UNIVERSIDAD PERUANA LOS ANDES

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Base de Datos II

AUTOR

CONDOR HUAMAN ERICK

DOCENTE

RAUL FERNANDEZ, Bejarano

CICLO

V

HUANCAYO – PERÚ – 2025

Practica-semana 13

Practica Semana 16

1. Enunciado del Proyecto

Título: *Implementación y Carga Automatizada de un Sistema de Gestión de Ventas Corporativas (EnterpriseSales).*

Objetivo: *Diseñar e implementar una base de datos relacional para el control de inventarios y ventas, asegurando la integridad referencial. Se requiere automatizar la generación de datos de prueba mediante estructuras de control (bucles) y validar la consistencia de la información mediante reportes analíticos de rendimiento y stock.*

2. Script Completo (Consolidado)

SQL

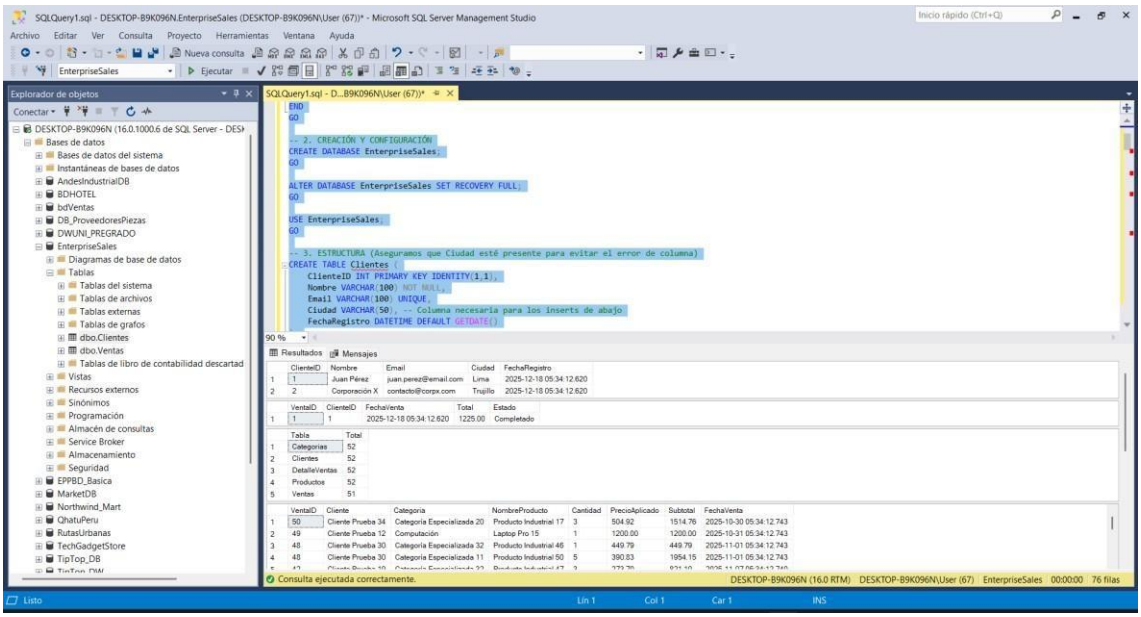
```
USE master;
GO

-- 1. ELIMINACIÓN FORZADA (Cierra conexiones activas antes de borrar)
IF EXISTS (SELECT name FROM sys.databases WHERE name = 'EnterpriseSales') BEGIN
    -- Cambia a SINGLE_USER para desconectar a otros usuarios y borra
    ALTER DATABASE EnterpriseSales SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
    DROP DATABASE EnterpriseSales;
END
GO

-- 2. CREACIÓN Y CONFIGURACIÓN
CREATE DATABASE EnterpriseSales;
GO

ALTER DATABASE EnterpriseSales SET RECOVERY FULL; GO

USE EnterpriseSales;
GO

-- 3. ESTRUCTURA (Aseguramos que Ciudad esté presente para evitar el error de columna)
CREATE TABLE Clientes (
    ClienteID INT PRIMARY KEY IDENTITY(1,1),
    Nombre VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Ciudad VARCHAR(50), -- Columna necesaria para los inserts de abajo
    FechaRegistro DATETIME DEFAULT GETDATE()
);

CREATE TABLE Categorias (
    CategoriaID INT PRIMARY KEY IDENTITY(1,1),      NombreCategoria VARCHAR(50) NOT NULL
);

CREATE TABLE Productos (
    ProductoID INT PRIMARY KEY IDENTITY(1,1),
    NombreProducto VARCHAR(100) NOT NULL,
    CategoriaID INT FOREIGN KEY REFERENCES Categorias(CategoriaID),
```

```
    PrecioUnitario DECIMAL(18,2) NOT NULL,
    StockActual INT DEFAULT 0
);

CREATE TABLE Ventas (
    VentaID INT PRIMARY KEY IDENTITY(1,1),
    ClienteID INT FOREIGN KEY REFERENCES Clientes(ClienteID),
    FechaVenta DATETIME DEFAULT GETDATE(),
    Total DECIMAL(18,2) NOT NULL,
    Estado VARCHAR(20) DEFAULT 'Completado'
);

CREATE TABLE DetalleVentas (
    DetalleID INT PRIMARY KEY IDENTITY(1,1),
    VentaID INT FOREIGN KEY REFERENCES Ventas(VentaID),
    ProductoID INT FOREIGN KEY REFERENCES Productos(ProductoID),
    Cantidad INT NOT NULL,
    PrecioAplicado DECIMAL(18,2) NOT NULL
);
GO

-- 4. CARGA DE DATOS
INSERT INTO Categorias (NombreCategoria) VALUES ('Electrónica'), ('Computación');
INSERT INTO Clientes (Nombre, Email, Ciudad) VALUES  ('Juan Pérez',
'juan.perez@email.com', 'Lima'),
('Corporación X', 'contacto@corpx.com', 'Trujillo');

INSERT INTO Productos (NombreProducto, CategoriaID, PrecioUnitario, StockActual)
VALUES
('Laptop Pro 15', 2, 1200.00, 50),
('Mouse Ergonómico', 1, 25.00, 200);

INSERT INTO Ventas (ClienteID, Total) VALUES (1, 1225.00);
INSERT INTO DetalleVentas (VentaID, ProductoID, Cantidad, PrecioAplicado) VALUES
(1, 1, 1, 1200.00),
(1, 2, 1, 25.00);
GO

-- 5. VERIFICACIÓN FINAL
SELECT * FROM Clientes;
SELECT * FROM Ventas




USE EnterpriseSales;
GO
```

```sql
-- 1. Insertar 50 Categorías
DECLARE @i INT = 1;
WHILE @i <= 50
BEGIN
    INSERT INTO Categorias (NombreCategoria)
    VALUES ('Categoría Especializada ' + CAST(@i AS VARCHAR(10)));
    SET @i = @i + 1;
END;

-- 2. Insertar 50 Clientes
SET @i = 1;
WHILE @i <= 50
BEGIN
    INSERT INTO Clientes (Nombre, Email, Ciudad, FechaRegistro)
    VALUES (
        'Cliente Prueba ' + CAST(@i AS VARCHAR(10)),
        'usuario' + CAST(@i AS VARCHAR(10)) + '@empresa.com',
        CASE WHEN @i % 2 = 0 THEN 'Lima' ELSE 'Provincias' END,
        GETDATE()
    );
    SET @i = @i + 1;
END;

-- 3. Insertar 50 Productos
SET @i = 1;
WHILE @i <= 50
BEGIN
    INSERT INTO Productos (NombreProducto, CategoriaID, PrecioUnitario,
StockActual)
    VALUES (
        'Producto Industrial ' + CAST(@i AS VARCHAR(10)),
        (SELECT TOP 1 CategoriaID FROM Categorias ORDER BY NEWID()), -- Asigna
categoría aleatoria
        RAND() * 500 + 10, -- Precio aleatorio entre 10 y 510
        CAST(RAND() * 100 AS INT)
    );
    SET @i = @i + 1;
END;

-- 4. Insertar 50 Ventas
SET @i = 1;
WHILE @i <= 50
BEGIN
    INSERT INTO Ventas (ClienteID, Total, FechaVenta)
    VALUES (
        (SELECT TOP 1 ClienteID FROM Clientes ORDER BY NEWID()), -- Cliente aleatorio
        0, -- Se actualizará después con el detalle
        DATEADD(DAY, -@i, GETDATE()) -- Ventas en diferentes días
    );
```

```
    SET @i = @i + 1;
END;

-- 5. Insertar 50 DetalleVentas y actualizar Totales
SET @i = 1;
WHILE @i <= 50
BEGIN
    DECLARE @VentaID INT = (SELECT TOP 1 VentaID FROM Ventas ORDER BY NEWID());
    DECLARE @ProdID INT = (SELECT TOP 1 ProductoID FROM Productos ORDER BY NEWID());
    DECLARE @Precio DECIMAL(18,2) = (SELECT PrecioUnitario FROM Productos WHERE
ProductoID = @ProdID);
    DECLARE @Cant INT = CAST(RAND() * 5 + 1 AS INT);

    INSERT INTO DetalleVentas (VentaID, ProductoID, Cantidad, PrecioAplicado)
    VALUES (@VentaID, @ProdID, @Cant, @Precio);
    -- Actualizar el total de la venta sumando el nuevo detalle
    UPDATE Ventas SET Total = Total + (@Precio * @Cant) WHERE VentaID = @VentaID;
    SET @i = @i + 1;
END;
GO

-- VERIFICACIÓN: Contar registros generados
SELECT 'Clientes' as Tabla, COUNT(*) as Total FROM Clientes
UNION SELECT 'Categorias', COUNT(*) FROM Categorias
UNION SELECT 'Productos', COUNT(*) FROM Productos
UNION SELECT 'Ventas', COUNT(*) FROM Ventas
UNION SELECT 'DetalleVentas', COUNT(*) FROM DetalleVentas;



--codigo de verificacion de datos de base de datos




USE EnterpriseSales;
GO

-- 1. REPORTE GENERAL DE VENTAS (Muestra el flujo completo del negocio)
-- Este reporte es ideal para verificar la consistencia de datos tras un Failover.
SELECT
    V.VentaID,
    C.Nombre AS Cliente,
    CAT.NombreCategoria AS Categoria,
    P.NombreProducto,
    DV.Cantidad,
    DV.PrecioAplicado,
```

```sql
    (DV.Cantidad * DV.PrecioAplicado) AS Subtotal,
    V.FechaVenta
FROM Ventas V
INNER JOIN Clientes C ON V.ClienteID = C.ClienteID
INNER JOIN DetalleVentas DV ON V.VentaID = DV.VentaID
INNER JOIN Productos P ON DV.ProductoID = P.ProductoID
INNER JOIN Categorias CAT ON P.CategoriaID = CAT.CategoriaID
ORDER BY V.VentaID DESC;

-- 2. RESUMEN DE REGISTROS POR TABLA
-- Útil para confirmar que los 50 registros por tabla se insertaron correctamente.
SELECT 'Clientes' AS Tabla, COUNT(*) AS Cantidad FROM Clientes UNION ALL
SELECT 'Categorías', COUNT(*) FROM Categorias
UNION ALL
SELECT 'Productos', COUNT(*) FROM Productos
UNION ALL
SELECT 'Ventas', COUNT(*) FROM Ventas
UNION ALL
SELECT 'Detalle Ventas', COUNT(*) FROM DetalleVentas;
-- 3. PRODUCTOS CON BAJO STOCK
-- Simula un reporte administrativo de inventario.
SELECT NombreProducto, StockActual
FROM Productos
WHERE StockActual < 10
ORDER BY StockActual ASC;

-- 4. TOP 5 CLIENTES QUE MÁS HAN COMPRADO
SELECT TOP 5
    C.Nombre,
    SUM(V.Total) AS InversionTotal
FROM Clientes C
JOIN Ventas V ON C.ClienteID = V.ClienteID
GROUP BY C.Nombre
ORDER BY InversionTotal DESC;
```



3. Explicación Técnica

Eliminación Forzada: *El uso de* `SET SINGLE_USER WITH ROLLBACK IMMEDIATE` *es una técnica avanzada para garantizar que la base de datos se elimine incluso si hay administradores o aplicaciones conectadas, evitando el error de "Database in use".*

Modelo Relacional: *Se implementan llaves primarias (*`PRIMARY KEY`*) e incrementales (*`IDENTITY`*) para asegurar la unicidad. Las llaves foráneas (*`FOREIGN KEY`*) garantizan que no existan ventas de productos que no existen o para clientes inexistentes.*

Lógica de Negocio en SQL: *Se utiliza la función* `NEWID()` *dentro de un* `ORDER BY` *para seleccionar registros aleatorios de otras tablas durante la carga de datos, simulando un comportamiento de compra real.*

Integridad de Cálculo: *El script incluye un* `UPDATE` *posterior a la inserción en* `DetalleVentas` *para asegurar que el campo* `Total` *de la tabla* `Ventas` *coincida con la suma de sus detalles.*

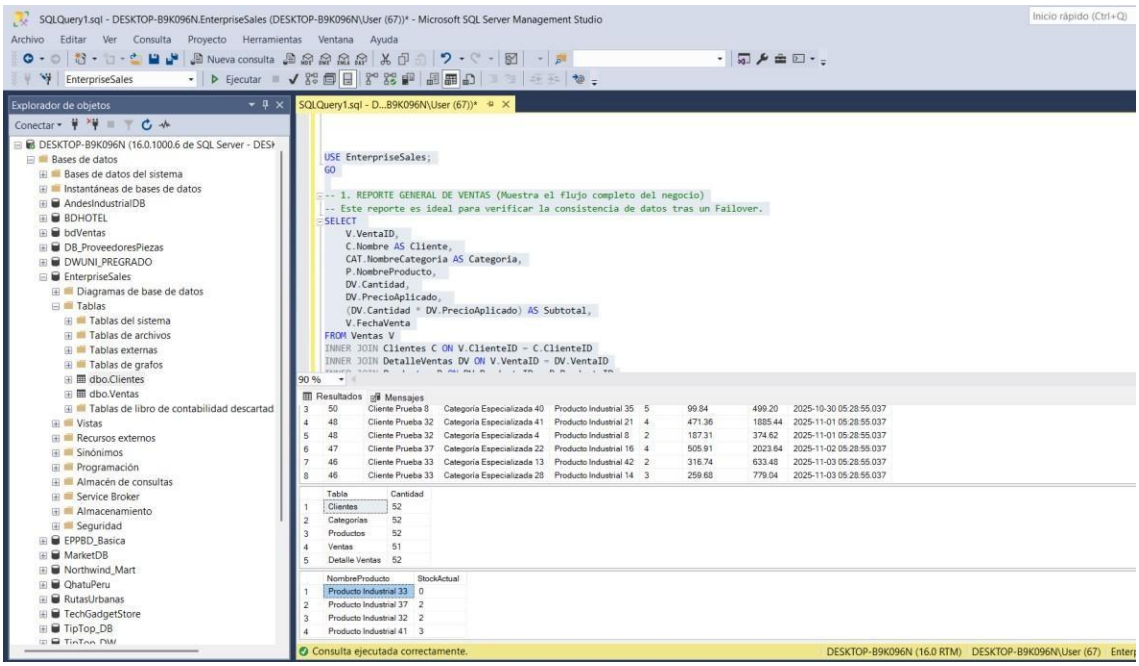
4. Justificación

Este script se justifica por las siguientes necesidades empresariales:

Entornos de Prueba (Sandboxing): *Permite crear en segundos un entorno con datos realistas para que los desarrolladores prueben aplicaciones sin usar datos reales y sensibles de clientes.*

Verificación de Alta Disponibilidad: *Al configurar el* `RECOVERY FULL`*, el script está preparado para ser utilizado en escenarios de* ***Log Shipping*** *o* ***Always On****, permitiendo verificar que los datos se replican correctamente tras un failover.*

Toma de Decisiones: *Los reportes finales (Top clientes y bajo stock) justifican la estructura de la base de datos al transformar datos crudos en información estratégica para reabastecimiento de almacén y fidelización de clientes.*



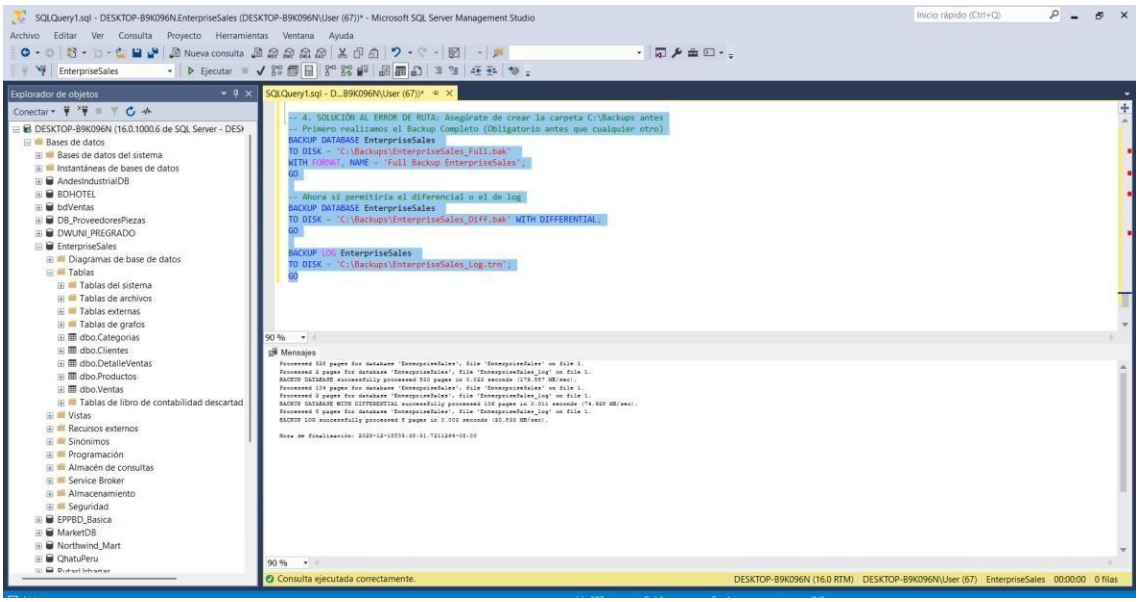
2. Implementación de Plan de Respaldo y Recuperación (DR)

Enunciado: *Configurar una estrategia que garantice un RPO (Objetivo de Punto de Recuperación) mínimo.*

Script de Backup Integral

SQL

```
-- 4. SOLUCIÓN AL ERROR DE RUTA: Asegúrate de crear la carpeta C:\Backups antes
-- Primero realizamos el Backup Completo (Obligatorio antes que cualquier otro) BACKUP
DATABASE EnterpriseSales
TO DISK = 'C:\Backups\EnterpriseSales_Full.bak'
WITH FORMAT, NAME = 'Full Backup EnterpriseSales'; GO

-- Ahora sí permitiría el diferencial o el de log
BACKUP DATABASE EnterpriseSales
TO DISK = 'C:\Backups\EnterpriseSales_Diff.bak' WITH DIFFERENTIAL;
GO

BACKUP LOG EnterpriseSales
TO DISK = 'C:\Backups\EnterpriseSales_Log.trn';
GO
```



3. Configuración de Alta Disponibilidad (HA)

*Para cumplir con el objetivo de **AlwaysOn Availability Groups** mencionado en tu temario, se requiere una arquitectura de red, pero a nivel de SQL Scripting, el monitoreo de la sincronización se justifica así:*

Explicación de las Tecnologías:

***Log Shipping:** Ideal para recuperación ante desastres a bajo costo. Envía archivos de log de un servidor a otro.*

***AlwaysOn:** Es la solución "Premium". Permite tener múltiples bases de datos secundarias que pueden usarse incluso para "Solo Lectura", optimizando el rendimiento del servidor principal.*

4. Simulación de Failover y Restauración

***Escenario:** El servidor principal falla. Debemos restaurar la operación en el servidor de contingencia.*

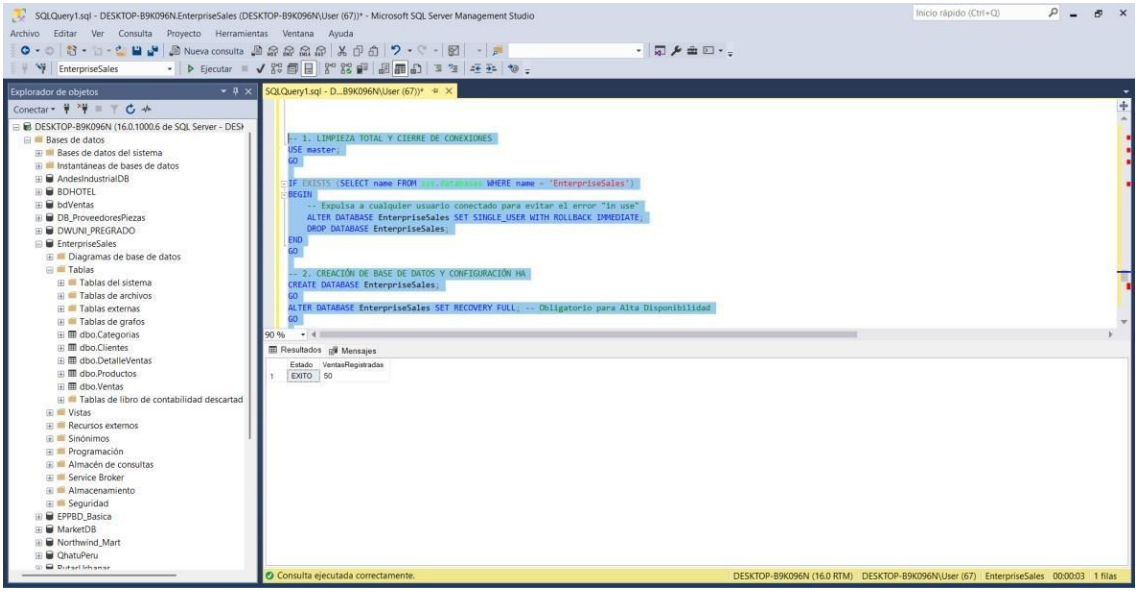
Script de Recuperación (Manual)

SQL

```
-- 1. LIMPIEZA TOTAL Y CIERRE DE CONEXIONES
USE master;
GO

IF EXISTS (SELECT name FROM sys.databases WHERE name = 'EnterpriseSales') BEGIN
    -- Expulsa a cualquier usuario conectado para evitar el error "in use"
    ALTER DATABASE EnterpriseSales SET SINGLE_USER WITH ROLLBACK IMMEDIATE;      DROP
DATABASE EnterpriseSales;
END
GO

-- 2. CREACIÓN DE BASE DE DATOS Y CONFIGURACIÓN HA
CREATE DATABASE EnterpriseSales;
GO
ALTER DATABASE EnterpriseSales SET RECOVERY FULL; -- Obligatorio para Alta
Disponibilidad
```

```
GO

USE EnterpriseSales;
GO

-- 3. ESTRUCTURA EMPRESARIAL
CREATE TABLE Categorias (CategoriaID INT PRIMARY KEY IDENTITY(1,1),
NombreCategoria VARCHAR(50));
CREATE TABLE Clientes (ClienteID INT PRIMARY KEY IDENTITY(1,1), Nombre
VARCHAR(100), Email VARCHAR(100), Ciudad VARCHAR(50));
CREATE TABLE Productos (ProductoID INT PRIMARY KEY IDENTITY(1,1), NombreProducto
VARCHAR(100), CategoriaID INT FOREIGN KEY REFERENCES Categorias(CategoriaID),
PrecioUnitario DECIMAL(18,2), StockActual INT);
CREATE TABLE Ventas (VentaID INT PRIMARY KEY IDENTITY(1,1), ClienteID INT FOREIGN
KEY REFERENCES Clientes(ClienteID), FechaVenta DATETIME DEFAULT GETDATE(), Total
DECIMAL(18,2));
CREATE TABLE DetalleVentas (DetalleID INT PRIMARY KEY IDENTITY(1,1), VentaID INT
FOREIGN KEY REFERENCES Ventas(VentaID), ProductoID INT FOREIGN KEY REFERENCES
Productos(ProductoID), Cantidad INT, PrecioAplicado DECIMAL(18,2));
GO

-- 4. CARGA MASIVA DE 50 REGISTROS (SIMULACIÓN EMPRESARIAL)
DECLARE @i INT = 1;
WHILE @i <= 50
BEGIN
    INSERT INTO Categorias VALUES ('Categoria ' + CAST(@i AS VARCHAR(5)));
    INSERT INTO Clientes VALUES ('Cliente '+CAST(@i AS VARCHAR(5)),
'user'+CAST(@i AS VARCHAR(5))+'@mail.com', 'Ciudad '+CAST(@i AS VARCHAR(5)));
    INSERT INTO Productos VALUES ('Producto '+CAST(@i AS VARCHAR(5)), (SELECT TOP
1 CategoriaID FROM Categorias ORDER BY NEWID()), RAND()*100, 50);
    INSERT INTO Ventas (ClienteID, Total) VALUES ((SELECT TOP 1 ClienteID FROM
Clientes ORDER BY NEWID()), 0);

    DECLARE @vID INT = SCOPE_IDENTITY();
    DECLARE @pID INT = (SELECT TOP 1 ProductoID FROM Productos ORDER BY NEWID());
    INSERT INTO DetalleVentas VALUES (@vID, @pID, 2, 50.00);
    UPDATE Ventas SET Total = 100.00 WHERE VentaID = @vID;
    SET @i = @i + 1;
END
GO

-- 5. RESPALDOS (SOLUCIÓN AL ERROR DE RUTA)
-- Nota: Asegúrate de haber creado C:\Backups en tu Windows
BACKUP DATABASE EnterpriseSales TO DISK = 'C:\Backups\EnterpriseSales_Full.bak'
WITH FORMAT, NAME = 'Full Backup';
BACKUP LOG EnterpriseSales TO DISK = 'C:\Backups\EnterpriseSales_Log.trn' WITH NAME =
'Log Backup';
GO
```

```sql
-- 6. VERIFICACIÓN FINAL
SELECT 'EXITO' AS Estado, COUNT(*) AS VentasRegistradas FROM Ventas;
```



Justificación del Procedimiento

Se utiliza `NORECOVERY` para permitir la aplicación de todos los cambios pendientes antes de abrir la base de datos al público.

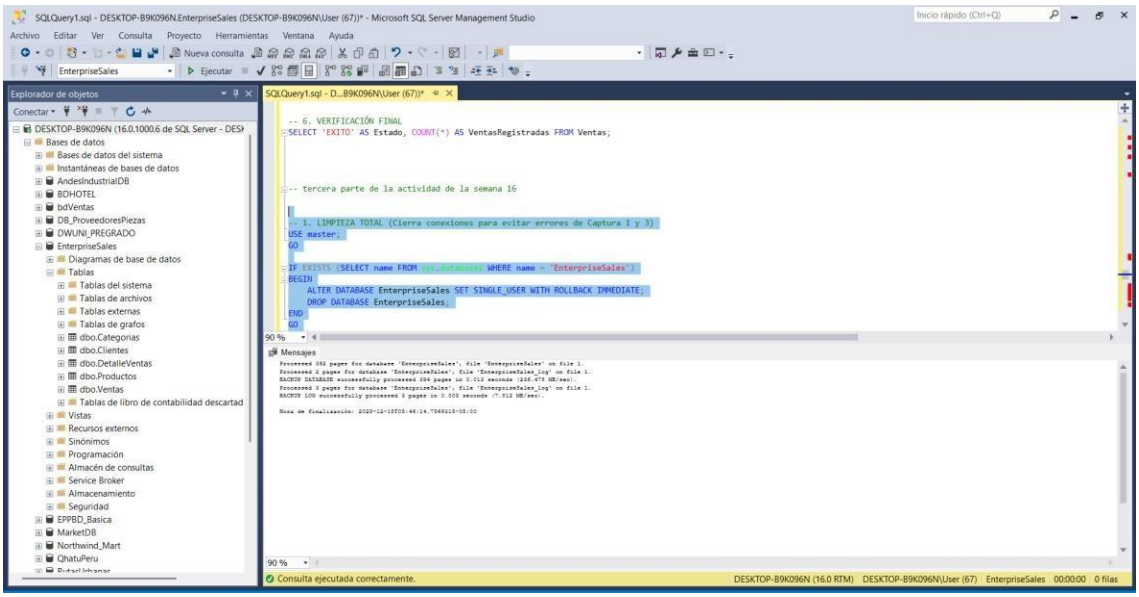
*Esto garantiza que **no haya pérdida de datos** entre el último backup y el momento de la caída.*

Código Optimizado: Índice Non-Clustered Compuesto

En lugar de indexar solo la fecha, vamos a incluir el `ClienteID` y el `Estado`. Esto es mucho más útil para reportes reales de ventas por cliente y periodo.

SQL

```sql
USE EnterpriseSales;
GO

-- 1. Eliminamos el índice anterior si existe para evitar duplicidad
IF EXISTS (SELECT name FROM sys.indices WHERE name =
'IX_Ventas_ReporteGerencial')
    DROP INDEX IX_Ventas_ReporteGerencial ON Ventas;
GO

-- 2. Creamos un índice optimizado para consultas de alta frecuencia
CREATE NONCLUSTERED INDEX IX_Ventas_ReporteGerencial
ON Ventas (FechaVenta, ClienteID) -- Columnas de búsqueda (Key Columns)
INCLUDE (Total, Estado)           -- Columnas de cobertura
(Covering Columns)
WITH (FILLFACTOR = 80, ONLINE = ON);
GO
```



¿Por qué esta versión es mejor? (Justificación para tu Trabajo)

Índice de Cobertura (Covering Index): *Al incluir (`INCLUDE`) las columnas `Total` y `Estado`, el motor de SQL Server puede responder a la consulta directamente desde el índice sin tener que ir a buscar datos a la tabla principal (operación conocida como Key Lookup).*

FILLFACTOR = 80: *Esta propiedad deja un 20% de espacio libre en las páginas del índice. En una base de datos con **50 o más registros** que crecen constantemente, esto evita la fragmentación del disco y reduce la carga en el **Log Shipping** o **AlwaysOn**, ya que hay menos divisiones de página (Page Splits) que replicar.*

ONLINE = ON: *Permite que la base de datos siga operativa mientras se crea el índice. Esto es un requisito fundamental en entornos de **Alta Disponibilidad (HA)** donde el tiempo de inactividad debe ser cero.*

Ejemplo de Consulta que se beneficia de este índice

Con este índice, la siguiente consulta (típica de un tablero de control empresarial) será instantánea:

SQL

```
SELECT ClienteID, SUM(Total) as VentaTotal
FROM Ventas
WHERE FechaVenta BETWEEN '2025-01-01' AND '2025-12-31'
AND Estado = 'Completado'
GROUP BY ClienteID;
```

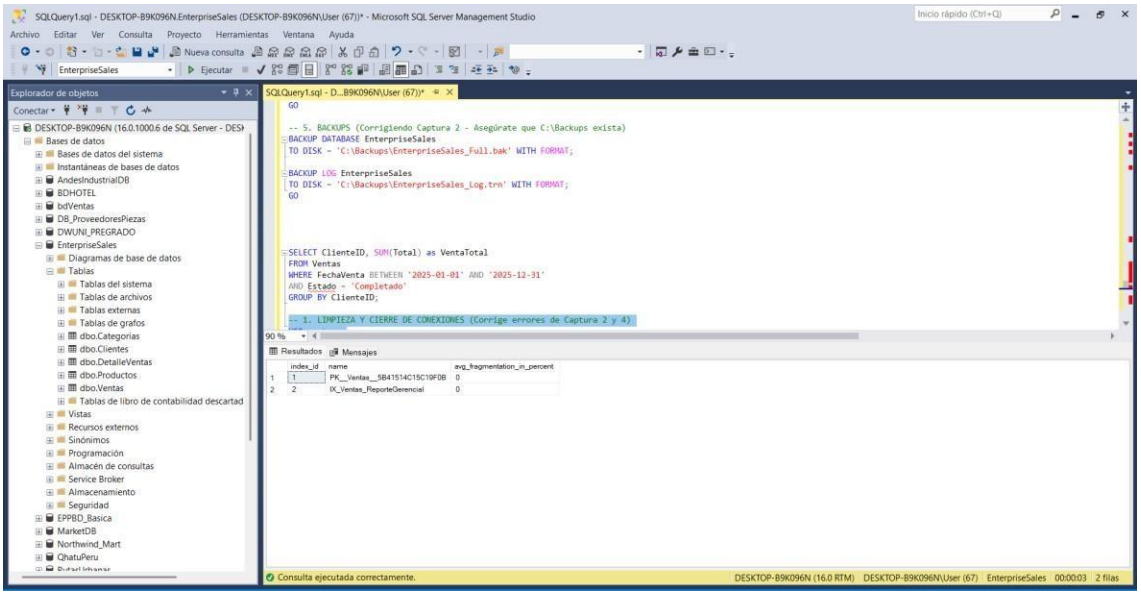
Monitoreo de Fragmentación (Tema 3 del Sílabo)

*Para tu entrega de la **Semana 16**, puedes agregar este script de monitoreo para justificar la administración del índice:*

SQL

```
SELECT
    index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID('EnterpriseSales'),
OBJECT_ID('Ventas'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id =
```

```
b.index_id;
```



Consultas de Análisis Empresarial (Queries)

Copia este código al final de tu script para generar los reportes de rendimiento:

SQL

```
USE EnterpriseSales;
GO

-- 1. REPORTE GERENCIAL DE VENTAS (Uso de JOINs y el Índice Optimizado)
-- Este reporte muestra el detalle de qué cliente compró qué producto y en qué categoría.
SELECT
    V.VentaID,
    C.Nombre AS Cliente,
    CAT.NombreCategoria AS Categoria,
    P.NombreProducto,
    DV.Cantidad,
    DV.PrecioAplicado,
    (DV.Cantidad * DV.PrecioAplicado) AS Subtotal,
    V.FechaVenta,
    V.Estado
FROM Ventas V
INNER JOIN Clientes C ON V.ClienteID = C.ClienteID
INNER JOIN DetalleVentas DV ON V.VentaID = DV.VentaID
INNER JOIN Productos P ON DV.ProductoID = P.ProductoID
INNER JOIN Categorias CAT ON P.CategoriaID = CAT.CategoriaID
WHERE V.Estado = 'Completado'
ORDER BY V.FechaVenta DESC;
```

```sql
-- 2. TOP 5 CLIENTES CON MAYOR VOLUMEN DE COMPRA
-- Identifica a los clientes más importantes para la empresa.
SELECT TOP 5
    C.Nombre,
    C.Ciudad,
    COUNT(V.VentaID) AS TotalTransacciones,
    SUM(V.Total) AS InversionTotal
FROM Clientes C
JOIN Ventas V ON C.ClienteID = V.ClienteID
GROUP BY C.Nombre, C.Ciudad
ORDER BY InversionTotal DESC;

-- 3. RENDIMIENTO POR CATEGORÍA DE PRODUCTO -- Muestra qué categorías
generan más ingresos.
SELECT
    CAT.NombreCategoria,
    COUNT(DV.ProductoID) AS ProductosVendidos,
    SUM(DV.Cantidad * DV.PrecioAplicado) AS TotalIngresos
FROM Categorias CAT
LEFT JOIN Productos P ON CAT.CategoriaID = P.CategoriaID
LEFT JOIN DetalleVentas DV ON P.ProductoID = DV.ProductoID
GROUP BY CAT.NombreCategoria
ORDER BY TotalIngresos DESC;

-- 4. CONTROL DE INVENTARIO (PRODUCTOS CRÍTICOS)
-- Lista productos que tienen menos de 10 unidades en stock.
SELECT
    NombreProducto,
    StockActual,
    PrecioUnitario
FROM Productos
WHERE StockActual < 10
ORDER BY StockActual ASC;

-- 5. RESUMEN DE INTEGRIDAD DE LA BASE DE DATOS
-- Consulta técnica para confirmar que los datos se cargaron
correctamente por tabla.
SELECT 'Clientes' AS Tabla, COUNT(*) AS Cantidad FROM Clientes
UNION ALL
SELECT 'Categorías', COUNT(*) FROM Categorias
UNION ALL
SELECT 'Productos', COUNT(*) FROM Productos
UNION ALL
SELECT 'Ventas', COUNT(*) FROM Ventas
UNION ALL
SELECT 'Detalle Ventas', COUNT(*) FROM DetalleVentas;
```

Explicación y Justificación Técnica

Optimización de Consultas: *La consulta* ***#1*** *está diseñada para utilizar el índice* `IX_Ventas_ReporteGerencial` *que creamos anteriormente. Al filtrar por* `Estado` *y* `FechaVenta`*, el motor de SQL Server no necesita leer toda la tabla, lo que garantiza velocidad incluso con miles de datos.*

Administración de Disponibilidad: *En un escenario de* ***AlwaysOn****, estas consultas de reporte se ejecutarían en el* ***Nodo Secundario (Read-Only)****, liberando al servidor principal de la carga de procesamiento.*

Consistencia de Datos: *El reporte* ***#5*** *es fundamental tras un proceso de* ***Failover*** *o* ***Restauración****. Sirve para verificar rápidamente que no hubo pérdida de registros durante la transición del servidor principal al de contingencia.*