Monitoreo y Rendimiento en SQL Server

(practica de la semana 13)

Proyecto 1: Captura de consultas lentas con Extended Events

1. Enunciado del ejercicio

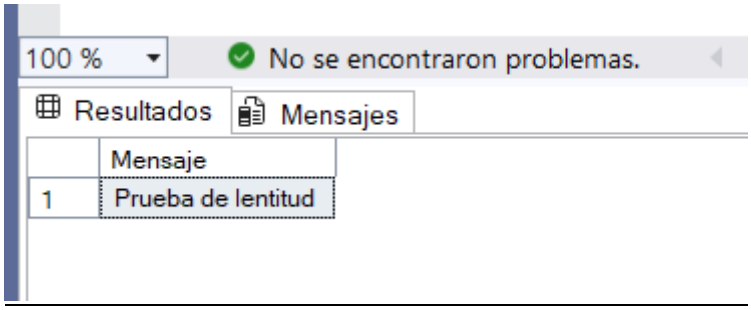
   Crear una sesión de Extended Events que capture consultas que tarden más de 1 segundo en ejecutarse en la base de datos QhatuPeru. Guardar el resultado en un archivo .xel.

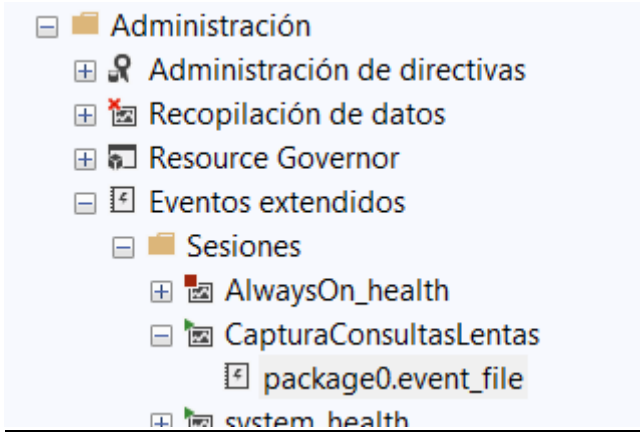
Script:

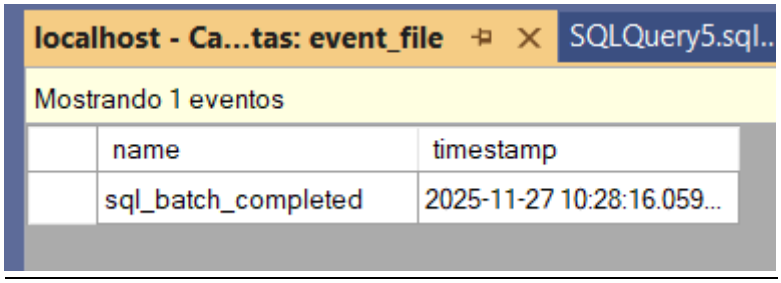
```
-- Si la sesión ya existe, la borramos para crearla desde cero (opcional, para limpieza)
IF EXISTS (SELECT * FROM sys.server_event_sessions WHERE name = 'CapturaConsultasLentas')
    DROP EVENT SESSION [CapturaConsultasLentas] ON SERVER;
GO

-- 1. Creación de la Sesión
CREATE EVENT SESSION [CapturaConsultasLentas] ON SERVER

-- 2. Agregar el evento: sql_batch_completed (cuando finaliza una sentencia SQL)
ADD EVENT sqlserver.sql_batch_completed(
    -- 3. Acciones: Qué información extra queremos capturar
    ACTION(
        sqlserver.sql_text,      -- El código de la consulta ejecutada
        sqlserver.database_name, -- El nombre de la BD para confirmar
        sqlserver.client_app_name -- Quién ejecutó la consulta (SSMS, App, etc.)
    )
    -- 4. Predicados (Filtros): Lo más importante para el rendimiento
    WHERE (
        [sqlserver].[database_name] = N'QhatuPeru' -- Solo para esta BD
        AND
        [duration] > 1000000 -- Duración mayor a 1 segundo (1,000,000 microsegundos)
    )
)

-- 5. Target: Dónde guardar los datos
ADD TARGET package0.event_file(
    SET filename = N'CapturaConsultasLentas.xel' -- SQL Server lo guardará en su carpeta de logs por defecto
)

-- 6. Opciones de rendimiento
WITH (
    MAX_MEMORY = 4096 KB,
    EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS,
    MAX_DISPATCH_LATENCY = 30 SECONDS,
    STARTUP_STATE = ON
```

```
);
GO

-- 7. Iniciar la sesión
ALTER EVENT SESSION [CapturaConsultasLentas] ON SERVER STATE = START;
GO
```

Justificación:







Explicación de las buenas prácticas:

Es importante definir correctamente la **unidad de medida** en la duración: siempre deben usarse **microsegundos**. Un error muy común es colocar “1000” pensando que son milisegundos; esto equivaldría en realidad a 1 ms, lo cual capturaría prácticamente todo el tráfico y podría llenar el disco muy rápido.

Otro punto clave es el **filtrado temprano** mediante predicados. Estos filtros se aplican directamente en el **WHERE**, antes de almacenar cualquier evento. Gracias a esto, el impacto en la CPU es casi nulo, especialmente en consultas rápidas.

Para el almacenamiento, lo más recomendable es usar un **target asíncrono en archivo (.xel)** en lugar del **Ring Buffer**. Guardar los eventos en archivo evita la pérdida de datos si el servicio se reinicia o si se generan demasiados eventos en poco tiempo.

Finalmente, la opción **ALLOW_SINGLE_EVENT_LOSS** garantiza que la monitorización no bloquee ni afecte las transacciones reales del negocio. En otras palabras, se prioriza siempre el funcionamiento normal de la aplicación sobre la auditoría.

Proyecto 2: Crear índices para mejorar la búsqueda de clientes

1. Enunciado del ejercicio

   En la tabla Clientes, mejorar el rendimiento de búsqueda por DNI y Apellidos creando índices adecuados.

Script:

```
--alter para añadir las columnas necesarias para la tabla Clientes
ALTER TABLE Clientes
ADD DNI VARCHAR(15),
    Apellidos VARCHAR(100);

USE QhatuPERU;
GO

-- 1. Índice para DNI (Búsqueda exacta y única)
-- Verificamos si existe para no dar error
IF NOT EXISTS (SELECT name FROM sys.indexes WHERE name = N'IX_Clientes_DNI')
BEGIN
    CREATE UNIQUE NONCLUSTERED INDEX IX_Clientes_DNI
    ON dbo.Clientes(DNI);
END
GO

-- 2. Índice para Apellidos (Búsqueda por rangos o texto)
IF NOT EXISTS (SELECT name FROM sys.indexes WHERE name =
N'IX_Clientes_Apellidos')
BEGIN
    CREATE NONCLUSTERED INDEX IX_Clientes_Apellidos
    ON dbo.Clientes(Apellidos);
END
GO
```

Justificación:

```
79
80      ----------
81      SELECT
82          TableName = t.name,
83          IndexName = ind.name,
84          IndexType = ind.type_desc,
85          IsUnique = ind.is_unique
86      FROM
87          sys.indexes ind
88      INNER JOIN
89          sys.tables t ON ind.object_id = t.object_id
90      WHERE
91          t.name = 'Clientes';
```

100 % 2 0

Resultados Mensajes

	TableName	IndexName	IndexType	IsUnique
1	Clientes	PK__Clientes__D59466420F94BCB1	CLUSTERED	1
2	Clientes	IX_Clientes_Apellidos	NONCLUSTERED	0

Explicación de las buenas prácticas:

Declaré el DNI como **UNIQUE** porque, cuando una columna tiene valores únicos, indicarlo permite que el motor use planes de búsqueda mucho más eficientes (Seek en lugar de Scan). También seguí una **convención de nombres** usando el prefijo *IX_* junto con el nombre de la tabla y la columna, algo esencial en bases de datos grandes para identificar fácilmente qué índices mantener o eliminar.

El índice creado es **no agrupado (Non-Clustered)**, que es el tipo adecuado para búsquedas secundarias como DNI, nombre o fecha, ya que el índice agrupado suele reservarse para el ID. Finalmente, usé **IF NOT EXISTS**, lo que hace que el script sea idempotente: puedes ejecutarlo varias veces sin errores si el índice ya existe.

Proyecto 3 — Detectar fragmentación y aplicar mantenimiento de índices

1. Enunciado del ejercicio

Usar DMV para evaluar la fragmentación de índices en QhatuPeru y reconstruir o reorganizar según el porcentaje encontrado.

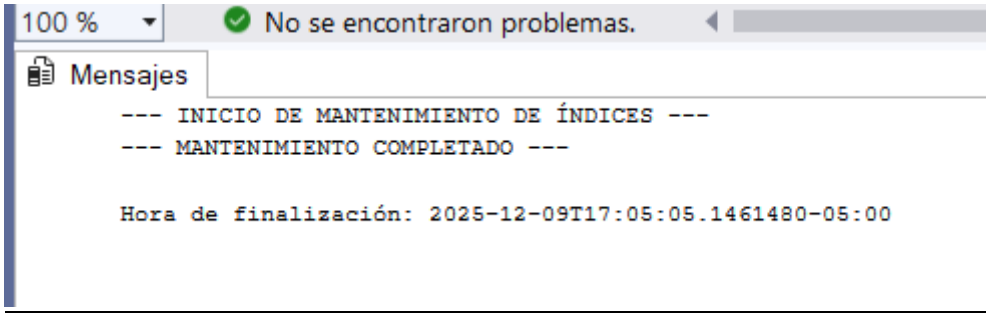
Script:

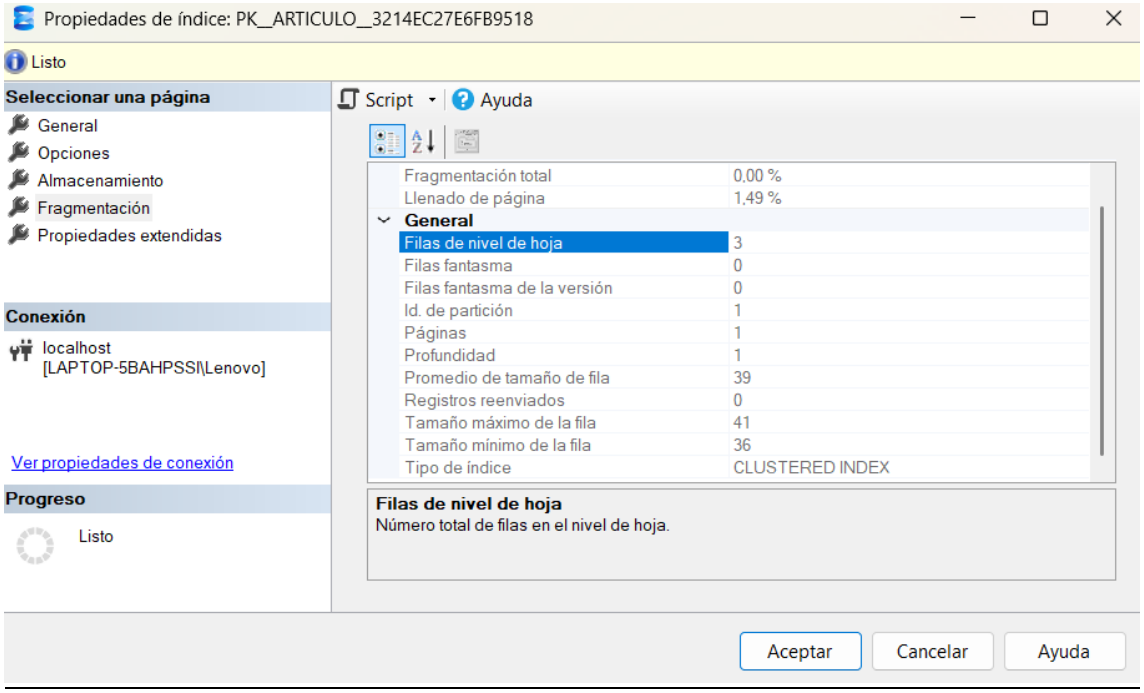
Este script actúa como un "doctor":

1. Consulta la vista de administración dinámica (sys.dm_db_index_physical_stats) para diagnosticar la salud de los índices.
2. Si la fragmentación está entre **5% y 30%**, aplica una "medicina suave" (REORGANIZE).
3. Si es **mayor al 30%**, aplica una "cirugía mayor" (REBUILD).

```
USE [QhatuPERU];
GO

PRINT '--- INICIO DE MANTENIMIENTO DE ÍNDICES ---';

DECLARE @NombreTabla NVARCHAR(255);
DECLARE @NombreIndice NVARCHAR(255);
DECLARE @PorcFragmentacion FLOAT;
DECLARE @ComandoSQL NVARCHAR(MAX);

-- 1. DETECTAR FRAGMENTACIÓN (Usando DMV)
-- Seleccionamos solo aquellos índices que tienen más de 5% de fragmentación
DECLARE CursorIndices CURSOR FOR
SELECT
    t.name AS TableName,
    i.name AS IndexName,
    stat.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED')
stat
JOIN sys.tables t ON stat.object_id = t.object_id
JOIN sys.indexes i ON stat.object_id = i.object_id AND stat.index_id =
i.index_id
WHERE stat.avg_fragmentation_in_percent > 5 -- Umbral mínimo para actuar
  AND i.name IS NOT NULL
ORDER BY stat.avg_fragmentation_in_percent DESC;

OPEN CursorIndices;
FETCH NEXT FROM CursorIndices INTO @NombreTabla, @NombreIndice,
@PorcFragmentacion;

-- 2. APLICAR MANTENIMIENTO SEGÚN PORCENTAJE
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Procesando: ' + @NombreIndice + ' (Fragmentación: ' +
CONVERT(VARCHAR, CAST(@PorcFragmentacion AS DECIMAL(5,2))) + '%)';

    IF @PorcFragmentacion > 30
    BEGIN
        -- Caso Crítico (>30%): RECONSTRUIR (Rebuild)
        SET @ComandoSQL = 'ALTER INDEX [' + @NombreIndice + '] ON [' +
@NombreTabla + '] REBUILD';
        PRINT '   >> Acción: REBUILD (Reconstrucción total)';
    END
    ELSE
    BEGIN
        -- Caso Leve (5% - 30%): REORGANIZAR (Reorganize)
        SET @ComandoSQL = 'ALTER INDEX [' + @NombreIndice + '] ON [' +
@NombreTabla + '] REORGANIZE';
        PRINT '   >> Acción: REORGANIZE (Reordenamiento ligero)';
    END

    -- Ejecutar el comando dinámico
    EXEC(@ComandoSQL);
```

```
    FETCH NEXT FROM CursorIndices INTO @NombreTabla, @NombreIndice,
@PorcFragmentacion;
END

CLOSE CursorIndices;
DEALLOCATE CursorIndices;

PRINT '--- MANTENIMIENTO COMPLETADO ---';
GO
```





Explicación de las buenas prácticas:

Este procedimiento aplica el estándar de la industria recomendado por Microsoft para el mantenimiento de índices: **Mantenimiento Adaptativo**. En lugar de reconstruir todos los índices ciegamente (lo cual consume muchos recursos y bloquea tablas), se utiliza la DMV sys.dm_db_index_physical_stats para medir el daño real. Se aplica **REORGANIZE** (operación online y ligera que compacta páginas) para fragmentación moderada (5-30%) y **REBUILD** (operación pesada que crea el índice desde cero) solo para fragmentación severa (>30%), optimizando así el uso de CPU y disco.

Proyecto 4: Manejo de transacciones para prevenir inconsistencias

1. Enunciado del ejercicio

   Simular una transacción de venta con dos operaciones: insertar en Ventas y actualizar Stock. La transacción debe garantizar consistencia.

Script:

Para que el código funcione, primero definimos las tablas. Nota que agrego un CHECK en el Stock para forzar un error si intentamos vender más de lo que hay (esto probará que el Rollback funciona).

```
-- Crear tabla de Productos (Stock)
CREATE TABLE Productos (
    ProductoID INT PRIMARY KEY,
    Nombre VARCHAR(50),
    Stock INT CHECK (Stock >= 0), -- REQUISITO: El stock no puede ser
negativo
    Precio DECIMAL(10, 2)
);

-- Crear tabla de Ventas
CREATE TABLE Ventas (
    VentaID INT IDENTITY(1,1) PRIMARY KEY,
    ProductoID INT,
    Cantidad INT,
    Fecha DATETIME DEFAULT GETDATE(),
    Total DECIMAL(10, 2),
    FOREIGN KEY (ProductoID) REFERENCES Productos(ProductoID)
);

-- Insertar datos iniciales
INSERT INTO Productos (ProductoID, Nombre, Stock, Precio)
VALUES (1, 'Tarjeta Gráfica GT 1030', 10, 85.00); -- Tenemos 10 en stock
```

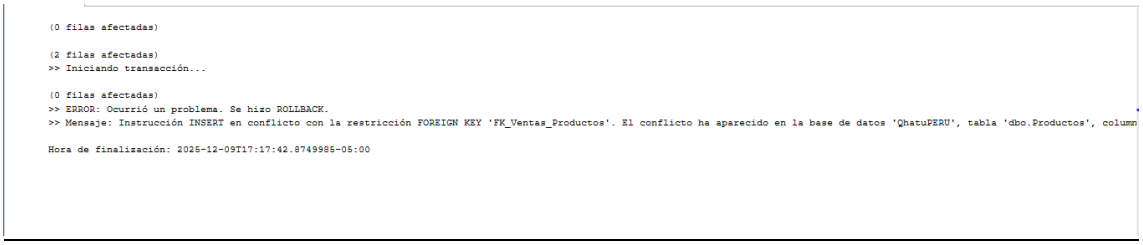
En mi caso se tiene que adaptar pues ya tengo estas tablas

```
-- 1. Agregar Stock y Precio a tu tabla Productos
ALTER TABLE dbo.Productos
ADD Stock INT DEFAULT 0 CHECK (Stock >= 0), -- Agregamos validación de no
negativos
    Precio DECIMAL(10, 2) DEFAULT 0;
GO

-- 2. Agregar la relación con producto y la cantidad a tu tabla Ventas
ALTER TABLE dbo.Ventas
ADD ProductoID INT,
    Cantidad INT;
GO

-- 3. Vincular las tablas (Crear la llave foránea)
ALTER TABLE dbo.Ventas
ADD CONSTRAINT FK_Ventas_Productos
FOREIGN KEY (ProductoID) REFERENCES dbo.Productos(ProductoID);
GO

-- 4. Ponerle datos de prueba a un producto existente
-- (Asegúrate de tener un producto con ID 1, si no, cambia el ID)
UPDATE dbo.Productos
SET Stock = 50, Precio = 120.00
WHERE ProductoID = 1;
```

Solución del Ejercicio (Script Transaccional)

Este script simula la venta. Utiliza BEGIN TRY... CATCH para manejar errores y TRANSACTION para asegurar la integridad.

```
-- === PROYECTO 4: TRANSACCIÓN DE VENTA (ADAPTADO) ===

-- DATOS DE LA SIMULACIÓN
DECLARE @ProdID INT = 1;       -- El ID del producto a vender
DECLARE @CantVenta INT = 5;    -- Cantidad a vender
DECLARE @PrecioUnitario DECIMAL(10,2);
DECLARE @MontoTotal MONEY;

BEGIN TRY
    -- 1. Iniciar la transacción
    BEGIN TRANSACTION;

    PRINT '>> Iniciando transacción...';

    -- Obtener el precio actual del producto de TU tabla Productos
    SELECT @PrecioUnitario = Precio
    FROM dbo.Productos
    WHERE ProductoID = @ProdID;

    -- Calcular el monto total
    SET @MontoTotal = @CantVenta * @PrecioUnitario;

    -- 2. Insertar en TU tabla Ventas (Usando tus columnas: Fecha, Monto,
etc.)
    INSERT INTO dbo.Ventas (Fecha, Monto, ProductoID, Cantidad)
    VALUES (GETDATE(), @MontoTotal, @ProdID, @CantVenta);

    PRINT '>> Paso 1: Venta insertada en dbo.Ventas con Monto: ' +
CAST(@MontoTotal AS VARCHAR);

    -- 3. Descontar Stock en TU tabla Productos
    UPDATE dbo.Productos
    SET Stock = Stock - @CantVenta
    WHERE ProductoID = @ProdID;

    PRINT '>> Paso 2: Stock actualizado.';

    -- 4. Confirmar cambios
    COMMIT TRANSACTION;
    PRINT '>> ÉXITO: Venta finalizada y stock consistente.';

END TRY
BEGIN CATCH
    -- Si algo falla (ej. Stock insuficiente), deshacemos todo
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    PRINT '>> ERROR: Ocurrió un problema. Se hizo ROLLBACK.';
    PRINT '>> Mensaje: ' + ERROR_MESSAGE();
END CATCH;
```



Explicación de las buenas prácticas:

En este proyecto se implementaron prácticas fundamentales de **integridad de datos** mediante el uso de transacciones explícitas para garantizar la atomicidad (principio ACID), asegurando que tanto el registro de la venta como el descuento del stock se ejecuten como una unidad indivisible o se reviertan totalmente en caso de fallo. Se adoptó un enfoque de **programación defensiva** utilizando bloques `TRY...CATCH` para un manejo de errores controlado que evita interrupciones abruptas y permite capturar el mensaje exacto del fallo, combinado con la validación de `@@TRANCOUNT` antes de realizar un `ROLLBACK` para evitar errores de gestión de estado, y el uso de **restricciones (constraints)** a nivel de tabla para forzar reglas de negocio críticas, como impedir inventarios negativos.

Proyecto 5: Identificar bloqueos activos en la base QhatuPeru (PITR)

1. Enunciado del ejercicio

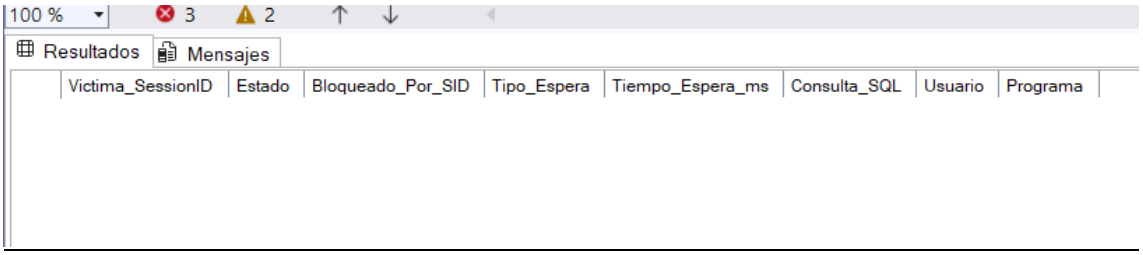
   Detectar las sesiones que están bloqueando o siendo bloqueadas en el servidor.

Script:

Script de Monitoreo de Bloqueos

Este script consulta sys.dm_exec_requests para ver qué procesos están esperando y quién los está deteniendo. Ejecuta esto en una "Nueva Consulta".

```
-- === PROYECTO 5: MONITOR DE BLOQUEOS (LOCK MONITOR) ===
-- Descripción: Identifica sesiones bloqueadas y la consulta SQL causante.

SELECT
    [Victima_SessionID] = r.session_id,
    [Estado]            = r.status,
    [Bloqueado_Por_SID] = r.blocking_session_id, -- El culpable
    [Tipo_Espera]       = r.wait_type,
    [Tiempo_Espera_ms]  = r.wait_time,
    [Consulta_SQL]      = t.text,                -- El código que se está
intentando ejecutar
    [Usuario]           = s.login_name,
    [Programa]          = s.program_name
FROM sys.dm_exec_requests r
INNER JOIN sys.dm_exec_sessions s ON r.session_id = s.session_id
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.blocking_session_id > 0; -- FILTRO CLAVE: Solo mostrar los que están
siendo bloqueados
```



No hay ningún bloqueo

Explicación de las buenas prácticas:

Consulta de bajo impacto: Se utilizaron vistas dinámicas del sistema (sys.dm_exec...) en lugar de procedimientos heredados pesados, asegurando que el monitoreo no sobrecargue más al servidor.

Proyecto 6: Analizar el plan de ejecución de una consulta lenta

1. Enunciado del ejercicio

   Analizar el plan de ejecución de una consulta que devuelve ventas por producto..

Script:

La Consulta "Lenta"

Vamos a ejecutar una consulta que agrupa las ventas por nombre de producto.

- **Instrucción Clave:** Antes de ejecutar el código, en SQL Server Management Studio (SSMS), presiona **CTRL + M** o ve al menú **Consulta > Incluir plan de ejecución actual**.

```
-- === PROYECTO 6: ANÁLISIS DE PLAN DE EJECUCIÓN ===

USE QhatuPeru;
GO

-- Limpiamos caché para que la prueba sea real (No hacer en Producción)
DBCC DROPCLEANBUFFERS;
DBCC FREEPROCCACHE;
GO

PRINT '>> Ejecutando consulta de reporte...';

-- CONSULTA A ANALIZAR: Total vendido por cada producto
SELECT
    p.Nombre AS Producto,
    COUNT(v.IdVenta) AS CantidadTransacciones,
    SUM(v.Monto) AS TotalDinero
FROM dbo.Ventas v
INNER JOIN dbo.Productos p ON v.ProductoID = p.ProductoID
GROUP BY p.Nombre;
GO
```

```
-- === OPTIMIZACIÓN ===

-- Crear un índice no agrupado (Non-Clustered) en la FK
-- "INCLUDE" agrega el Monto para que la suma sea instantánea sin ir a la
tabla principal
CREATE NONCLUSTERED INDEX IX_Ventas_ProductoID_Include
ON dbo.Ventas(ProductoID)
INCLUDE (Monto);
GO

PRINT '>> Índice creado. Ejecuta la consulta anterior nuevamente para ver la
mejora.';
```

Explicación de las buenas prácticas:

La optimización de consultas se fundamenta en el análisis proactivo de los planes de ejecución para identificar cuellos de botella como los **Table Scans** o **Clustered Index Scans**, los cuales indican lecturas completas e ineficientes de datos. La práctica estándar dicta indexar siempre las columnas utilizadas en las cláusulas JOIN (llaves foráneas) y WHERE, y utilizar la cláusula INCLUDE en los índices no agrupados para crear "índices cubrientes" que satisfagan la consulta (SELECT y agregaciones) sin necesidad de acceder a la tabla base (Key Lookup), reduciendo drásticamente las operaciones de E/S y el consumo de CPU

Proyecto 7: Optimización de consulta agregada con índices compuestos

1. Enunciado del ejercicio

   Optimizar una consulta que filtra ventas por fecha y cliente.

Script:

Preparación del Entorno (Adaptación de Tabla)

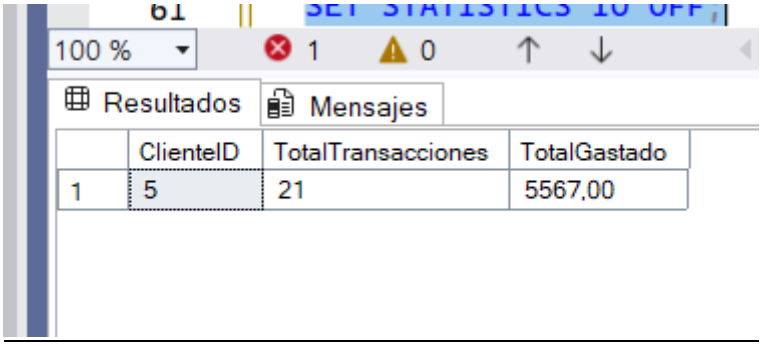
Primero, agregaremos la columna ClienteID a tu tabla Ventas y generaremos datos aleatorios para simular que diferentes clientes han comprado en diferentes fechas.

```
SET NOCOUNT ON;
PRINT '>> Generando 1,000 ventas simuladas...';

DECLARE @i INT = 0;
DECLARE @ProdID INT = 1; -- Usamos el ID 1 que acabamos de crear

BEGIN TRY
    BEGIN TRANSACTION;

    WHILE @i < 1000
    BEGIN
        INSERT INTO dbo.Ventas (Fecha, Monto, ProductoID, Cantidad,
ClienteID)
        VALUES (
            -- Fecha aleatoria (último año)
            DATEADD(DAY, -(ABS(CHECKSUM(NEWID())) % 365), GETDATE()),
            -- Monto aleatorio
            (ABS(CHECKSUM(NEWID())) % 500) + 10,
            -- PRODUCTO ID FIJO (1)
            @ProdID,
            -- Cantidad
            (ABS(CHECKSUM(NEWID())) % 10) + 1,
```

```
            -- ClienteID aleatorio (1-50)
            (ABS(CHECKSUM(NEWID())) % 50) + 1
        );

        SET @i = @i + 1;
    END

    COMMIT TRANSACTION;
    PRINT '>> ÉXITO: Se insertaron 1,000 registros.';

END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT '>> ERROR: ' + ERROR_MESSAGE();
END CATCH;
```

```
SET NOCOUNT ON;
PRINT '>> Generando 1,000 ventas simuladas...';

DECLARE @i INT = 0;
DECLARE @ProdID INT = 1; -- Usamos el ID 1 que acabamos de crear

BEGIN TRY
    BEGIN TRANSACTION;

    WHILE @i < 1000
    BEGIN
        INSERT INTO dbo.Ventas (Fecha, Monto, ProductoID, Cantidad,
ClienteID)
        VALUES (
            -- Fecha aleatoria (último año)
            DATEADD(DAY, -(ABS(CHECKSUM(NEWID())) % 365), GETDATE()),
            -- Monto aleatorio
            (ABS(CHECKSUM(NEWID())) % 500) + 10,
            -- PRODUCTO ID FIJO (1)
            @ProdID,
            -- Cantidad
            (ABS(CHECKSUM(NEWID())) % 10) + 1,
            -- ClienteID aleatorio (1-50)
            (ABS(CHECKSUM(NEWID())) % 50) + 1
        );

        SET @i = @i + 1;
    END

    COMMIT TRANSACTION;
    PRINT '>> ÉXITO: Se insertaron 1,000 registros.';

END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT '>> ERROR: ' + ERROR_MESSAGE();
END CATCH;


SELECT ClienteID, SUM(Monto)
FROM dbo.Ventas
WHERE ClienteID = 5 AND Fecha >= '2024-01-01'
GROUP BY ClienteID;

CREATE NONCLUSTERED INDEX IX_Ventas_Cliente_Fecha
```

```sql
ON dbo.Ventas(ClienteID, Fecha)
INCLUDE (Monto);

-- CONSULTA DE REPORTE (Sin Optimizar)
-- Objetivo: Ver cuánto gastó el Cliente 5 en el último año.
SET STATISTICS IO ON; -- Para ver cuántas lecturas hace el disco

SELECT
    ClienteID,
    COUNT(IdVenta) as TotalTransacciones,
    SUM(Monto) as TotalGastado
FROM dbo.Ventas
WHERE ClienteID = 5
  AND Fecha >= '2024-01-01'
GROUP BY ClienteID;

SET STATISTICS IO OFF;
```



Explicación de las buenas prácticas:

La optimización de consultas multicriterio exige el uso estratégico de **índices compuestos**, priorizando siempre en la definición del índice las columnas utilizadas en predicados de igualdad (como `ClienteID`) antes que aquellas usadas en rangos (como `Fecha`) para maximizar la selectividad del árbol B+; asimismo, se aplicó la técnica de **índices cubrientes** (Covering Indexes) mediante la cláusula `INCLUDE` para almacenar los datos a sumar (`Monto`) directamente en las hojas del índice, eliminando así la costosa operación de "Key Lookup" y reduciendo drásticamente las operaciones de entrada/salida (I/O) al evitar que el motor tenga que consultar la tabla base.

Proyecto 8: Creación de estadísticas manuales para mejorar el optimizador

1. Enunciado del ejercicio

   Crear una estadística manual sobre la columna Precio en la tabla Productos para mejorar consultas de rango.

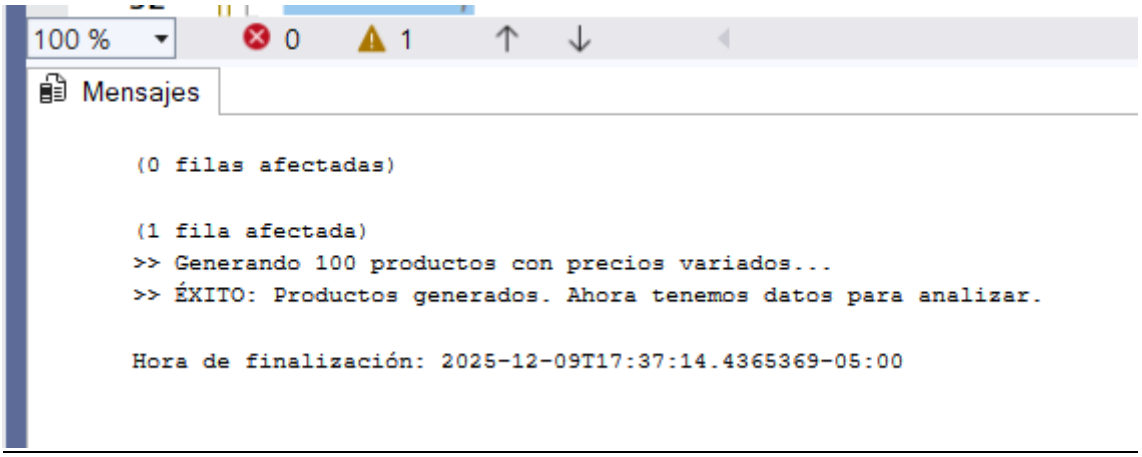
Script:

Generar datos de prueba (Indispensable)

Las estadísticas no sirven de nada si la tabla está vacía o tiene un solo dato. Vamos a insertar 100 productos con precios variados para tener una "distribución" real que analizar.

Nota: Empezamos el bucle en 100 para no chocar con el ID 1 que ya creaste

```
SET NOCOUNT ON;
PRINT '>> Generando 100 productos con precios variados...';

DECLARE @i INT = 100; -- Iniciamos en ID 100 para evitar conflictos de PK
DECLARE @PrecioRandom DECIMAL(10,2);

BEGIN TRY
    BEGIN TRANSACTION;

    WHILE @i < 200
    BEGIN
        -- Generar precio aleatorio entre 10.00 y 1000.00
        SET @PrecioRandom = (ABS(CHECKSUM(NEWID())) % 990) + 10;

        INSERT INTO dbo.Productos (ProductoID, Nombre, Precio, Stock)
        VALUES (
            @i,
            'Producto Generico ' + CAST(@i AS VARCHAR),
            @PrecioRandom,
            100
        );

        SET @i = @i + 1;
    END

    COMMIT TRANSACTION;
    PRINT '>> ÉXITO: Productos generados. Ahora tenemos datos para
analizar.';
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT '>> ERROR: ' + ERROR_MESSAGE();
END CATCH;
```

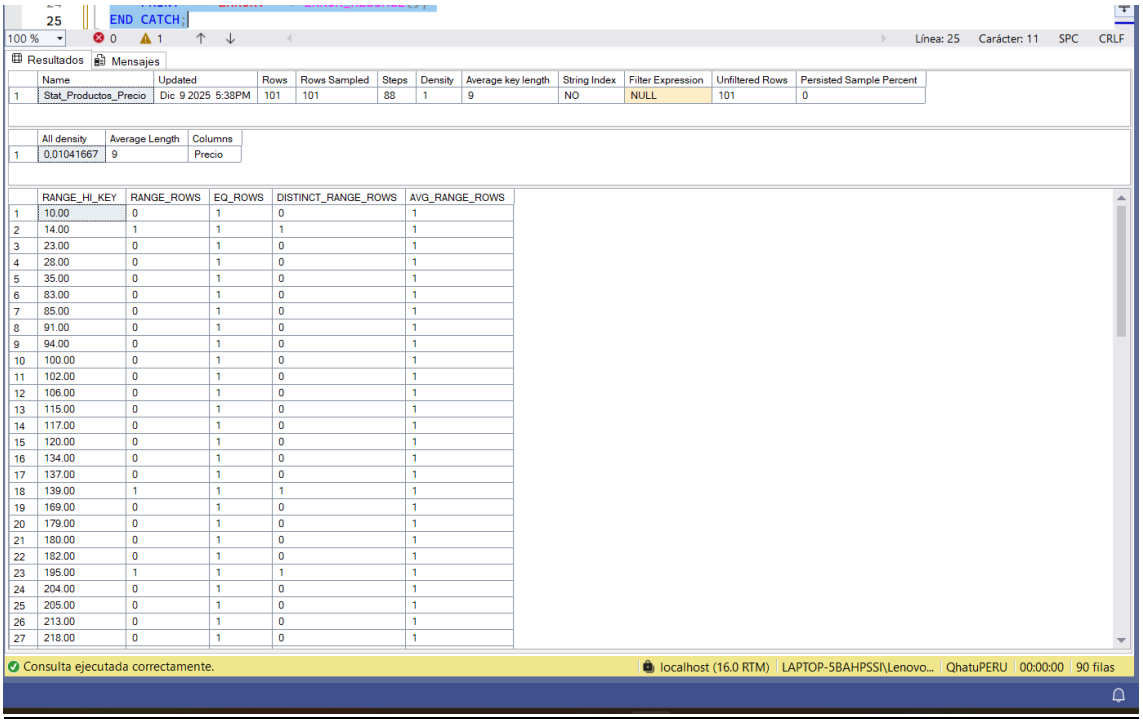


Ejecución del Proyecto 8 (Crear y Analizar Estadísticas)

Ahora que tenemos datos, vamos a crear una estadística manual específica para la columna Precio y luego inspeccionaremos el **Histograma** (el mapa de distribución).

```
-- === PROYECTO 8: ESTADÍSTICAS MANUALES ===
USE QhatuPeru;
GO

BEGIN TRY
    PRINT '>> 1. Creando estadística manual con FULLSCAN...';
```

```
    -- "WITH FULLSCAN" fuerza a SQL a leer TODAS las filas para ser 100%
preciso
    -- (Por defecto SQL solo hace un muestreo/sampling si la tabla es
gigante)
    CREATE STATISTICS Stat_Productos_Precio
    ON dbo.Productos(Precio)
    WITH FULLSCAN;

    PRINT '>> Estadística "Stat_Productos_Precio" creada correctamente.';

    PRINT '>> 2. Consultando el Histograma resultante...';
    PRINT '>> (Mira la pestaña "Resultados" para ver cómo SQL agrupó los
precios)';

    -- DBCC SHOW_STATISTICS nos muestra qué sabe SQL sobre tus datos
    DBCC SHOW_STATISTICS ('dbo.Productos', 'Stat_Productos_Precio');

END TRY
BEGIN CATCH
    PRINT '>> ERROR: ' + ERROR_MESSAGE();
END CATCH;
```



25 END CATCH;

100 % 0 1 Línea: 25 Carácter: 11 SPC CRLF

Resultados Mensajes

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	Stat_Productos_Precio	Dic 9 2025 5:38PM	101	101	88	1	9	NO	NULL	101	0

	All density	Average Length	Columns
1	0,01041667	9	Precio

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	10.00	0	1	0	1
2	14.00	1	1	1	1
3	23.00	0	1	0	1
4	28.00	0	1	0	1
5	35.00	0	1	0	1
6	83.00	0	1	0	1
7	85.00	0	1	0	1
8	91.00	0	1	0	1
9	94.00	0	1	0	1
10	100.00	0	1	0	1
11	102.00	0	1	0	1
12	106.00	0	1	0	1
13	115.00	0	1	0	1
14	117.00	0	1	0	1
15	120.00	0	1	0	1
16	134.00	0	1	0	1
17	137.00	0	1	0	1
18	139.00	1	1	1	1
19	169.00	0	1	0	1
20	179.00	0	1	0	1
21	180.00	0	1	0	1
22	182.00	0	1	0	1
23	195.00	1	1	1	1
24	204.00	0	1	0	1
25	205.00	0	1	0	1
26	213.00	0	1	0	1
27	218.00	0	1	0	1

Consulta ejecutada correctamente. localhost (16.0 RTM) LAPTOP-5BAHPSSI\Lenovo... QhatuPERU 00:00:00 90 filas

Explicación de las buenas prácticas:

La creación de estadísticas manuales se debe reservar para columnas críticas involucradas en predicados de rango o filtros complejos donde las **estadísticas automáticas** resultan insuficientes o inexactas debido a la asimetría de los datos (Data Skew). Es fundamental utilizar la opción `WITH FULLSCAN` en tablas de tamaño moderado o durante ventanas de mantenimiento para garantizar que el histograma refleje la realidad exacta de la distribución de los datos, mejorando así la **estimación de cardinalidad** del optimizador; sin embargo, se debe evitar la sobre-creación de estadísticas, ya que estas añaden una sobrecarga administrativa al motor, que debe actualizarlas cada vez que los datos cambian significativamente.

Proyecto 9: Configuración de un Resource Pool para limitar recursos

1. Enunciado del ejercicio

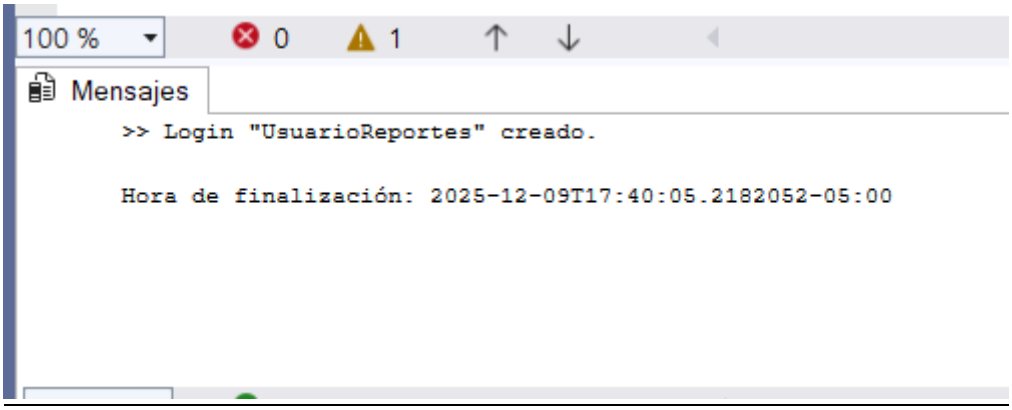
   Crear un Resource Pool que limite el uso de CPU al 20% para consultas analíticas pesadas.

Script:

Preparación del Entorno (Simulación del Usuario)

Para probar esto, crearemos un usuario específico (UsuarioReportes) al cual le aplicaremos el castigo (límite de CPU).

```
USE master;
GO

-- Creamos un login para simular al "culpable" de las consultas pesadas
IF NOT EXISTS (SELECT name FROM sys.sql_logins WHERE name =
'UsuarioReportes')
BEGIN
    CREATE LOGIN UsuarioReportes WITH PASSWORD = 'Password123!';
    PRINT '>> Login "UsuarioReportes" creado.';
END
GO
```

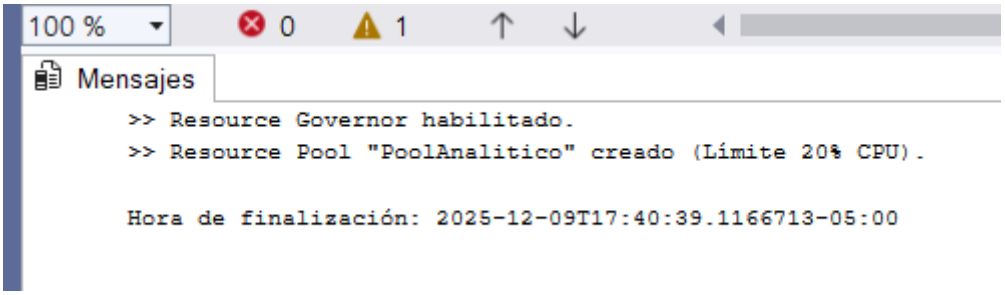


Implementación del Resource Governor

Aquí configuraremos el límite del 20%. Como son comandos de configuración de servidor, se ejecutan paso a paso.

```
USE master;
GO

BEGIN TRY
    -- 1. Habilitar el Resource Governor (por si estaba apagado)
    ALTER RESOURCE GOVERNOR RECONFIGURE;
    PRINT '>> Resource Governor habilitado.';

    -- 2. Crear el "Pool" de Recursos (La jaula)
    -- Aquí definimos el límite físico: MAX_CPU_PERCENT = 20
    IF NOT EXISTS (SELECT name FROM sys.resource_governor_resource_pools
WHERE name = 'PoolAnalitico')
    BEGIN
        CREATE RESOURCE POOL PoolAnalitico
        WITH (MAX_CPU_PERCENT = 20); -- ¡Aquí está la magia!
        PRINT '>> Resource Pool "PoolAnalitico" creado (Límite 20% CPU).';
    END
```

```
    -- 3. Crear el "Grupo" de Carga (La etiqueta)
    -- Este grupo usará la jaula que creamos arriba
    IF NOT EXISTS (SELECT name FROM sys.resource_governor_workload_groups
WHERE name = 'GrupoReportes')
    BEGIN
        CREATE WORKLOAD GROUP GrupoReportes
        USING PoolAnalitico;
        PRINT '>> Workload Group "GrupoReportes" asociado al Pool.';
    END

    -- Aplicar cambios en memoria
    ALTER RESOURCE GOVERNOR RECONFIGURE;

END TRY
BEGIN CATCH
    PRINT '>> ERROR CONFIGURANDO RG: ' + ERROR_MESSAGE();
END CATCH;
GO
```

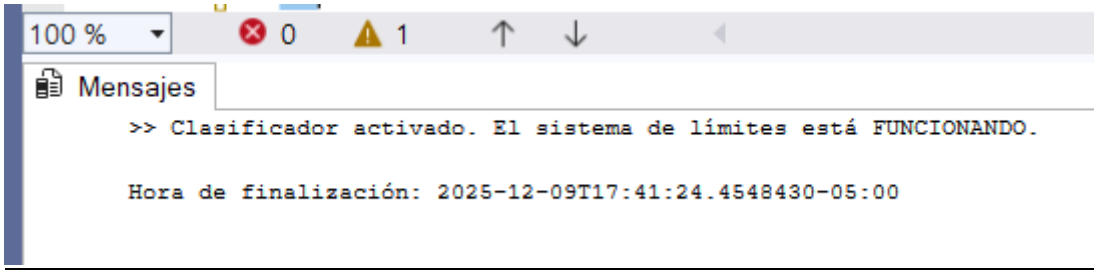


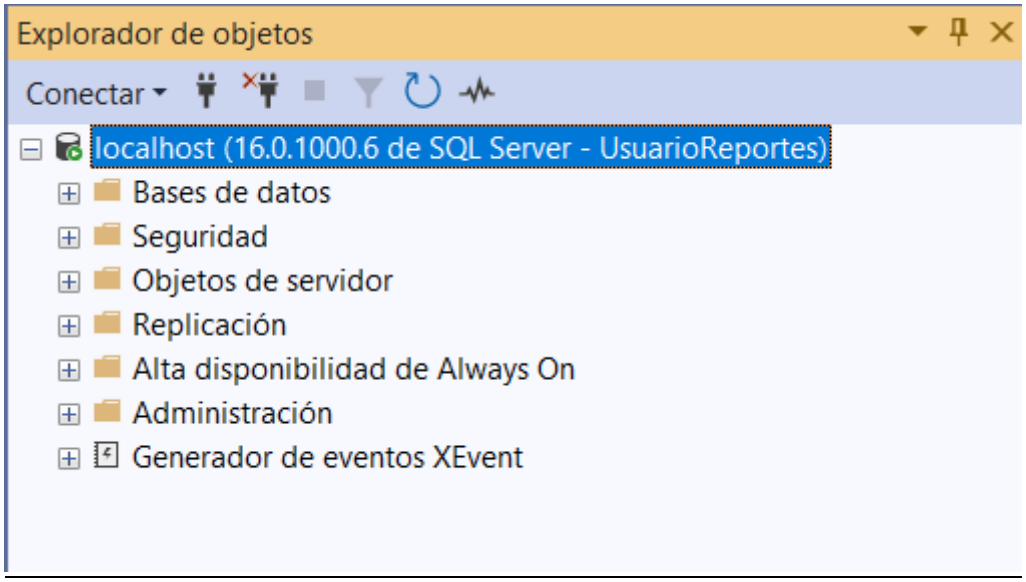
La Función Clasificadora (El Portero)

Ahora necesitamos una función que le diga a SQL Server: *"Si entra el UsuarioReportes, mándalo al grupo limitado. Si es cualquier otro, déjalo pasar normal"*

```
USE master;
GO

-- Si ya existe la función, la borramos para recrearla limpia
IF OBJECT_ID('dbo.fn_ClasificadorReportes') IS NOT NULL
    DROP FUNCTION dbo.fn_ClasificadorReportes;
GO

-- Crear la función clasificadora
CREATE FUNCTION dbo.fn_ClasificadorReportes()
RETURNS SYSNAME -- Devuelve el nombre del grupo
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @NombreGrupo SYSNAME;

    -- LÓGICA: Si el usuario es 'UsuarioReportes', castigarlo.
    IF SUSER_NAME() = 'UsuarioReportes'
    BEGIN
        SET @NombreGrupo = 'GrupoReportes';
    END
    ELSE
    BEGIN
        -- Si es cualquier otro (ej. sa), va al grupo por defecto (sin
límites)
        SET @NombreGrupo = 'default';
    END
```

```
    RETURN @NombreGrupo;
END;
GO

-- 4. Asignar esta función al Gobernador y reiniciar
ALTER RESOURCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = dbo.fn_ClasificadorReportes);

ALTER RESOURCE GOVERNOR RECONFIGURE;
PRINT '>> Clasificador activado. El sistema de límites está FUNCIONANDO.';
GO
```





Justificación:

```
-- EJECUTAR COMO ADMINISTRADOR (NO como UsuarioReportes)
SELECT
    s.session_id,
    s.login_name,
    g.name as [Grupo_Carga],
    p.name as [Pool_Recursos (La Jaula)],
    p.max_cpu_percent as [Limite_CPU]
FROM sys.dm_exec_sessions s
LEFT JOIN sys.resource_governor_workload_groups g ON s.group_id = g.group_id
LEFT JOIN sys.resource_governor_resource_pools p ON g.pool_id = p.pool_id
WHERE s.login_name = 'UsuarioReportes'; -- Filtramos solo al usuario que nos
interesa
```

100 % No se encontraron problemas.

Resultados Mensajes

	session_id	login_name	Grupo_Carga	Pool_Recursos (La Jaula)	Limite_CPU
1	62	UsuarioReportes	GrupoReportes	PoolReportes	20
2	63	UsuarioReportes	GrupoReportes	PoolReportes	20
3	65	UsuarioReportes	GrupoReportes	PoolReportes	20
4	76	UsuarioReportes	GrupoReportes	PoolReportes	20
5	77	UsuarioReportes	GrupoReportes	PoolReportes	20
6	78	UsuarioReportes	GrupoReportes	PoolReportes	20
7	79	UsuarioReportes	GrupoReportes	PoolReportes	20

Explicación de las buenas prácticas:

La implementación de **Resource Governor** debe seguir una estrategia de contención progresiva: primero se define un **Resource Pool** para establecer los límites físicos (CPU/Memoria), luego un **Workload Group** para organizar las solicitudes, y finalmente una **Classifier Function** eficiente (schemabinding) que redirija el tráfico basándose en criterios ligeros como el `SUSER_NAME()` o `APP_NAME()`. Es vital probar la función clasificadora exhaustivamente antes de activarla, ya que un error en ella podría bloquear nuevas conexiones a la instancia; además, siempre se debe mantener el grupo `default` sin restricciones severas para asegurar que las tareas administrativas y críticas no se vean afectadas.

Proyecto 10: Crear un trigger de auditoría ligera usando Extended Events

1. Enunciado del ejercicio

   Auditar inserciones en la tabla Productos usando Extended Events sin afectar rendimiento.

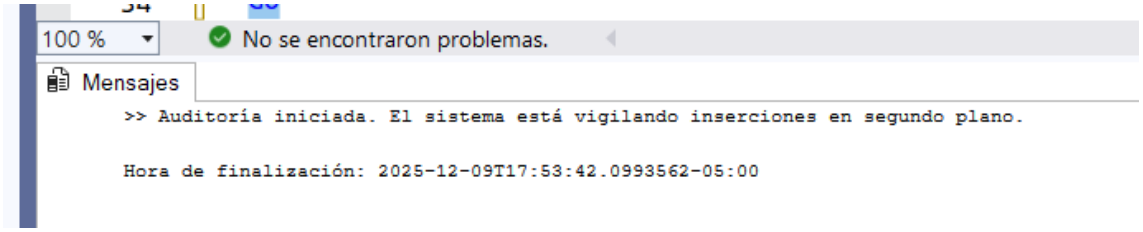
Script:

Creación de la Sesión de Auditoría (XEvent)

Vamos a crear un "Event Session" que escuche silenciosamente cada vez que alguien ejecute un INSERT en la tabla Productos.

```
-- === PROYECTO 10: AUDITORÍA CON EXTENDED EVENTS ===
USE master;
GO

-- 1. Si ya existe la sesión, la borramos para empezar de cero
IF EXISTS (SELECT * FROM sys.server_event_sessions WHERE name =
'Auditoria_Inserciones_Productos')
    DROP EVENT SESSION [Auditoria_Inserciones_Productos] ON SERVER;
GO

-- 2. Crear la sesión de eventos
-- Capturamos: Quién, Cuándo, Desde qué App y Qué comando ejecutó.
CREATE EVENT SESSION [Auditoria_Inserciones_Productos] ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION(
        sqlserver.database_name,    -- En qué BD ocurrió
        sqlserver.client_app_name, -- Qué programa usó (SSMS, .NET, Java)
        sqlserver.username,         -- Quién fue
```

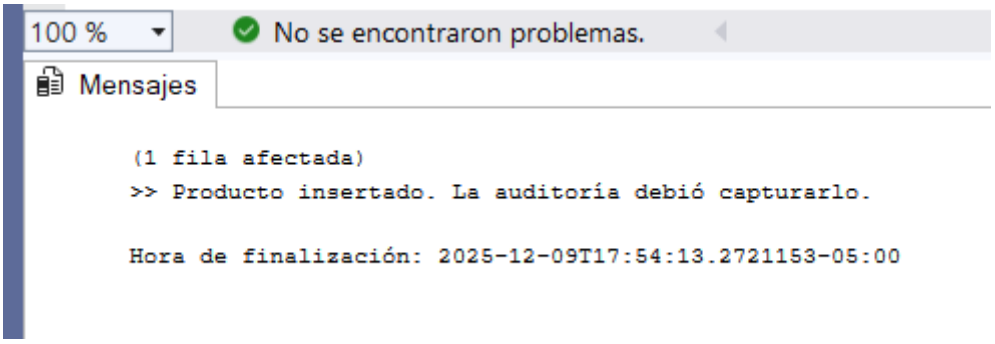
```
        sqlserver.sql_text          -- El código exacto ejecutado
    )
    -- FILTRO: Solo queremos ver INSERTs que afecten a la tabla Productos
    WHERE (
        sqlserver.like_i_sql_unicode_string(sql_text, N'%INSERT%Productos%')
        AND sqlserver.database_name = N'QhatuPeru' -- Asegúrate que coincida
con tu BD
    )
)
ADD TARGET package0.ring_buffer -- Guardar en memoria (RAM) para lectura
rápida
WITH (STARTUP_STATE = OFF);
GO

-- 3. Iniciar la "Cámara de Seguridad"
ALTER EVENT SESSION [Auditoria_Inserciones_Productos] ON SERVER STATE =
START;
PRINT '>> Auditoría iniciada. El sistema está vigilando inserciones en
segundo plano.';
GO
```



Generar Tráfico (Simular el "Delito")

Ahora que la auditoría está corriendo, vamos a insertar un producto para ver si lo detecta.

```
USE QhatuPeru;
GO

-- Insertar un producto nuevo para probar la auditoría
INSERT INTO dbo.Productos (ProductoID, Nombre, Precio, Stock)
VALUES (999, 'Producto Espía X', 50.00, 10);

PRINT '>> Producto insertado. La auditoría debió capturarlo.';
```

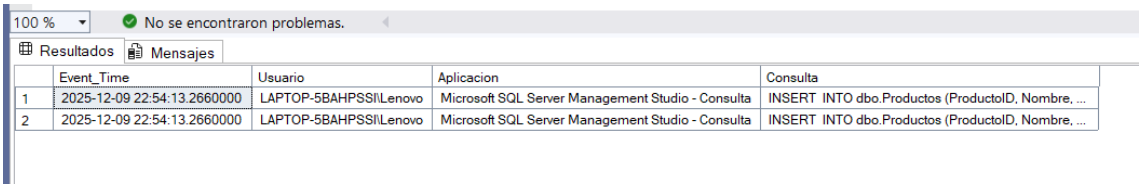


Consultar la Evidencia (Leer el Log)

A diferencia de una tabla normal, los XEvents se guardan en un formato XML en memoria. Para leerlos de forma humana, usamos este script.

```
-- === LECTURA DE DATOS DE EXTENDED EVENTS ===
DECLARE @Target_Data XML;
```

```sql
-- Extraemos los datos crudos de la memoria (Ring Buffer)
SELECT @Target_Data = CAST(xet.target_data AS XML)
FROM sys.dm_xe_session_targets AS xet
JOIN sys.dm_xe_sessions AS xe ON xe.address = xet.event_session_address
WHERE xe.name = 'Auditoria_Inserciones_Productos'
  AND xet.target_name = 'ring_buffer';

-- Procesamos el XML para verlo como tabla
SELECT
    Event_Time = T.C.value('(@timestamp)[1]', 'datetime2'),
    Usuario    = T.C.value('(action[@name="username"]/value)[1]',
'nvarchar(100)'),
    Aplicacion = T.C.value('(action[@name="client_app_name"]/value)[1]',
'nvarchar(100)'),
    Consulta   = T.C.value('(action[@name="sql_text"]/value)[1]',
'nvarchar(max)')
FROM @Target_Data.nodes('//RingBufferTarget/event') AS T(C);
```


100 % No se encontraron problemas.

Resultados Mensajes

	Event_Time	Usuario	Aplicacion	Consulta
1	2025-12-09 22:54:13.2660000	LAPTOP-5BAHPSSI\Lenovo	Microsoft SQL Server Management Studio - Consulta	INSERT INTO dbo.Productos (ProductoID, Nombre, ...
2	2025-12-09 22:54:13.2660000	LAPTOP-5BAHPSSI\Lenovo	Microsoft SQL Server Management Studio - Consulta	INSERT INTO dbo.Productos (ProductoID, Nombre, ...

Explicación de las buenas prácticas:

En este proyecto aplicamos **auditoría asíncrona** mediante Extended Events, la cual es superior a los Triggers tradicionales para tareas de monitoreo porque tiene un **impacto casi nulo en el rendimiento**. Mientras un Trigger ocurre dentro de la transacción (bloqueando al usuario hasta que termina), los Extended Events funcionan capturando la actividad en un hilo separado (Fire-and-Forget). Además, utilizamos el `ring_buffer` como destino para pruebas rápidas en memoria, evitando la latencia de disco, aunque en un entorno de producción real configuraríamos un `event_file` para asegurar la persistencia de los logs de auditoría a largo plazo.
