrderNumber, N'', N''''); -- escape comillas

-- Captura del ID devuelto por remoto
IF OBJECT_ID('tempdb..#NewSO', 'U') IS NOT NULL DROP TABLE #NewSO;
CREATE TABLE #NewSO (SalesOrderID INT NOT NULL);

-----------------------------------------------------------------
-- Batch que se ejecuta EN el remoto (un solo string, sin anidación)
-----------------------------------------------------------------
DECLARE @RemoteBatch NVARCHAR(MAX) = N'
SET NOCOUNT ON
USE ' + QUOTENAME (@bd) + N';

INSERT INTO dbo.SalesOrderHeader
(
    RevisionNumber, OrderDate, DueDate, ShipDate, Status,
    OnlineOrderFlag, SalesOrderNumber, PurchaseOrderNumber,
    AccountNumber, CustomerID, SalesPersonID, TerritoryID
)
VALUES
(
    0,
    CONVERT(datetime, ''' + @OrderDateIso + N''', 126),
    DATEADD(DAY, 7, CONVERT(datetime, ''' + @OrderDateIso + N''', 126)),
    CONVERT(datetime, ''' + @OrderDateIso + N''', 126),
    5,
    0,
    N''' + @SalesOrderNumberEsc + N''',
    NULL,
    NULL,
    ' + CAST(@CustomerID AS NVARCHAR(20)) + N',
    ' + CAST(@SalesPersonID AS NVARCHAR(20)) + N',
    ' + CAST(@TerritoryID AS NVARCHAR(20)) + N'
);

DECLARE @NewSalesOrderID INT = CAST(SCOPE_IDENTITY() AS INT);

-- Generar SalesOrderDetailID sin colisiones
DECLARE @NewDetailID INT;
SELECT @NewDetailID = ISNULL(MAX(SalesOrderDetailID), 0) + 1
FROM dbo.SalesOrderDetail WITH (UPDLOCK, HOLDLOCK);

INSERT INTO dbo.SalesOrderDetail
(
    SalesOrderID, SalesOrderDetailID, CarrierTrackingNumber,
    OrderQty, ProductID, SpecialOfferID,
    UnitPrice, UnitPriceDiscount,
    LineTotal, rowguid, ModifiedDate
)
VALUES
(
    @NewSalesOrderID,
    @NewDetailID,
    NULL,
```

```
        CAST(' + CAST(@OrderQty AS NVARCHAR(20)) + N' AS smallint),
        ' + CAST(@ProductID AS NVARCHAR(20)) + N',
        1,
        CAST(' + CONVERT(VARCHAR(50), @UnitPrice) + N' AS money),
        0,
        CAST(' + CONVERT(VARCHAR(100), @LineTotal) + N' AS numeric(38,6)),
        NEWID(),
        GETDATE()
    );

    SELECT @NewSalesOrderID AS SalesOrderID;
';

    -- Como AT necesita el nombre del linked server “hardcodeado”, armamos 1 nivel de SQL dinámico
    DECLARE @Cmd NVARCHAR(MAX) =
        N'INSERT INTO #NewSO(SalesOrderID)
          EXEC (N''' + REPLACE(@RemoteBatch, N'''', N'''''') + N''') AT ' + QUOTENAME(@servidor) + N';';

    -- DEBUG: descomenta si vuelve a fallar para ver el SQL exacto que se ejecuta
    -- SELECT @Cmd AS DebugCmd;

    EXEC (@Cmd);

    DECLARE @SalesOrderID INT = (SELECT TOP 1 SalesOrderID FROM #NewSO);

    SELECT
        @SalesOrderID AS SalesOrderID,
        @servidor     AS LinkedServer,
        @bd           AS BaseDatos;
END;
GO



------------------------------------------------------------
--. SP CONSULTA DISTRIBUIDA DE SALESORDERHEADER
------------------------------------------------------------
IF OBJECT_ID('dbo.usp_ConsultaOrderHeaderDistribuida', 'P') IS NOT NULL
    DROP PROCEDURE dbo.usp_ConsultaOrderHeaderDistribuida;
GO

CREATE PROCEDURE dbo.usp_ConsultaOrderHeaderDistribuida
@consulta NVARCHAR(MAX)
AS
BEGIN
    DECLARE @TerritoryID INT,
            @region NVARCHAR(50),
            @query NVARCHAR(MAX);

    -- Extraer TerritoryID del WHERE
    SELECT @TerritoryID = TRY_CAST(
        LTRIM(RTRIM(
            REPLACE(SUBSTRING(@consulta, CHARINDEX('=', @consulta)+1, 20), ';','')
        )) AS INT);

    IF @TerritoryID IS NULL
    BEGIN
        RAISERROR('No se pudo extraer TerritoryID.', 16, 1);
        RETURN;
    END;

    -- Buscar región
    SELECT @region =
        CASE [Group]
            WHEN 'North America' THEN 'AW_NA'
            ELSE [Group]
        END
```

```
    FROM AdventureWorks2022.Sales.SalesTerritory
    WHERE TerritoryID = @TerritoryID;

    IF @region IS NULL
    BEGIN
        RAISERROR('TerritoryID no existe en ninguna región.', 16, 1);
        RETURN;
    END;

    SET @query = 'WHERE TerritoryID = ' + CAST(@TerritoryID AS NVARCHAR);

    EXEC dbo.execquery @region, 'SalesOrderHeader', @query;
END;
GO
```

- Para esta consulta, estoy recibiendo un área, y decide a que fragmentoir, insertnado SalesOrderHeader en el servidor remoto correcto y regresa el SalesOrderID generado más el TerritoryID.

```
CREATE PROCEDURE dbo.usp_CrearSalesOrderHeaderDistribuido
    @Area          NVARCHAR(50),  -- 'North America', 'Europe' o 'AW_NA'/'AW_EU'
    @CustomerID    INT,
    @SalesPersonID INT = NULL
AS
BEGIN
    SET NOCOUNT ON;-- inicia el cuerpo


    -- normalizo la entrada de mi predicado
    DECLARE @Region NVARCHAR(50) =
        CASE
                WHEN @Area IN (N'North America', N'NA', N'Norte America', N'Norte América', N'AW_NA')
THEN N'AW_NA'
            WHEN @Area IN (N'Europe', N'EU', N'Europa', N'AW_EU') THEN N'AW_EU'
            ELSE @Area
        END;

    --Preparo el valor que existe en SalesTerritory[Group]
    DECLARE @GrupoTerritorio NVARCHAR(50) =
        CASE @Region
            WHEN N'AW_NA' THEN N'North America'
            WHEN N'AW_EU' THEN N'Europe'
            ELSE @Area
        END;


    --Para mi ruteo
    DECLARE @servidor SYSNAME, @bd SYSNAME;

    --Consuulta al diccionario
    SELECT TOP (1)
        @servidor = servidor,
        @bd       = bd
    FROM dbo.diccionariodist
    WHERE tabla = N'SalesOrderHeader'
      AND predicado = @Region
    ORDER BY id;

    IF @servidor IS NULL OR @bd IS NULL
        THROW 50011, 'No se encontró servidor/BD para SalesOrderHeader en el diccionario.', 1;

    -- Co-localización
    IF NOT EXISTS (
        SELECT 1
        FROM dbo.diccionariodist
        WHERE tabla = N'SalesOrderDetail'
```

```
        AND predicado = @Region
        AND servidor = @servidor
        AND bd = @bd
    )
        THROW 50012, 'SalesOrderDetail no está co-localizado con SalesOrderHeader para esa región.',
1;


    -- 2) Ejecutar INSERT en remoto (EXEC ... AT) y resolver TerritoryID ahí

    --genera la fecha
   DECLARE @OrderDateIso NVARCHAR(30) = CONVERT(NVARCHAR(30), GETDATE(), 126);
      DECLARE @GrupoEsc NVARCHAR(100) = REPLACE(@GrupoTerritorio, N'', N''''); -- escape de
comillas

    DECLARE @RemoteQuery NVARCHAR(MAX) = N'
SET NOCOUNT ON;
USE ' + QUOTENAME(@bd) + N';

DECLARE @TerritoryID INT;

SELECT TOP 1 @TerritoryID = TerritoryID
FROM dbo.SalesTerritory
WHERE [Group] = '' + @GrupoEsc + N''
ORDER BY TerritoryID;

IF @TerritoryID IS NULL
BEGIN
    SELECT CAST(NULL AS INT) AS NewSalesOrderID, CAST(NULL AS INT) AS TerritoryID;
    RETURN;
END;

DECLARE @SalesOrderNumber NVARCHAR(50) =
    N''SO-'' + REPLACE(CONVERT(NVARCHAR(36), NEWID()), N''-'', N'''');

INSERT INTO dbo.SalesOrderHeader
(
    RevisionNumber, OrderDate, DueDate, ShipDate, Status,
    OnlineOrderFlag, SalesOrderNumber, PurchaseOrderNumber,
    AccountNumber, CustomerID, SalesPersonID, TerritoryID
)
VALUES
(
    0,
    CONVERT(datetime, '' + @OrderDateIso + N'', 126),
    DATEADD(DAY, 7, CONVERT(datetime, '' + @OrderDateIso + N'', 126)),
    CONVERT(datetime, '' + @OrderDateIso + N'', 126),
    5,
    0,
    @SalesOrderNumber,
    NULL,
    NULL,
    ' + CAST(@CustomerID AS NVARCHAR(20)) + N',
       ' + CASE WHEN @SalesPersonID IS NULL THEN N'NULL' ELSE CAST(@SalesPersonID AS
NVARCHAR(20)) END + N',
    @TerritoryID
);

SELECT CAST(SCOPE_IDENTITY() AS INT) AS NewSalesOrderID,
       @TerritoryID AS TerritoryID;
';

    ------------------------------------------------------------------
    -- 3) Ejecutar remoto y capturar sin #temp: OPENQUERY + CTE + OUTPUT vars
    ------------------------------------------------------------------
    DECLARE @NewSalesOrderID INT = NULL;
    DECLARE @TerritoryID     INT = NULL;
```

```
    DECLARE @Cmd NVARCHAR(MAX) = N'
;WITH r AS
(
    SELECT NewSalesOrderID, TerritoryID
     FROM OPENQUERY(' + QUOTENAME(@servidor) + N', ''' + REPLACE(@RemoteQuery, N'''', N'''''')
+ N''')
)
SELECT
    @NewSalesOrderID = NewSalesOrderID,
    @TerritoryID     = TerritoryID
FROM r;
';

    EXEC sys.sp_executesql
        @Cmd,
        N'@NewSalesOrderID INT OUTPUT, @TerritoryID INT OUTPUT',
        @NewSalesOrderID OUTPUT,
        @TerritoryID OUTPUT;

    IF @NewSalesOrderID IS NULL OR @TerritoryID IS NULL
            THROW 50013, 'No se pudo crear el SalesOrderHeader en el fragmento (TerritoryID o ID
generado NULL).', 1;

    ------------------------------------------------------------------
    -- 4) Resultado final
    ------------------------------------------------------------------
    SELECT
        @NewSalesOrderID AS NewSalesOrderID,
        @TerritoryID     AS TerritoryID,
        @Region          AS Region,
        @servidor        AS LinkedServer,
        @bd              AS BaseDatos;
END;
GO
```

- Para la tercera prueba, traté de no consultar al diccionario para saber a qué fragmento va, trae el ruteo por región

```
CREATE PROCEDURE dbo.usp_RegistrarVentaArea_SinDiccionario
    @Area          NVARCHAR(50),   -- 'North America', 'Europe', 'AW_NA', 'AW_EU'
    @CustomerID    INT,
    @SalesPersonID INT,
    @TerritoryID   INT,
    @ProductID     INT,
    @OrderQty      INT,
    @UnitPrice     MONEY,
    @FechaOrden    DATETIME
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON;

    ------------------------------------------------------------------
    -- 1) Normalizar área a región fija (SIN diccionario)
    ------------------------------------------------------------------
    DECLARE @Region NVARCHAR(50) =
        CASE
            WHEN @Area IN (N'Europe', N'EU', N'Europa', N'AW_EU') THEN N'AW_EU'
                WHEN @Area IN (N'North America', N'NA', N'Norte America', N'Norte América', N'AW_NA')
THEN N'AW_NA'
            ELSE @Area
        END;

    IF @Region NOT IN (N'AW_NA', N'AW_EU')
        THROW 50000, 'Área no soportada. Usa: North America/Europe o AW_NA/AW_EU.', 1;
```

```
    ------------------------------------------------------------------
    -- 2) Derivados
    ------------------------------------------------------------------
    DECLARE @SalesOrderNumber NVARCHAR(50) =
        N'SO-' + REPLACE(CONVERT(NVARCHAR(36), NEWID()), N'-', N'');

    DECLARE @LineTotal NUMERIC(38,6) =
        CONVERT(NUMERIC(38,6), @OrderQty) * CONVERT(NUMERIC(38,6), @UnitPrice);

    DECLARE @OrderDateIso NVARCHAR(30) = CONVERT(NVARCHAR(30), @FechaOrden, 126);
        DECLARE  @SalesOrderNumberEsc  NVARCHAR(100)  =  REPLACE(@SalesOrderNumber,  N'',
N''''); -- escape

    ------------------------------------------------------------------
    -- 3) Batch que se ejecuta EN EL FRAGMENTO (remoto)
    --    Inserta Header + Detail y regresa SalesOrderID
    ------------------------------------------------------------------
    DECLARE @RemoteBatch NVARCHAR(MAX) = N'
SET NOCOUNT ON;

INSERT INTO dbo.SalesOrderHeader
(
    RevisionNumber, OrderDate, DueDate, ShipDate, Status,
    OnlineOrderFlag, SalesOrderNumber, PurchaseOrderNumber,
    AccountNumber, CustomerID, SalesPersonID, TerritoryID
)
VALUES
(
    0,
    CONVERT(datetime, ''' + @OrderDateIso + N''', 126),
    DATEADD(DAY, 7, CONVERT(datetime, ''' + @OrderDateIso + N''', 126)),
    CONVERT(datetime, ''' + @OrderDateIso + N''', 126),
    5,
    0,
    N''' + @SalesOrderNumberEsc + N''',
    NULL,
    NULL,
    ' + CAST(@CustomerID AS NVARCHAR(20)) + N',
    ' + CAST(@SalesPersonID AS NVARCHAR(20)) + N',
    ' + CAST(@TerritoryID AS NVARCHAR(20)) + N'
);

DECLARE @NewSalesOrderID INT = CAST(SCOPE_IDENTITY() AS INT);

DECLARE @NewDetailID INT;
SELECT @NewDetailID = ISNULL(MAX(SalesOrderDetailID), 0) + 1
FROM dbo.SalesOrderDetail WITH (UPDLOCK, HOLDLOCK);

INSERT INTO dbo.SalesOrderDetail
(
    SalesOrderID, SalesOrderDetailID, CarrierTrackingNumber,
    OrderQty, ProductID, SpecialOfferID,
    UnitPrice, UnitPriceDiscount,
    LineTotal, rowguid, ModifiedDate
)
VALUES
(
    @NewSalesOrderID,
    @NewDetailID,
    NULL,
    CAST(' + CAST(@OrderQty AS NVARCHAR(20)) + N' AS smallint),
    ' + CAST(@ProductID AS NVARCHAR(20)) + N',
    1,
    CAST(' + CONVERT(VARCHAR(50), @UnitPrice) + N' AS money),
    0,
    CAST(' + CONVERT(VARCHAR(100), @LineTotal) + N' AS numeric(38,6)),
```

```
    NEWID(),
    GETDATE()
);

SELECT @NewSalesOrderID AS SalesOrderID;
';

    ----------------------------------------------------------------
    -- 4) Ejecutar REMOTO en el servidor correspondiente (sin diccionario)
    --    y capturar el ID SIN #temp (usamos TABLE VARIABLE)
    ----------------------------------------------------------------
    DECLARE @R TABLE (SalesOrderID INT);

    IF @Region = N'AW_NA'
    BEGIN
        INSERT INTO @R (SalesOrderID)
        EXEC ( @RemoteBatch ) AT [AW_NA];
    END
    ELSE
    BEGIN
        INSERT INTO @R (SalesOrderID)
        EXEC ( @RemoteBatch ) AT [AW_EU];
    END

    ----------------------------------------------------------------
    -- 5) Respuesta
    ----------------------------------------------------------------
    DECLARE @SalesOrderID INT = (SELECT TOP 1 SalesOrderID FROM @R);

    SELECT
        @SalesOrderID AS SalesOrderID,
        @Region       AS Region;
END;
GO

EXEC dbo.usp_RegistrarVentaArea_SinDiccionario
    @Area = N'North America',
    @CustomerID = 11000,
    @SalesPersonID = 279,
    @TerritoryID = 1,
    @ProductID = 776,
    @OrderQty = 2,
    @UnitPrice = 25.00,
    @FechaOrden = GETDATE();
```

Puede que la consulta sea mucho más sencilla de lo que parece, quiero creer que las consultas truenan porque la tabla que estamos insertando de SalesOrdeHeader no trae la llave primaria y el cliente ingresa manualmente el ID, por lo tanto se rompe la consulta, tratamos de hacer que la tabla de orderheader y orderDetail nos regresará automáticamente el ID modificando la base de cada una de las regiones en los servidores, lamentablemente aún nos marcaba errores.

Gallegos Lozano Karen Scarlett

Para la realización del **inciso C**, la consulta mostrada en el laboratorio es remota. Estoy creando tres procedure y en dos de ellos agregó un parámetro por región que suma la venta por nodo y por último agrego otro parámetro que suma ambos. Los parámetros los leo mediante el linked server.

Reescribo el código de forma distribuida

```
DECLARE @FechaInicio DATE = '2013-01-01';
DECLARE @FechaFin    DATE = '2013-12-31';
```

```
WITH Ventas AS
(
   SELECT TotalDue
   FROM [SRV_NA].[AW_NA].dbo.SalesOrderHeader
   WHERE OrderDate >= @FechaInicio
     AND OrderDate <  DATEADD(DAY, 1, @FechaFin)

   UNION ALL

   SELECT TotalDue
   FROM [SRV_EU].[AW_EU].dbo.SalesOrderHeader
   WHERE OrderDate >= @FechaInicio
     AND OrderDate <  DATEADD(DAY, 1, @FechaFin)
)
SELECT SUM(TotalDue) AS TotalVentasGlobal
FROM Ventas;
```

En conclusión, el proyecto fue para reforzar lo visto en clase como las consultas remotas, distribuidas, las conexiones remotas entre servidor - cliente y el uso de una API REST. Durante la realización del mismo surgieron bastantes problemáticas ya que no logré diferencias entre una consulta remota a una distribuida hasta el final del proyecto, pero se logró llegar al resultado (no como una consulta distribuida del todo) pero en este apartado adjunto el cómo deberían haber sido las consultas, de este modo se ha podido plantear de manera correcta el cómo deberían ser las consultas y que hay una manera más sencilla a diferencia de realizar consultas muy rebuscadas. Concluyendo, espero hacer un mejor trabajo ya que al final logré entender mejor lo que se nos estaba solicitando.

Bazaldua Montoya Jose Juan realización del inciso B y D

Inciso B:

Para realizar este inciso comencé con la idea de determinar los parámetros de entrada del SP, FechaInicio, FechaFinal y Region, siendo esta ultima la parte que se compara con el diccionario de distribución par saber que nodo ir a resolver la consulta accediendo al servidor de manera remota ejecutando una consulta dinámica con el esquema de 4 partes. En este punto se intuye que la consulta entra mas en la definición de ser una simple consulta remota a una distribuida principalmente por especificar a que nodo ira a resolver.

Para que esta consulta pase a ser Distribuida se requiere NO ACCEDER a un solo nodos sino a los dos, quitando el filtro de la Region y solo dejando el rango de fechas en las que queremos ver el resultado, Usando CTE para acceder al diccionario y obtener la región (predicado en el diccionario), Haciendo el calculo y Realizando el UNION para mostrar ambos resultados de cada servidor.

La construcción de la consulta mostrada en clase se realizo comenzando con validaciones y control de errores para las fechas, lo cual podríamos haber usado TRIGGER para ese control.

Construimos el CTE para el mapeo por región, hacemos el COUNT () y SUM () para NumOrdenes y TotalVentas respectivamente con esto hacemos el calculo encesario para la consulta, después a tra ves de la condición de reunión (join) validando FechaInicio y FechaFinal con OrderDate. Por ultimo solo aplicamos el EXEC a sql (STRING) para ejecutar la consulta en los nodos

Para llevar acabo la Consulta distribuida:

Primero en el SP solo declaramos las fechas de inicio y fin, Modificamos el Mapeo (with) para que solo filtre del diccionario las tabla SalesOrderHeader

```
CREATE OR ALTER PROCEDURE dbo.sp_TotalVentasPorTerritorio
  @FechaInicio DATE,
    @FechaFin    DATE,

WITH Mapeo AS (
    SELECT DISTINCT servidor, bd, predicado
    FROM dbo.diccionariodist
    WHERE tabla = N'SalesOrderHeader'

    )
```

Con esto hacemos que no vaya a un solo nodo sino a los dos, dentro de sql la consulta se ejecuta practicante de la misma manera a excepción de que repetir el código para hacer la consulta en ambos especificando el predicado del diccionario de manera predeterminada y atraves de UNION unir las dos consultas resueltas en cada nodo.

```
N'SELECT N''' + RegionEU+ N''' AS Region,
    st.TerritoryID,
    st.Name AS Territorio,
    COUNT(*) AS NumOrdenes,
    SUM(soh.TotalDue) AS TotalVentas
 FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader AS soh
 JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesTerritory   AS st
        ON soh.TerritoryID = st.TerritoryID
 WHERE soh.OrderDate >= @pInicio
  AND soh.OrderDate  < DATEADD(day, 1, @pFin)
 GROUP BY st.TerritoryID, st.Name',
N' UNION ALL
N'SELECT N''' + RegionNA + N''' AS Region,
```

```
        st.TerritoryID,
        st.Name AS Territorio,
        COUNT(*) AS NumOrdenes,
        SUM(soh.TotalDue) AS TotalVentas
 FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader AS soh
 JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesTerritory   AS st
        ON soh.TerritoryID = st.TerritoryID
 WHERE soh.OrderDate >= @pInicio
   AND soh.OrderDate  < DATEADD(day, 1, @pFin)
 GROUP BY st.TerritoryID, st.Name','
'
```

Con estas modificaciones se espera ver el mismo resultado obtenido en la consulta remota que s presento en el laboratorio, pero esta vez de manera distribuida acudiendo a los dos nodos resolviendo la consulta y al final uniéndolo todo en una consulta lógica.

Inciso D:

Como en mi inciso anterior comenzamos con determinar los parámetros de entrada del SP d eigual manera para obtener el producto mas y menos vendido pedimos un rango de fechas para acotarlo dentro de un periodo de tiempo, la consulta mostrada en el laboratorio se muestra con mas parámetros como la Region y la Metrica, siendo esta ultima la forma de ordenamiento que tiene la consulta ya sea por importe o por cantidad, desde aquí los factores que hace mas redundante el código es la cantidad de parámetros que maneje ya que en realidad con acotar la consulta por fechas es mas que suficiente, como en el inciso anterior el especificar la región hace que solo vaya aun solo nodo a realizar la consulta (Remoto) en lugar de resolver y devolver de ambos nodos (distribuida).

Se usaron como las CTEs para la realización de la consulta comenzando con validaciones para evitar la falta de parámetros.

```
-- Validaciones básicas
IF @FechaInicio IS NULL OR @FechaFin IS NULL
        THROW 51001, 'Debe proporcionar @FechaInicio y @FechaFin.', 1;

IF @FechaFin < @FechaInicio
        THROW 51002, '@FechaFin no puede ser menor que @FechaInicio.', 1;
```

```
IF @Metrica NOT IN (N'Cantidad', N'Importe')
        THROW 51004, 'Parametro @Metrica inválido. Use "Cantidad" o "Importe".', 1;
```

En este punto el procedimiento queda igual que el original a ecepcion de los parámetros de entrada

```
CREATE OR ALTER PROCEDURE dbo.sp_ProductoMasYMenosVendidoPorCategoria
   @FechaInicio DATE,
        @FechaFin       DATE,
        @Categoria   NVARCHAR(100),
        @Metrica          NVARCHAR(20)  = N'Cantidad'    -- Especificamos por default que se ordene por cantidad y omitimos la Region
SELECT @sql =
        STRING_AGG(
        N'SELECT N''' + Region + N''' AS Region,
          p.ProductID,
          p.Name AS Producto,
          ps.Name AS Subcategoria,
          pc.Name AS Categoria,
          SUM(sod.OrderQty)  AS Cantidad,
          SUM(sod.LineTotal) AS Importe
      FROM ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderDetail AS sod
      JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.SalesOrderHeader AS soh
        ON sod.SalesOrderID = soh.SalesOrderID
      JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.Product AS p
        ON sod.ProductID = p.ProductID
      JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.ProductSubcategory AS ps
        ON p.ProductSubcategoryID = ps.ProductSubcategoryID
      JOIN ' + QUOTENAME(servidor) + N'.' + QUOTENAME(bd) + N'.dbo.ProductCategory AS pc
        ON ps.ProductCategoryID = pc.ProductCategoryID
     WHERE soh.OrderDate >= @pInicio
       AND soh.OrderDate  < DATEADD(day, 1, @pFin)
```

```
        AND pc.Name = @pCategoria

    GROUP BY p.ProductID, p.Name, ps.Name, pc.Name',

        N' UNION ALL '
```

Mantenemos la lógica de la consulta actual como se puede observar en el UNION ALL al final de esta consulta es lo que hace la diferencia en hacer la distribuida ya que no accedemos a un solo nodo si no a los dos en este caso para resolver en cada nodo y unir ambas resoluciones de una sola salida, de igual manera el uso de CTE como mapeo para leer el diccionario y otros mas para reutilizar el código

```
-- Ordenamientos según la métrica (se usarán dentro de CTEs separadas)

DECLARE @OrdenMax NVARCHAR(200) =

        CASE WHEN @Metrica = N'Cantidad'

        THEN N'Cantidad DESC, Importe DESC, ProductID'

        ELSE N'Importe  DESC, Cantidad DESC, ProductID'

        END;


DECLARE @OrdenMin NVARCHAR(200) =

        CASE WHEN @Metrica = N'Cantidad'

        THEN N'Cantidad ASC, Importe ASC, ProductID'

        ELSE N'Importe  ASC, Cantidad ASC, ProductID'
```

Siendo estos últimos mostrados que se podrían omitir para evitar los ordenamientos, y solo irnos con la métrica por cantidad para realizar la consulta esto para evitar que si un producto es mas vendido por importe puede ser un alto costo y una cantidad baja.

En general hay muchos bloques del batch que se pueden omitir haciendo la consulta mucho mas simple de lo que mostro, simplemente haciendo los cálculos pertinentes en cada nodo y devolviendo todo en una salida a traves de UNION ALL.

Después de desarrollar los procedimientos sp_TotalVentasPorTerritorio y sp_ProductoMasYMenosVendidoPorCategoria, y de la última sesión con el profesor entendí mejor cómo funciona la lógica detrás de las consultas distribuidas y por qué son necesarias en un proyecto con fragmentación de datos. En nuestro caso, los datos están fragmentados horizontalmente por región y territorio, lo que significa que cada servidor almacena una parte del conjunto completo. Generalmente con el concepto de Fragmentación a un nivel conceptual entendí bien el como se lleva acabo las razones del por que usar una fragmentación, tener acceso a una sola parte del “todo” por decirlo alguna manera, el cómo accedemos a esos datos dependiendo de la aplicación o de la consulta. Me queda claro el como implementar lo distribuido o lo remoto dentro de las consultas, para el proyecto

presentado era necesario irnos mas por lo distribuido, los resultados mostrados en laboratorio si bien se apegan a lo esperado ver en las consultas, hay muchas otras maneras en las que con mucho menos código y sin ser tan rebuscado, podemos llegar el mismo resultado.

Algunas buenas practicas para optimizar mas las consultas de las cuales me fui percatando: Evitar ordenamientos si no son necesarios, buscar la opción del uso de vistas, evitar el uso de TOP 1 para seleccionar, considerar el uso de TRY… CATCH y TRIGGER para manejo de errores, asi como otras practicas que so buenas saberlas ene este punto para en proyectos futuros poder implementar de manera más profesional.

En resumen, los incisos que implementé demuestran que, con fragmentación horizontal por región/territorio y una orquestación distribuida que pre-agrega por nodo, puedo obtener métricas confiables de manera escalable. La clave es simplificar los SP, evitar ordenamientos innecesarios, parametrizar.

Benites Garcilazo Luis Fernando:

En este proyecto implementamos la fragmentación lógica de una base de datos en dos instancias independientes de SQL Server divididas por región (AW_NA y AW_EU). El trabajo inició desde el diseño de tablas y la construcción de las consultas de análisis (A–F), ajustándolas conforme avanzó el desarrollo. Posteriormente creamos una API en Python que operó primero en entorno local y desplegada en AWS, utilizando dos bases de datos RDS y un servidor EC2. Este despliegue implicó la configuración de VPC, grupos de seguridad y el acceso a RDS mediante SSH desde la EC2.

Durante el proceso se presentaron problemas de consistencia y migración de datos al pasar a la nube, pero se logró ejecutar correctamente la API y validar su funcionalidad mediante pruebas con curl. La API recibe parámetros del cliente, ejecuta consultas y procedimientos almacenados en el nodo correspondiente y regresa resultados consolidados.

Debido a que en AWS no fue posible establecer linked servers, se desarrollaron dos versiones del archivo app.py, cada una con conexiones adecuadas a sus respectivas bases de datos, realizando pequeñas adaptaciones según el entorno. Esta estrategia permitió demostrar consultas distribuidas: un mismo endpoint puede invocar procesos sobre distintos nodos sin centralizar los datos físicamente.

En términos de aprendizaje, el proyecto aportó experiencia práctica en diseño de bases de datos, fragmentación, despliegue de aplicaciones en AWS, configuración de red y seguridad, ejecución remota por SSH, y consumo/validación de APIs desde consola.

Pienso que de fallar tanto pude aprender muchísimo.

El proyecto presentado consiste en el diseño y desarrollo de una base de datos distribuida para gestionar información de ventas en diferentes regiones geográficas, específicamente en Norteamérica y Europa. Este sistema utiliza la fragmentación horizontal y primaria de las tablas de ventas para mejorar la eficiencia y el rendimiento de las consultas. La fragmentación horizontal divide los datos según la región, mientras que la fragmentación primaria se aplica a las órdenes de ventas, permitiendo una distribución coherente de la información a través de los diferentes fragmentos.

Además, se implementaron procedimientos almacenados (stored procedures) para realizar tareas como el registro de ventas, la consulta de ventas por territorio y la obtención de datos de productos más vendidos, empleados con mayor volumen de ventas y clientes con más órdenes. El proyecto también incluye la creación de una API utilizando Flask para permitir consultas y registros de ventas a través de una interfaz web, lo que facilita el acceso a la información desde diferentes ubicaciones.

En conclusión, este proyecto tiene como objetivo optimizar el manejo de grandes volúmenes de datos distribuidos en distintas regiones mediante técnicas de fragmentación, replicación y consultas remotas, lo que mejora el acceso a la información y facilita la toma de decisiones comerciales.

Sanchez Gallardo Janeth

Conclusión del ejercicio e: Empleado top en ventas por región

En este ejercicio se logró identificar al empleado con mayores ventas en cada región, aun cuando la información se encuentra distribuida en distintos servidores.

Mediante el uso de SQL dinámico, se consolidaron los datos de ventas provenientes de múltiples fuentes en una sola consulta lógica, permitiendo un análisis global sin necesidad de centralizar físicamente la información.

El uso de CTE (Common Table Expressions) facilitó la organización del proceso: primero unificando los datos, luego agrupando las ventas por región y empleado, y finalmente aplicando la función RANK() para determinar al empleado con el mejor desempeño en cada región.

Este ejercicio demuestra cómo SQL permite realizar análisis comparativos avanzados en entornos distribuidos, manteniendo eficiencia, escalabilidad y claridad en el código, además de resaltar la importancia de las funciones analíticas para la toma de decisiones basadas en datos.

Conclusión del ejercicio f: Cliente con más órdenes por región

En este ejercicio se determinó el cliente que realizó la mayor cantidad de órdenes en cada región, integrando información proveniente de diferentes servidores mediante consultas distribuidas.

La estrategia utilizada permitió contabilizar las órdenes por cliente, consolidarlas por región y seleccionar al cliente más relevante usando funciones de ranking.

La implementación de SQL dinámico y CTEs permitió manejar la fragmentación de los datos de forma transparente, evitando duplicidades y asegurando resultados consistentes. El uso de COUNT() junto con RANK() facilitó la identificación precisa del cliente top por región.

Este ejercicio evidencia cómo SQL puede emplearse eficazmente para análisis de comportamiento del cliente, incluso en arquitecturas distribuidas, proporcionando información clave para estrategias comerciales, segmentación de clientes y optimización de ventas.

Conclusión general

Los ejercicios e y f demostraron la capacidad de SQL para realizar análisis avanzados en entornos de bases de datos distribuidas, permitiendo integrar información proveniente de múltiples servidores como si se tratara de una sola fuente de datos. A través del uso de SQL dinámico, consultas distribuidas y CTE, se logró consolidar, agrupar y analizar grandes volúmenes de información sin necesidad de centralizar físicamente los datos.

El empleo de funciones analíticas como RANK(), junto con operaciones de agregación como SUM() y COUNT(), facilitó la identificación de los empleados con mayores ventas y los clientes con mayor número de órdenes por región, proporcionando resultados precisos y estructurados. Esto evidencia la importancia de diseñar consultas eficientes que permitan extraer información relevante para la toma de decisiones estratégicas.

En conjunto, estos ejercicios refuerzan el valor de SQL como una herramienta poderosa, flexible y escalable para el análisis de datos empresariales en arquitecturas distribuidas, destacando buenas prácticas en el manejo de datos, optimización de consultas y obtención de indicadores clave de desempeño.