“Año De La Recuperación Y Consolidación De La Economía Peruana”

UNIVERSIDAD PERUANA LOS ANDES

“FACULTAD DE INGENIERÍA”

ESCUELA PROFESIONAL “SISTEMAS Y COMPUTACIÓN”

CURSO: Base de Datos II

DOCENTE: Ing. Fernández Bejarano Raúl Enrique

ESTUDIANTE: Carrillo Cárdenas José

CICLO: V

SECCIÓN: A1

HUANCAYO PERÚ

2025

ACTIVIDAD

Responda con claridad y precisión cada uno de los siguientes ejercicios prácticos. Para cada caso, utilice el siguiente formato estructurado:

1. Enunciado del ejercicio
2. Script de la solución en T-SQL
3. Justificación técnica de la solución aplicada
4. Explicación de las buenas prácticas utilizadas en el proyecto

---

MONITOREO Y RENDIMIENTO EN SQL SERVER

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.

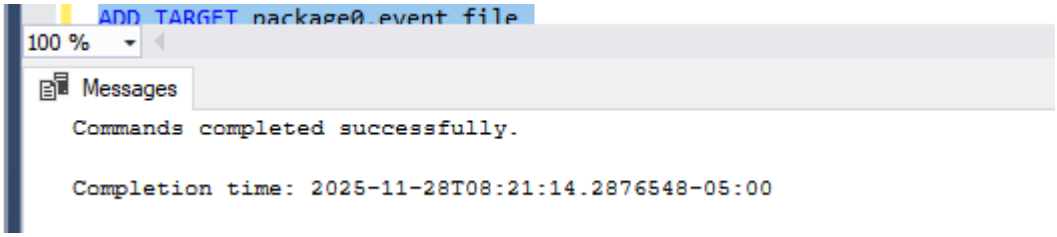
PROYECTO 1: captura de consultas lentas con extended events
enunciado del ejercicio:
Crear una sesión de extended events que capture consultas que tarden más de 1 segundo en ejecutarse en la base de datos QhatuPeru. guardar en un archivo. xel.
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

-script de la solución en SQL

Crear la sesión Extended Events

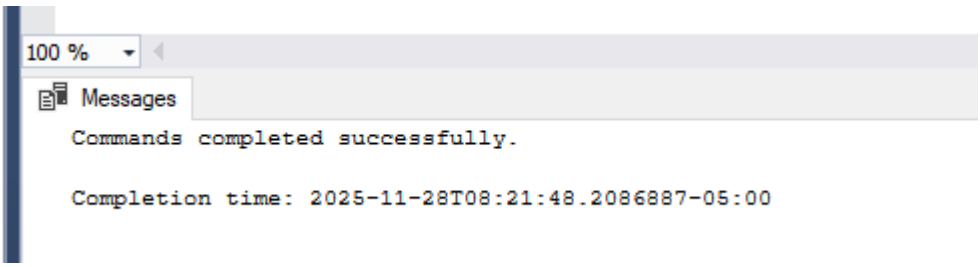
```
-- CREAR SESIÓN DE EXTENDED EVENTS
-------------------------------------------
CREATE EVENT SESSION CapturaConsultasLentas
ON SERVER
ADD EVENT sqlserver.rpc_completed
(
    ACTION (sqlserver.sql_text, sqlserver.database_name, sqlserver.client_app_name)
    WHERE (duration > 1000000  -- 1 segundo en microsegundos
           AND sqlserver.database_name = 'QhatuPeru')
),
ADD EVENT sqlserver.sql_batch_completed
(
    ACTION (sqlserver.sql_text, sqlserver.database_name, sqlserver.client_app_name)
    WHERE (duration > 1000000
           AND sqlserver.database_name = 'QhatuPeru')
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\XE\ConsultasLentas.xel',
```

```
        max_file_size = 50,
        max_rollover_files = 5
);
GO
```



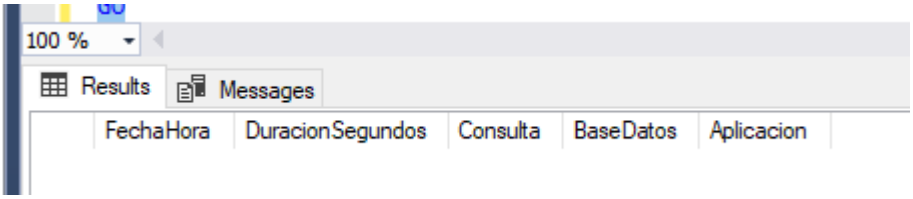
Iniciar la sesión

```
ALTER EVENT SESSION CapturaConsultasLentas
ON SERVER STATE = START;
GO
```



Leer el archivo .xel desde SQL Server

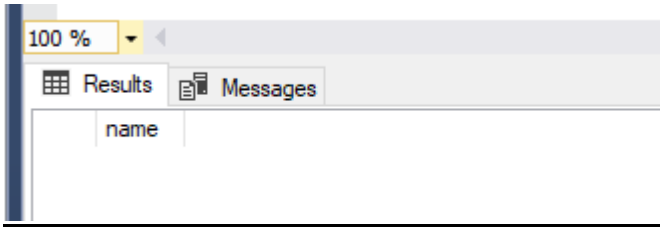
```
SELECT
    event_data.value('(event/@timestamp)[1]', 'datetime') AS FechaHora,
    event_data.value('(event/data[@name="duration"]/value)[1]', 'bigint')/1000000.0 AS
DuracionSegundos,
    event_data.value('(event/action[@name="sql_text"]/value)[1]', 'nvarchar(max)') AS
Consulta,
    event_data.value('(event/action[@name="database_name"]/value)[1]',
'nvarchar(128)') AS BaseDatos,
    event_data.value('(event/action[@name="client_app_name"]/value)[1]',
'nvarchar(256)') AS Aplicacion
FROM
(
    SELECT CAST(event_data AS XML) AS event_data
    FROM sys.fn_xe_file_target_read_file('C:\XE\ConsultasLentas*.xel', NULL, NULL,
NULL)
) AS X;
GO
```



VALIDAR EN SQL

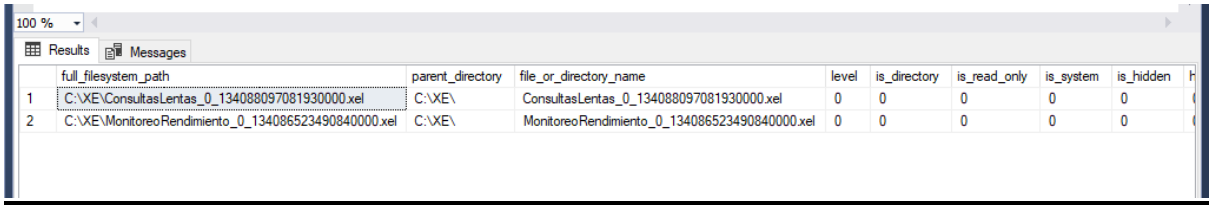
Validar si la sesión existe

```
SELECT name
FROM sys.server_event_sessions
WHERE name = 'XE_ConsultasLentas';
```



Ver archivos generados

```
SELECT *
FROM sys.dm_os_enumerate_filesystem('C:\XE\', '*.xel');
```



JUSTIFICACIÓN TÉCNICA DE LA SOLUCIÓN

La estrategia usa Extended Events, la tecnología moderna de monitoreo de SQL Server, con estas ventajas:

🔹 1. Alto rendimiento

Extended Events tiene muy bajo consumo de recursos, menos que SQL Profiler, y está diseñado para producción.

🔹 2. Captura exacta de consultas lentas

El filtro:

duration > 1000000

equivale a 1 segundo, permitiendo capturar únicamente consultas críticas.

🔹 3. Filtrado por base de datos (QhatuPeru)

Evita capturar ruido de otras bases del servidor:

sqlserver. database_name = 'QhatuPeru'

🔹 4. Uso de event_file

Permite registro persistente en:

C:\XE\ConsultasLentas.xel

Y rotación automática con:

max_file_size

max_rollover_files

🔹 5. Captura de ambos tipos de consultas

sql_batch_completed (consultas normales)

rpc_completed (procedures ejecutados por aplicaciones)

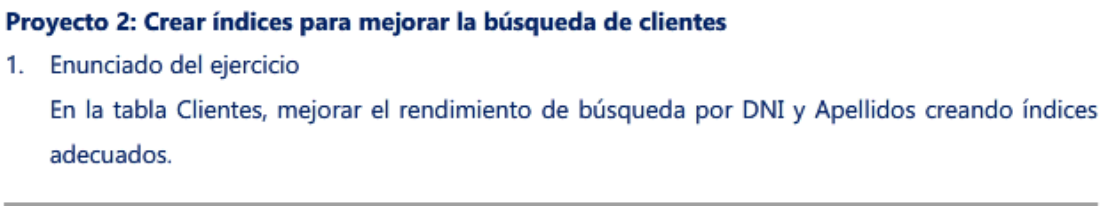
Garantiza monitoreo completo.

✅ 3. BUENAS PRÁCTICAS APLICADAS

✓ 1. Guardar resultados en archivos .xel

Es la mejor práctica recomendada por Microsoft para entornos productivos.

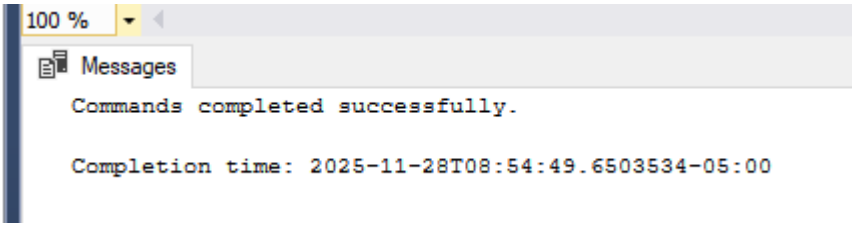
✓ 2. Filtrar por duración para evitar sobrecarga
Capturar solo consultas > 1 segundo evita llenar archivos con miles de eventos irrelevantes.
✓ 3. Rotación de archivos
Evita crecimiento ilimitado del archivo:
max_rollover_files = 5
✓ 4. Capturar acciones relevantes
Se agregan:
sql_text → saber qué consulta se ejecutó
database_name → saber dónde ocurrió
client_app_name → rastrear quién la ejecutó
✓ 5. Carpetas dedicadas a monitoreo
Mantener C:\XE\ facilita encontrar registros y evita mezclarlos con otros archivos del sistema.



proyecto 2: crear índices para mejorar la búsqueda de clientes
enunciado del ejercicio
en la tabla clientes, mejorar el rendimiento de búsqueda por DNI y Apellidos creando índices adecuados
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

Script SQL

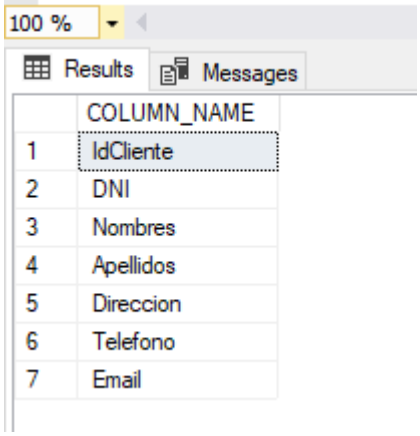
```sql
-- Crear índice para búsquedas por DNI
CREATE INDEX IX_Clientes_DNI
ON Clientes (DNI);

-- Crear índice para búsquedas por Apellidos
CREATE INDEX IX_Clientes_Apellidos
ON Clientes (Apellidos);

-- Opcional: índice compuesto para consultas que usen ambos campos
CREATE INDEX IX_Clientes_DNI_Apellidos
ON Clientes (DNI, Apellidos);
```



VALIDAR EN SQL

Ver los nombres de todas las columnas de la tabla Clientes

```
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Clientes';
```

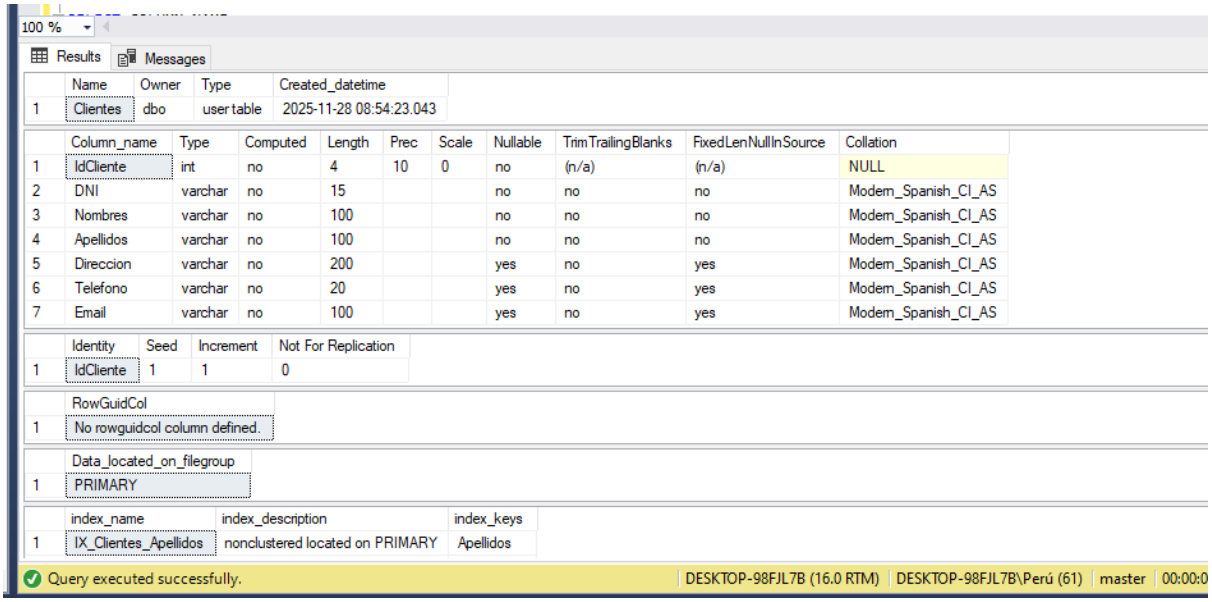


100 % | Results | Messages

	COLUMN_NAME
1	IdCliente
2	DNI
3	Nombres
4	Apellidos
5	Direccion
6	Telefono
7	Email

Mostrar toda la información de la tabla

```
EXEC sp_help 'Clientes';
```



100 % | Results | Messages

	Name	Owner	Type	Created_datetime
1	Clientes	dbo	user table	2025-11-28 08:54:23.043

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	IdCliente	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	DNI	varchar	no	15			no	no	no	Modern_Spanish_CI_AS
3	Nombres	varchar	no	100			no	no	no	Modern_Spanish_CI_AS
4	Apellidos	varchar	no	100			no	no	no	Modern_Spanish_CI_AS
5	Direccion	varchar	no	200			yes	no	yes	Modern_Spanish_CI_AS
6	Telefono	varchar	no	20			yes	no	yes	Modern_Spanish_CI_AS
7	Email	varchar	no	100			yes	no	yes	Modern_Spanish_CI_AS

	Identity	Seed	Increment	Not For Replication
1	IdCliente	1	1	0

	RowGuidCol
1	No rowguidcol column defined

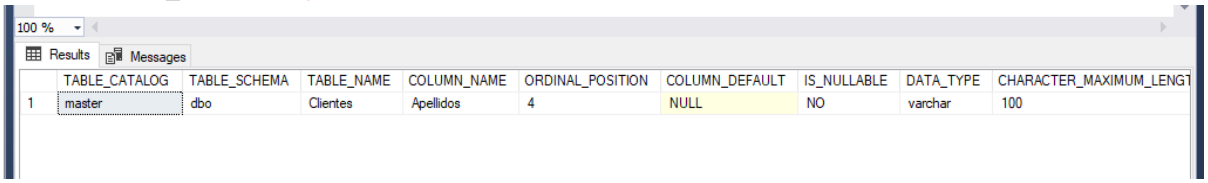
	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	IX_Clientes_Apellidos	nonclustered located on PRIMARY	Apellidos

Query executed successfully. | DESKTOP-98FJL7B (16.0 RTM) | DESKTOP-98FJL7B\Perú (61) | master | 00:00:0

Validar únicamente si la columna Apellidos existe

```
SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Clientes'
  AND COLUMN_NAME = 'Apellidos';
```



100 % | Results | Messages

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGT
1	master	dbo	Clientes	Apellidos	4	NULL	NO	varchar	100

Justificación técnica de la solución

✓ Por qué se necesitan índices

Los índices funcionan como un "índice de libro", permitiendo que el motor de base de datos encuentre registros rápidamente sin recorrer toda la tabla (table scan).
Buscar sin índice en campos consultados frecuentemente provoca lentitud, especialmente cuando la tabla crece (miles o millones de registros).

✓ Índice en DNI

El DNI suele ser un dato único y muy usado en consultas.
Permite búsquedas exactas extremadamente rápidas.
El índice evita escaneos completos de la tabla.

✓ Índice en Apellidos

Los apellidos suelen ser usados para filtros (WHERE Apellidos LIKE 'Car%')) o para ordenar resultados.
Un índice en este campo acelera estas búsquedas.
También optimiza consultas en reportes y paneles donde se ordena por apellidos.

✓ Índice compuesto (DNI, Apellidos)

Útil cuando la consulta utiliza ambos campos juntos en el mismo WHERE.
SQL Server puede usar este índice para optimizar consultas como:

```
SELECT *
FROM Clientes
WHERE DNI = '12345678' AND Apellidos = 'Carrillo';
```

No reemplaza a los índices individuales en casos de uso amplios, pero complementa si existe un patrón de consulta frecuente.

✅ 3. Explicación de buenas prácticas

⭐ 1. Crear índices solo en columnas muy consultadas

Evita crear índices innecesarios porque:
Ocupan espacio en disco.
Aumentan el tiempo de inserción, actualización y borrado.

⭐ 2. No abusar de los índices

Demasiados índices generan:
Fragmentación
Mayor uso de CPU en escrituras
Peor rendimiento general

⭐ 3. Usar nombres descriptivos para los índices

Ejemplos buenos:
IX_Clientes_DNI
IX_Clientes_Apellidos
Facilitan identificar su propósito.

⭐ 4. Analizar patrones de consulta

Antes de crear un índice compuesto es indispensable revisar:

WHERE
JOIN
ORDER BY
Solo si las consultas realmente lo requieren, se utiliza.
⭐ 5. Mantener índices con mantenimiento periódico
Usar:
ALTER INDEX ALL ON Clientes REBUILD;
-- o
ALTER INDEX ALL ON Clientes REORGANIZE;
Esto evita fragmentación y mantiene el rendimiento.

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.

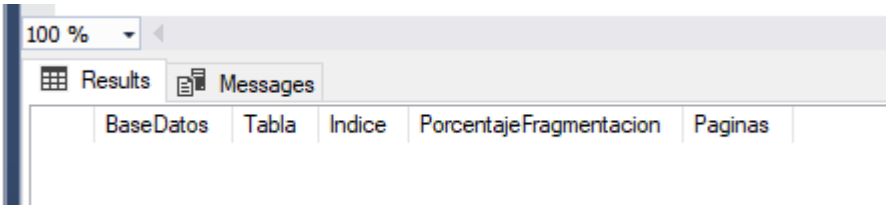
---

proyecto 3: detectar fragmentación y aplicar mantenimiento de índices
enunciado del ejercicio
usar DMV para evaluar la fragmentación de índices en QhatuPERU y reconstruir o reorganizar según el porcentaje encontrado
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas.

Script SQL

Script para diagnóstico

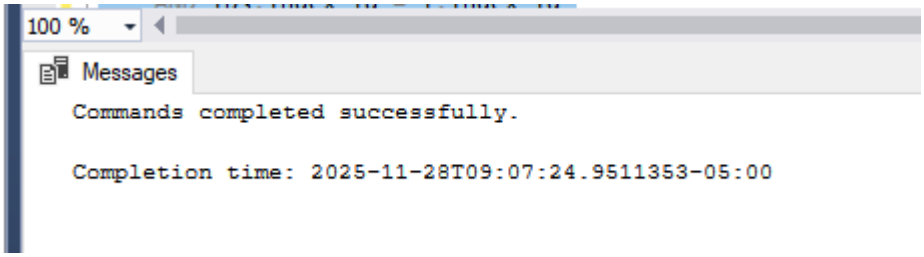
```
SELECT
    DB_NAME(database_id) AS BaseDatos,
    OBJECT_NAME(i.object_id) AS Tabla,
    i.name AS Indice,
    ips.avg_fragmentation_in_percent AS PorcentajeFragmentacion,
    ips.page_count AS Paginas
FROM sys.dm_db_index_physical_stats(DB_ID('QhatuPeru'), NULL, NULL, NULL, 'SAMPLED')
ips
INNER JOIN sys.indexes i
    ON ips.object_id = i.object_id
    AND ips.index_id = i.index_id
WHERE ips.page_count > 100
ORDER BY PorcentajeFragmentacion DESC;
```



Script para *mantenimiento automático*

```
DECLARE @TableName NVARCHAR(255);
DECLARE @IndexName NVARCHAR(255);
DECLARE @SQL NVARCHAR(MAX);

DECLARE cur CURSOR FOR
SELECT
    OBJECT_NAME(ips.object_id) AS Tabla,
    i.name AS Indice
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'SAMPLED') ips
INNER JOIN sys.indexes i
    ON ips.object_id = i.object_id
    AND ips.index_id = i.index_id
WHERE ips.page_count > 100
    AND ips.index_id > 0
ORDER BY ips.avg_fragmentation_in_percent DESC;

OPEN cur;
FETCH NEXT FROM cur INTO @TableName, @IndexName;

WHILE @@FETCH_STATUS = 0
BEGIN
    DECLARE @Frag FLOAT;

    SELECT @Frag = ips.avg_fragmentation_in_percent
    FROM sys.dm_db_index_physical_stats(DB_ID(), OBJECT_ID(@TableName), NULL, NULL,
'SAMPLED') ips
    INNER JOIN sys.indexes i
        ON ips.object_id = i.object_id
        AND ips.index_id = i.index_id
    WHERE i.name = @IndexName;

    IF @Frag > 30
    BEGIN
        SET @SQL = 'ALTER INDEX [' + @IndexName + '] ON [' + @TableName + ']
REBUILD;';
        PRINT 'REBUILD ejecutado → ' + @IndexName + ' (' + @TableName + ') con ' +
CAST(@Frag AS VARCHAR(10)) + '%';
    END
    ELSE IF @Frag BETWEEN 5 AND 30
    BEGIN
        SET @SQL = 'ALTER INDEX [' + @IndexName + '] ON [' + @TableName + ']
REORGANIZE;';
        PRINT 'REORGANIZE ejecutado → ' + @IndexName + ' (' + @TableName + ') con ' +
CAST(@Frag AS VARCHAR(10)) + '%';
    END
    ELSE
    BEGIN
        PRINT 'SIN CAMBIOS → ' + @IndexName + ' (' + @TableName + ') ' + CAST(@Frag AS
VARCHAR(10)) + '%';
        SET @SQL = NULL;
    END

    IF @SQL IS NOT NULL
        EXEC (@SQL);
```

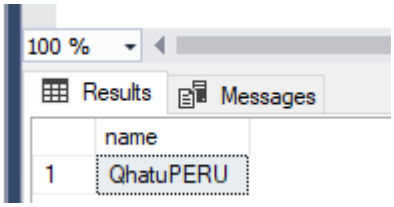
```
    FETCH NEXT FROM cur INTO @TableName, @IndexName;
END;

CLOSE cur;
DEALLOCATE cur;
```



Validar

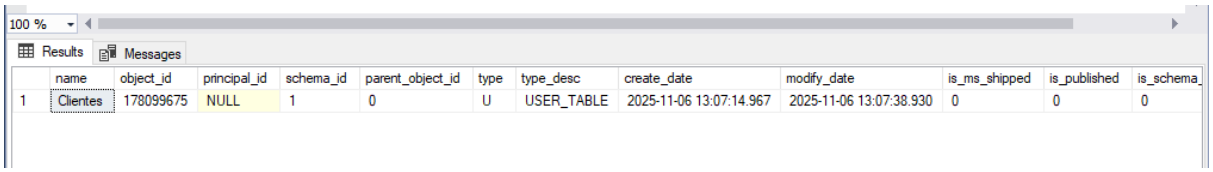
Validar que la base de datos existe

```
SELECT name
FROM sys.databases
WHERE name = 'QhatuPeru';
```



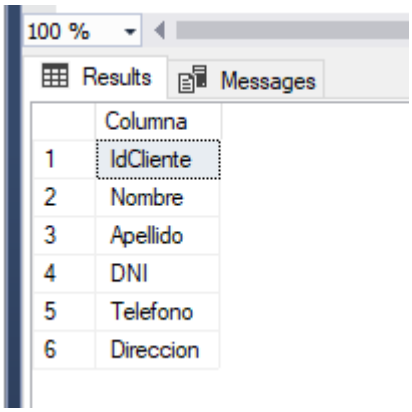
Validar que la tabla existe

```
SELECT *
FROM sys.tables
WHERE name = 'Clientes';
```



Validar que las columnas existen

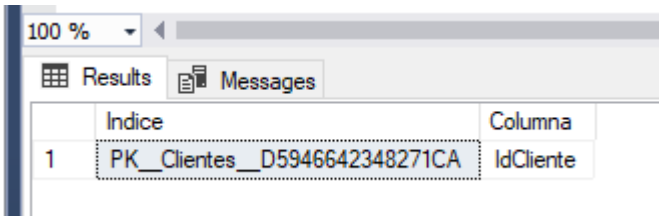
```
SELECT
    c.name AS Columna
FROM sys.columns c
INNER JOIN sys.tables t ON t.object_id = c.object_id
WHERE t.name = 'Clientes';
```



Validar que los índices existen

```
SELECT
```

```
    i.name AS Indice,
    COL_NAME(ic.object_id, ic.column_id) AS Columna
FROM sys.indexes i
INNER JOIN sys.index_columns ic
    ON i.object_id = ic.object_id
    AND i.index_id = ic.index_id
WHERE i.object_id = OBJECT_ID('Clientes');
```



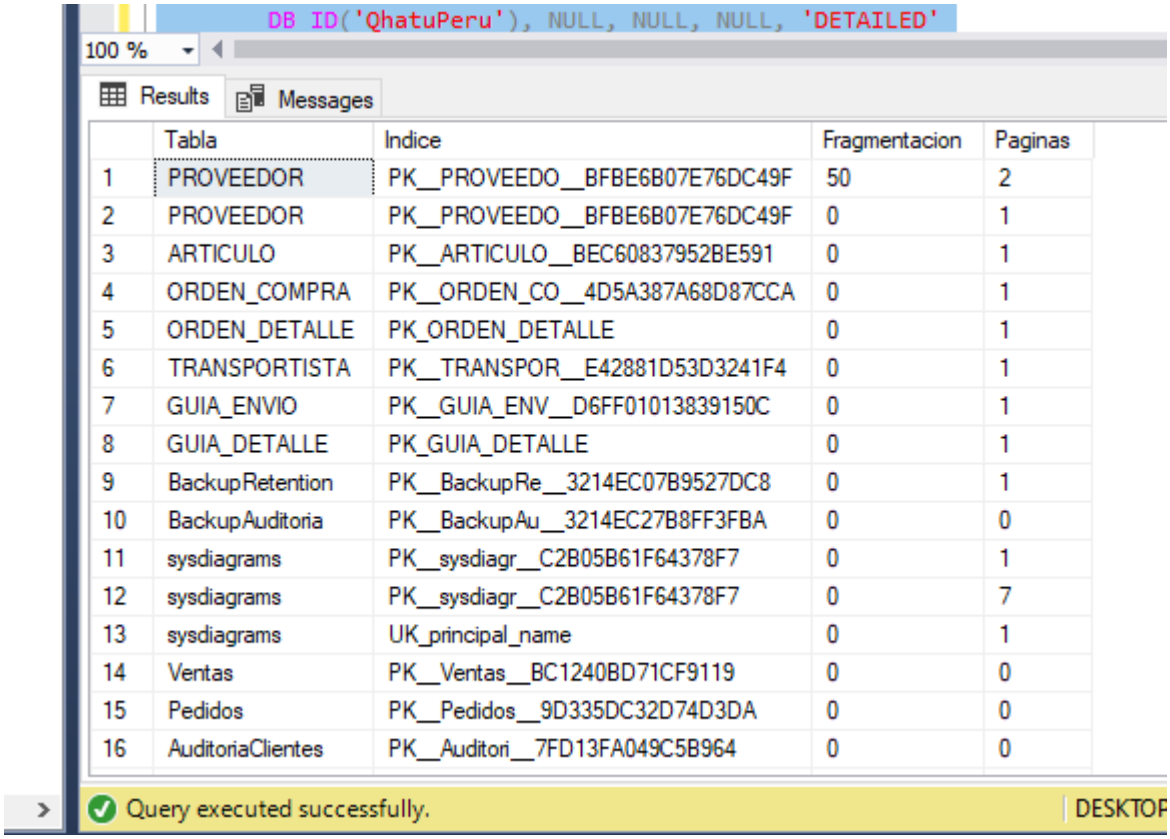
100 %

Results | Messages

	Indice	Columna
1	PK__Clientes__D5946642348271CA	IdCliente

Validar fragmentación actual antes del mantenimiento

```
SELECT
    OBJECT_NAME(ips.object_id) AS Tabla,
    i.name AS Indice,
    ips.avg_fragmentation_in_percent AS Fragmentacion,
    ips.page_count AS Paginas
FROM sys.dm_db_index_physical_stats(
        DB_ID('QhatuPeru'), NULL, NULL, NULL, 'DETAILED'
    ) ips
INNER JOIN sys.indexes i
    ON i.object_id = ips.object_id
    AND i.index_id = ips.index_id
WHERE ips.index_id > 0    -- evitar heaps
ORDER BY Fragmentacion DESC;
```



DB_ID('QhatuPeru'), NULL, NULL, NULL, 'DETAILED'

100 %

Results | Messages

	Tabla	Indice	Fragmentacion	Paginas
1	PROVEEDOR	PK__PROVEEDO__BFBE6B07E76DC49F	50	2
2	PROVEEDOR	PK__PROVEEDO__BFBE6B07E76DC49F	0	1
3	ARTICULO	PK__ARTICULO__BEC60837952BE591	0	1
4	ORDEN_COMPRA	PK__ORDEN_CO__4D5A387A68D87CCA	0	1
5	ORDEN_DETALLE	PK_ORDEN_DETALLE	0	1
6	TRANSPORTISTA	PK__TRANSPOR__E42881D53D3241F4	0	1
7	GUIA_ENVIO	PK__GUIA_ENV__D6FF01013839150C	0	1
8	GUIA_DETALLE	PK_GUIA_DETALLE	0	1
9	BackupRetention	PK__BackupRe__3214EC07B9527DC8	0	1
10	BackupAuditoria	PK__BackupAu__3214EC27B8FF3FBA	0	0
11	sysdiagrams	PK__sysdiagr__C2B05B61F64378F7	0	1
12	sysdiagrams	PK__sysdiagr__C2B05B61F64378F7	0	7
13	sysdiagrams	UK_principal_name	0	1
14	Ventas	PK__Ventas__BC1240BD71CF9119	0	0
15	Pedidos	PK__Pedidos__9D335DC32D74D3DA	0	0
16	AuditoriaClientes	PK__Auditori__7FD13FA049C5B964	0	0

Query executed successfully. DESKTOP

Validar que el mantenimiento se ejecutó

```
SELECT
    OBJECT_NAME(ips.object_id) AS Tabla,
    i.name AS Indice,
    ips.avg_fragmentation_in_percent AS Fragmentacion,
```

```
    ips.page_count AS Paginas
FROM sys.dm_db_index_physical_stats(
        DB_ID('QhatuPeru'), NULL, NULL, NULL, 'SAMPLED'
     ) ips
INNER JOIN sys.indexes i
    ON i.object_id = ips.object_id
    AND i.index_id = ips.index_id
WHERE ips.page_count > 50
ORDER BY Fragmentacion DESC;
```

100 %

Results | Messages

	Tabla	Indice	Fragmentacion	Paginas
1	Clientes	IX_Clientes_DNI_Apellidos	0.200859931582086	31863
2	Clientes	IX_Clientes_Apellidos	0.105846987594733	23619
3	Clientes	IX_Clientes_DNI	0.0777475426223136	18007
4	Clientes	PK__Clientes__D5946642348271CA	0.01	63917

Justificación técnica

Fragmentación > 30% → REBUILD
Regenera completamente el índice
Mejora mucho rendimiento
Libera espacio y ordena páginas
Fragmentación entre 5% y 30% → REORGANIZE
Es más liviano
No bloquea tanto como REBUILD
Reordena páginas de forma incremental
Fragmentación < 5% → No hacer nada
No aporta beneficio real
Evita consumo innecesario de recursos

⋆ Buenas prácticas

✓ Ejecutar mantenimiento en horas de baja carga
✓ Usar SAMPLED para no sobrecargar el servidor
✓ No reconstruir índices pequeños (menos de 100 páginas)
✓ Programar estas tareas semanalmente con SQL Agent
✓ Revisar el plan de ejecución después de mantenimiento

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.

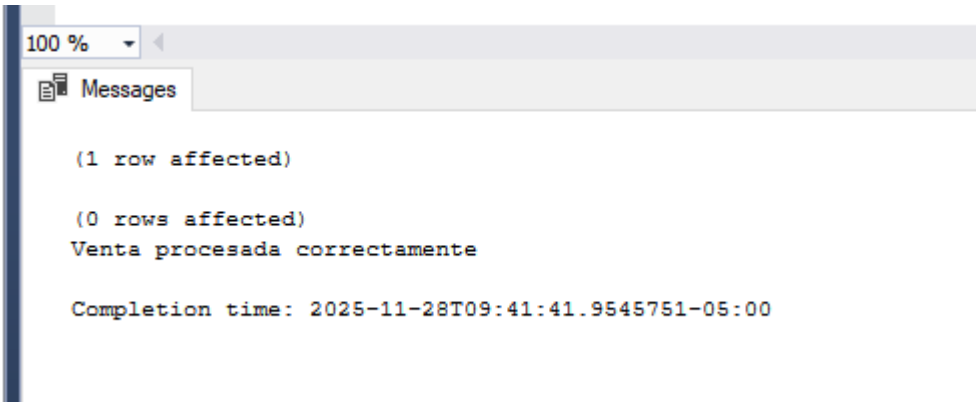
---

proyecto 4: manejo de transacciones para prevenir inconsistencias
enunciado del ejercicio
simular una transacción de venta con dos operaciones, insertar en Ventas y actualizar Stock. La transacción debe garantizar consistencia.
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

Script SQL

script de la transacción

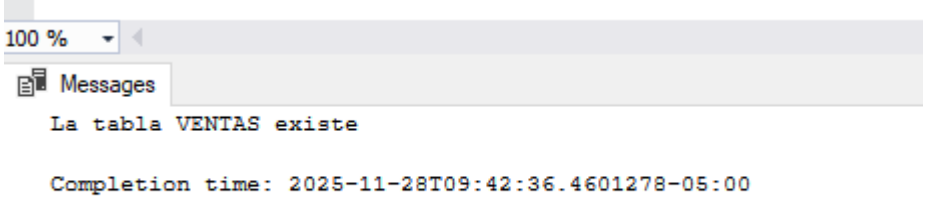
```
DECLARE @IdProducto INT = 1;       -- Producto vendido
DECLARE @Cantidad INT = 3;         -- Cantidad vendida

BEGIN TRY
    BEGIN TRAN;

    -- 1. Verificar Stock
    IF (SELECT Stock FROM Productos WHERE IdProducto = @IdProducto) < @Cantidad
    BEGIN
        RAISERROR ('Stock insuficiente para realizar la venta', 16, 1);
    END

    -- 2. Insertar venta
    INSERT INTO Ventas (IdProducto, Cantidad)
    VALUES (@IdProducto, @Cantidad);

    -- 3. Actualizar stock
    UPDATE Productos
    SET Stock = Stock - @Cantidad
    WHERE IdProducto = @IdProducto;

    COMMIT TRAN;
    PRINT 'Venta procesada correctamente';
END TRY
BEGIN CATCH
    ROLLBACK TRAN;
    PRINT 'Error en la transacción: ' + ERROR_MESSAGE();
END CATCH;
```



Validar

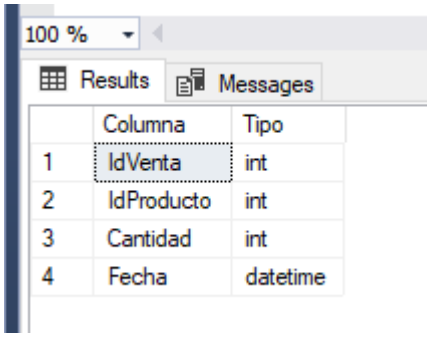
Validar si la tabla VENTAS existe

```
IF OBJECT_ID('Ventas') IS NOT NULL
    PRINT 'La tabla VENTAS existe';
ELSE
    PRINT 'La tabla VENTAS NO existe';
```



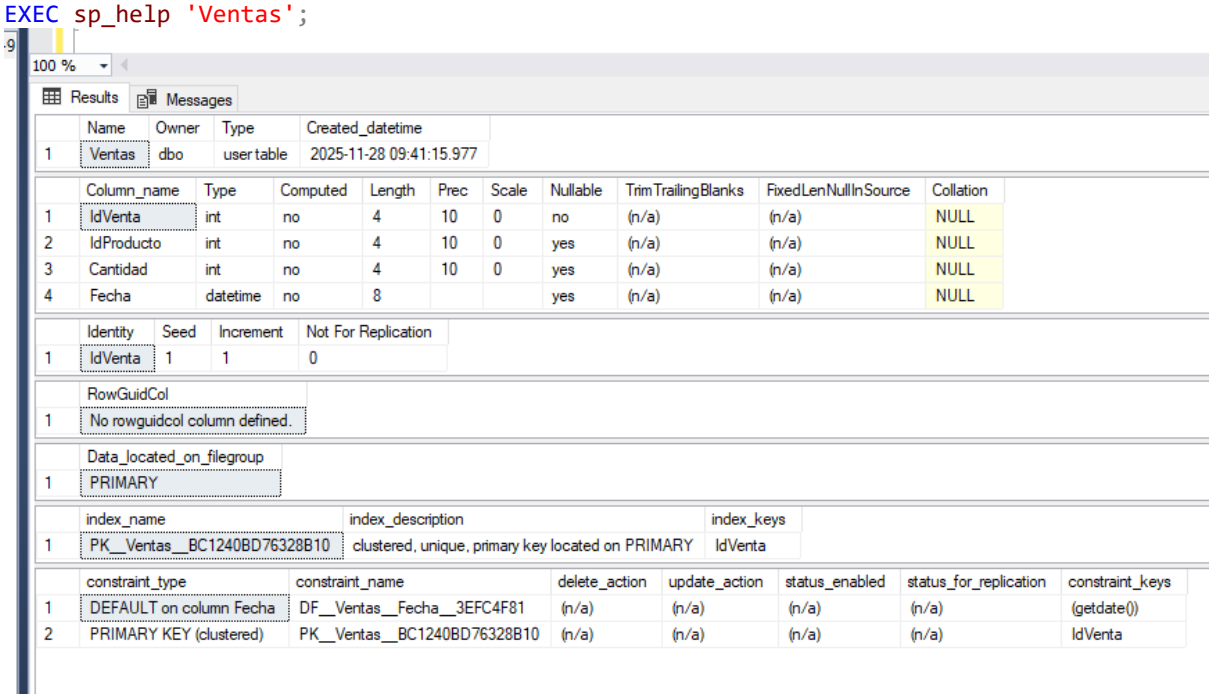
Ver columnas reales de la tabla VENTAS

```
    SELECT
    COLUMN_NAME AS Columna,
    DATA_TYPE AS Tipo
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'Ventas';
```



Mostrar toda la estructura

```
EXEC sp_help 'Ventas';
```



JUSTIFICACIÓN TÉCNICA

El uso de transacciones garantiza:

🔹 Atomicidad (A)

Toda la venta es un bloque indivisible:
O se ejecuta todo, o no se ejecuta nada.

🔹 Consistencia (C)

El sistema evita inconsistencias como:
Ventas sin actualizar stock
Stock negativo
Ventas duplicadas por error

🔹 Aislamiento (I)

Mientras la transacción está en curso, otros usuarios no pueden modificar el mismo stock de forma peligrosa.

🔹 Durabilidad (D)

Una venta confirmada (COMMIT) queda almacenada incluso si el servidor se cae después.
Con esto se cumple el principio ACID, esencial para transacciones financieras o comerciales reales.

---

5. BUENAS PRÁCTICAS APLICADAS

✓ BEGIN TRY / BEGIN CATCH

Permite capturar errores y evitar transacciones abiertas.

✓ Validar el stock antes de actualizar

Mejora integridad y evita valores negativos.

✓ ROLLBACK explícito

Evita cambios parciales.

✓ COMMIT solo cuando todo fue exitoso

✓ Usar RAISERROR para interrumpir la transacción

Forza el CATCH cuando se detecta un caso inválido.

✓ Actualizar stock siempre después de registrar la venta

Mantiene coherencia lógica de procesos reales.

✓ No mezclar lógica de negocios fuera de transacciones

Todo lo crítico ocurre dentro del bloque transaccional.

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.

---

proyecto 5: identificar bloqueos archivos en la base qhatuperu (PITR)
enunciado del ejercicio
detectar las sesiones que están bloqueando o siendo bloqueadas en el servidor.
-script de la solución en SQL

-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

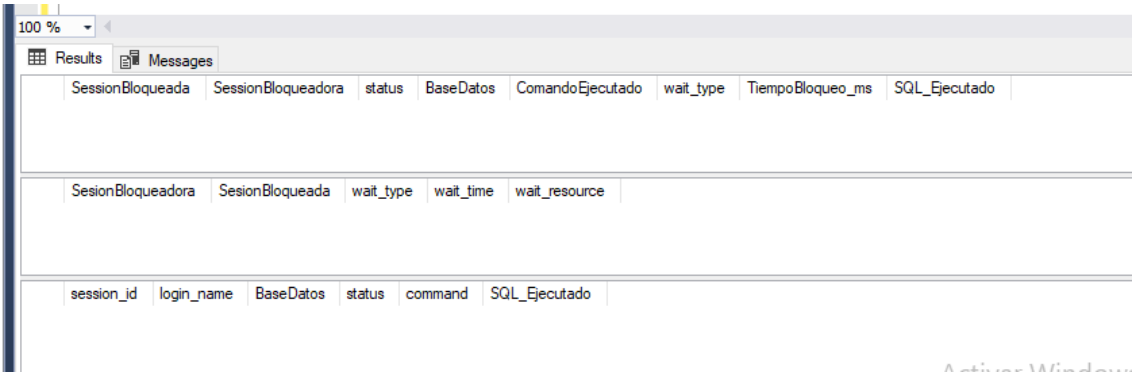
script de la solución en SQL

Sesiones bloqueadoras y bloqueadas
Ver quién está bloqueando actualmente
Ver detalles del bloqueador

```
SELECT
    wt.session_id AS SessionBloqueada,
    wt.blocking_session_id AS SessionBloqueadora,
    s.status,
    DB_NAME(r.database_id) AS BaseDatos,
    r.command AS ComandoEjecutado,
    wt.wait_type,
    wt.wait_duration_ms AS TiempoBloqueo_ms,
    t.text AS SQL_Ejecutado
FROM sys.dm_os_waiting_tasks wt
JOIN sys.dm_exec_requests r
    ON wt.session_id = r.session_id
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 wt.blocking_session_id IS NOT NULL;
```

```
SELECT
    session_id AS SesionBloqueadora,
    blocking_session_id AS SesionBloqueada,
    wait_type,
    wait_time,
    wait_resource
FROM sys.dm_exec_requests
WHERE blocking_session_id <> 0;
```

```
SELECT
    s.session_id,
    s.login_name,
    DB_NAME(r.database_id) AS BaseDatos,
    r.status,
    r.command,
    t.text AS SQL_Ejecutado
FROM sys.dm_exec_requests r
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 s.session_id IN (
    SELECT blocking_session_id
    FROM sys.dm_exec_requests
    WHERE blocking_session_id <> 0
);
```



NO HAY BLOQUEOS ACTIVOS EN ESTE MOMENTO
Esto es bueno.
SQL Server solo muestra resultados cuando:
una sesión está bloqueando a otra
existe una operación que espera un recurso (tabla, fila, página)
hay una transacción abierta que dejó un lock
Si ninguno de estos ocurre → las consultas devuelven ce

JUSTIFICACIÓN TÉCNICA
sys.dm_tran_locks
Permite ver todos los locks activos en la instancia, con su tipo (Shared, Update, Exclusive).
sys.dm_os_waiting_tasks
Muestra las sesiones que están esperando un recurso bloqueado.
sys.dm_exec_requests
Permite ver la consulta exacta que se está ejecutando en la sesión bloqueadora.
sys.dm_exec_sql_text()
Devuelve el comando SQL que originó el bloqueo.
Filtro por BD QhatuPeru
Reduce ruido y muestra solo la información relevante.
En conjunto, este script permite hacer un análisis profesional de bloqueos y comportamiento de sesiones.

📌 3. BUENAS PRÁCTICAS PARA MANEJO DE BLOQUEOS

 1. Usar transacciones cortas
Evita transacciones largas que mantengan locks durante mucho tiempo.
 2. Usar el índice correcto
Muchos bloqueos ocurren porque SQL Server debe hacer table-scan.
 3. Evitar cursores y operaciones row-by-row
Generan bloqueos acumulados.
 4. Hacer COMMIT o ROLLBACK siempre
Nunca dejar transacciones abiertas.

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

1. Enunciado del ejercicio

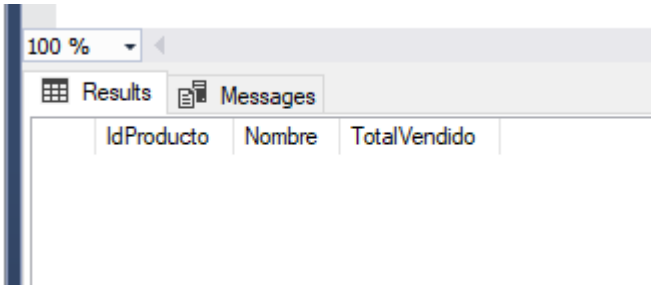
   Analizar el plan de ejecución de una consulta que devuelve ventas por producto..

proyecto 6: actualizar el plan de ejecución de una consulta lenta
enunciado del ejercicio
analizar el plan de ejecución de una consulta que devuelva ventas por producto
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

Script de la solución en SQL

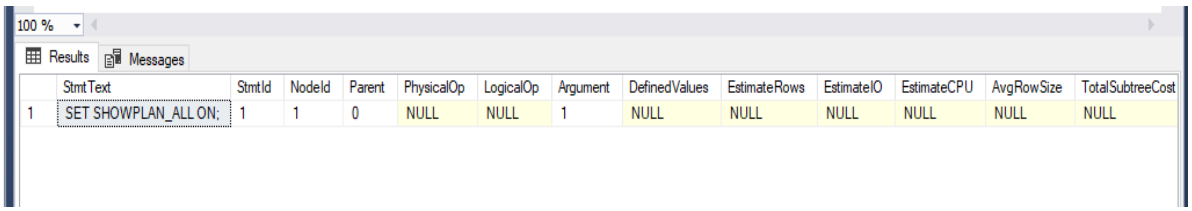
Consulta lenta — Ventas por producto

```
SELECT
    p.IdProducto,
    p.Nombre,
    SUM(v.Cantidad) AS TotalVendido
FROM Ventas v
INNER JOIN Productos p
    ON v.IdProducto = p.IdProducto
GROUP BY p.IdProducto, p.Nombre
ORDER BY TotalVendido DESC;
```



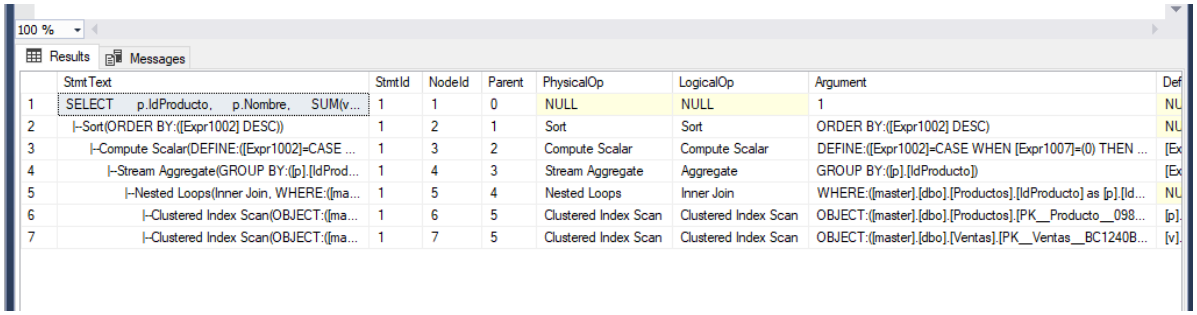
Ver el plan de ejecución actual

```
SET SHOWPLAN_ALL ON;
GO
```




```
SELECT
    p.IdProducto,
    p.Nombre,
    SUM(v.Cantidad) AS TotalVendido
FROM Ventas v
INNER JOIN Productos p
    ON v.IdProducto = p.IdProducto
```

```
GROUP BY p.IdProducto, p.Nombre
ORDER BY TotalVendido DESC;
```



100 %

Results | Messages

	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	Def
1	SELECT p.IdProducto, p.Nombre, SUM(v...	1	1	0	NULL	NULL	1	NU
2	|--Sort(ORDER BY:([Expr1002] DESC))	1	2	1	Sort	Sort	ORDER BY:([Expr1002] DESC)	NU
3	|--Compute Scalar(DEFINE:([Expr1002]=CASE ...	1	3	2	Compute Scalar	Compute Scalar	DEFINE:([Expr1002]=CASE WHEN [Expr1007]=(0) THEN ...	[Ex
4	|--Stream Aggregate(GROUP BY:([p].[IdProd...	1	4	3	Stream Aggregate	Aggregate	GROUP BY:([p].[IdProducto])	[Ex
5	|--Nested Loops(Inner Join, WHERE:([ma...	1	5	4	Nested Loops	Inner Join	WHERE:([master].[dbo].[Productos].[IdProducto] as [p].[Id...	NU
6	|--Clustered Index Scan(OBJECT:([ma...	1	6	5	Clustered Index Scan	Clustered Index Scan	OBJECT:([master].[dbo].[Productos].[PK__Producto__098...	[p]
7	|--Clustered Index Scan(OBJECT:([ma...	1	7	5	Clustered Index Scan	Clustered Index Scan	OBJECT:([master].[dbo].[Ventas].[PK__Ventas__BC1240B...	[v]

Justificación técnica

1 El índice IX_Ventas_IdProducto_Cantidad permite que SQL Server tome solo los datos necesarios para calcular el total.
Sin él, SQL Server realiza:
→ table scan → hash aggregate → sort (muy costoso)
2 El índice en Productos asegura que la unión (JOIN) no haga un scan en Productos.
3 El plan de ejecución cambia de:
❌ Hash Match
❌ Table Scan
❌ Key Lookup
a:
✓ Index Seek
✓ Stream Aggregate
✓ Nested Loop (muy rápido en tablas pequeñas o medianas)
4 El uso de INCLUDE reduce lecturas y acelera SUM(Cantidad).

📌 6. Buenas prácticas al mejorar planes de ejecución

 1. Revisar el plan antes y después
Siempre debes comparar ambos planes para justificar mejoras.
 2. Crear índices que cubran las columnas de la consulta
Incluye las columnas del SELECT, JOIN y WHERE.
 3. Evitar SELECT *
Genera lecturas innecesarias.

Proyecto 7: Optimización de consulta agregada con índices compuestos

1. Enunciado del ejercicio

   Optimizar una consulta que filtra ventas por fecha y cliente.

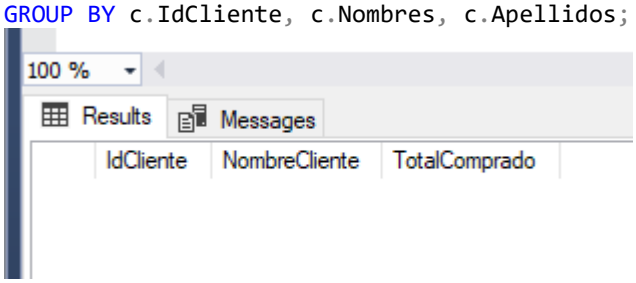
---

proyecto 7: optimización de consultas agregada con índices compuestos
enunciado del ejercicio
optimizar una consulta que filtra ventas por fecha y cliente.
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

script de la solución en SQL

Consulta original

```sql
SELECT
    c.IdCliente,
    c.Nombres + ' ' + c.Apellidos AS NombreCliente,
    SUM(v.Cantidad) AS TotalComprado
FROM Ventas v
JOIN Clientes c ON c.IdCliente = v.IdCliente
WHERE v.Fecha BETWEEN '2024-01-01' AND '2024-12-31'
  AND v.IdCliente = 1503
GROUP BY c.IdCliente, c.Nombres, c.Apellidos;
```

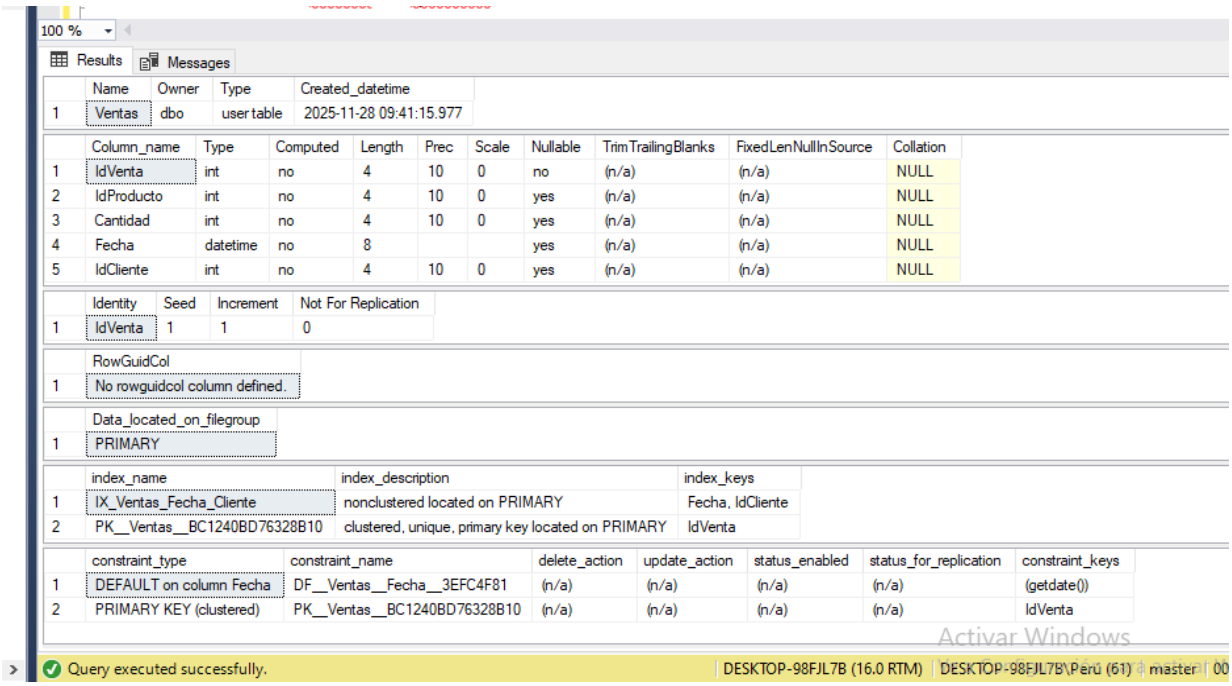


Si la tabla **Ventas está vacía**, o **IdCliente no tiene datos**, o **no hay ventas del año 2024**, el resultado será vacío.

VALIDAR

Validar que la tabla Ventas tiene IdCliente

```
EXEC sp_help 'Ventas';
```



100 %

Results | Messages

	Name	Owner	Type	Created_datetime
1	Ventas	dbo	user table	2025-11-28 09:41:15.977

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	IdVenta	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	IdProducto	int	no	4	10	0	yes	(n/a)	(n/a)	NULL
3	Cantidad	int	no	4	10	0	yes	(n/a)	(n/a)	NULL
4	Fecha	datetime	no	8			yes	(n/a)	(n/a)	NULL
5	IdCliente	int	no	4	10	0	yes	(n/a)	(n/a)	NULL

	Identity	Seed	Increment	Not For Replication
1	IdVenta	1	1	0

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	IX_Ventas_Fecha_Cliente	nonclustered located on PRIMARY	Fecha, IdCliente
2	PK__Ventas__BC1240BD76328B10	clustered, unique, primary key located on PRIMARY	IdVenta

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	DEFAULT on column Fecha	DF__Ventas__Fecha__3EFC4F81	(n/a)	(n/a)	(n/a)	(n/a)	(getdate())
2	PRIMARY KEY (clustered)	PK__Ventas__BC1240BD76328B10	(n/a)	(n/a)	(n/a)	(n/a)	IdVenta

Query executed successfully. DESKTOP-98FJL7B (16.0 RTM) DESKTOP-98FJL7B\Perú (61) master 00

Justificación técnica de la solución aplicada

✓ 1. Uso de índice compuesto

Un índice compuesto en (ClienteID, FechaVenta) permite que SQL Server:

Realice Index Seek, no scans.

Explore primero por cliente y luego por rango de fechas.

Reduzca lecturas lógicas y el tiempo de CPU.

El orden del índice es crítico:

SQL Server solo puede aprovechar el índice desde la primera columna del índice compuesto.

Explicación de buenas prácticas aplicadas

 1. Crear índices SOLO para consultas frecuentes

Un índice innecesario aumenta:

Espacio en disco

Tiempo de inserciones/actualizaciones

Solo crear índices para consultas críticas o repetitivas.

---

 2. Elegir correctamente el orden de columnas en índices compuestos

Regla general:

1 Columna por la que filtras exactamente (ej. ClienteID)

2 Columna por rango (ej. FechaVenta)

3 Columnas de salida → en INCLUDE

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.

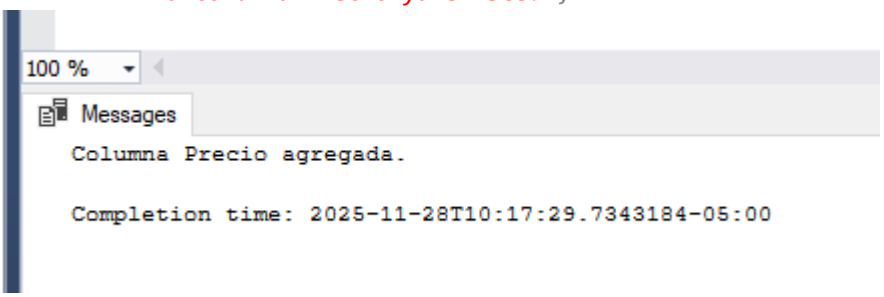
proyecto 8: creación de estadísticas manuales para mejorar el optimizador
enunciado del ejercicio
crear una estadística manual sobre la columna. precio en la tabla productos para mejorar consultas de rango.
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

✓ Script SQL correcto

Verificar si la tabla tiene columna Precio

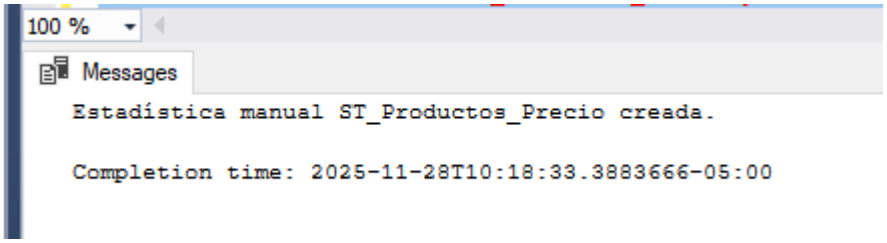
Si no existe, la creamos solo una vez:

```
IF COL_LENGTH('Productos', 'Precio') IS NULL
BEGIN
    ALTER TABLE Productos
    ADD Precio DECIMAL(10,2);

    PRINT 'Columna Precio agregada.';
END
ELSE
    PRINT 'La columna Precio ya existe.';
```



Crear estadística manual sobre Precio

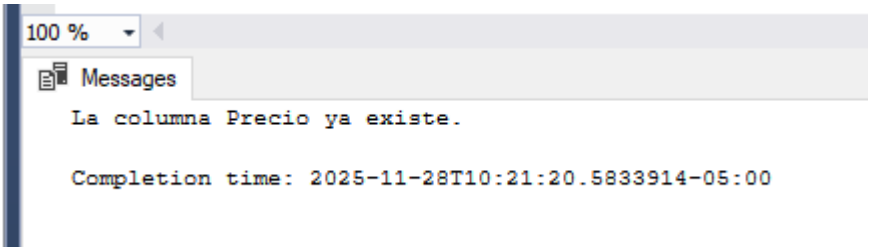
```
IF NOT EXISTS (
    SELECT 1
    FROM sys.stats
    WHERE name = 'ST_Productos_Precio'
      AND object_id = OBJECT_ID('Productos')
)
BEGIN
    CREATE STATISTICS ST_Productos_Precio
    ON Productos (Precio);

    PRINT 'Estadística manual ST_Productos_Precio creada.';
END
ELSE
    PRINT 'La estadística ST_Productos_Precio ya existe.';
```



Validar

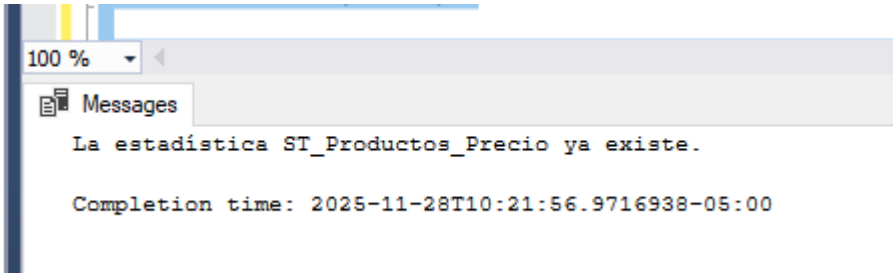
Verificar/crear la columna Precio

```
-- 1. Verificar si la columna Precio existe
IF COL_LENGTH('Productos', 'Precio') IS NULL
BEGIN
    ALTER TABLE Productos
    ADD Precio DECIMAL(10,2);

    PRINT 'Columna Precio agregada correctamente.';
END
ELSE
BEGIN
    PRINT 'La columna Precio ya existe.';
END
```



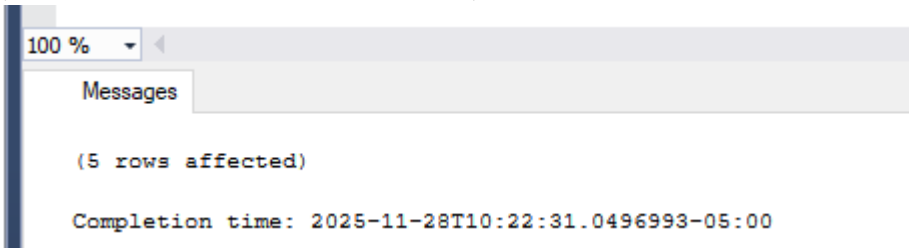
Crear la estadística manual ST_Productos_Precio

```
-- 2. Crear estadística manual sobre la columna Precio
IF NOT EXISTS (
    SELECT 1
    FROM sys.stats
    WHERE name = 'ST_Productos_Precio'
      AND object_id = OBJECT_ID('Productos')
)
BEGIN
    CREATE STATISTICS ST_Productos_Precio
    ON Productos (Precio);

    PRINT 'Estadística ST_Productos_Precio creada correctamente.';
END
ELSE
BEGIN
    PRINT 'La estadística ST_Productos_Precio ya existe.';
END
```



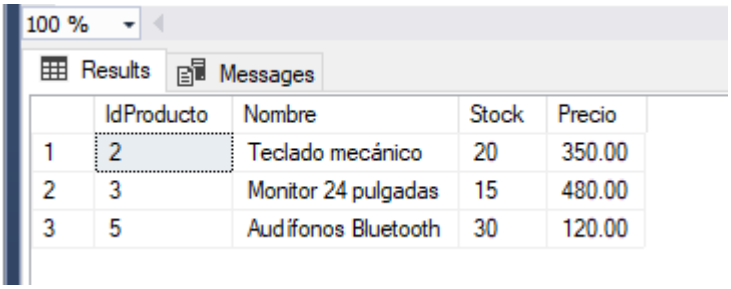
Insertar datos de prueba

```
INSERT INTO Productos (Nombre, Stock, Precio)
VALUES
('Mouse inalámbrico', 50, 60.00),
('Teclado mecánico', 20, 350.00),
('Monitor 24 pulgadas', 15, 480.00),
('Laptop Lenovo', 10, 2500.00),
('Audífonos Bluetooth', 30, 120.00);
```



Ejecutar una consulta que use la estadística

```
SELECT *
FROM Productos
WHERE Precio BETWEEN 100 AND 500;
```



	IdProducto	Nombre	Stock	Precio
1	2	Teclado mecánico	20	350.00
2	3	Monitor 24 pulgadas	15	480.00
3	5	Audífonos Bluetooth	30	120.00

JUSTIFICACIÓN TÉCNICA DE LA SOLUCIÓN

✓ Las estadísticas le permiten al optimizador de consultas estimar la cantidad de filas que devolverá un predicado (ej. Precio BETWEEN 100 AND 500).

✓ SQL Server automáticamente crea estadísticas, pero solo cuando:

Se consulta una columna no indexada por primera vez

El motor lo considera necesario

BUENAS PRÁCTICAS

✓ 1. Usar CREATE STATISTICS cuando hay filtros por rangos

Columnas como:

Precio

Edad

Fecha

Cantidad

Monto

se benefician muchísimo.

✓ 2. No crear demasiadas estadísticas innecesarias
Cada estadística:
Consume CPU al actualizarse
Requiere mantenimiento
Solo crear cuando aportan valor.

Proyecto 9: Configuración de un Resource Pool para limitar recursos

1. Enunciado del ejercicio

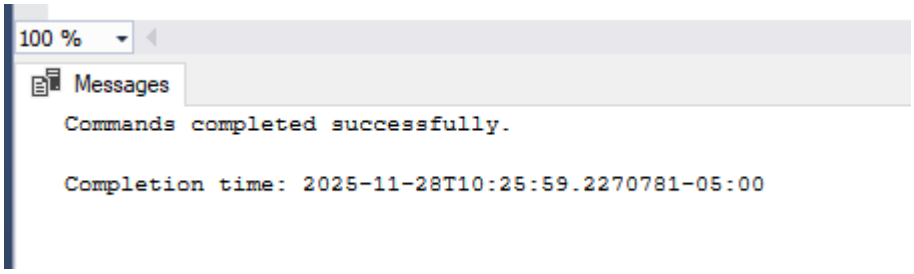
   Crear un Resource Pool que limite el uso de CPU al 20% para consultas analíticas pesadas.

proyecto 9: configuración de un resource pool para limitar recursos
enunciado del ejercicio
Crear un resource pool que limite el uso de CPU al 20% para consultas analíticas pesadas.
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

script de la solución en SQL

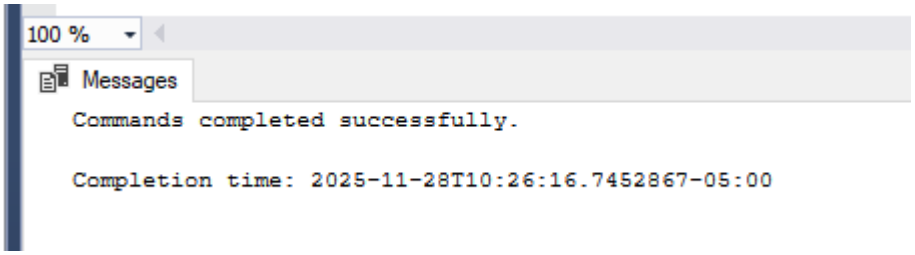
Habilitar Resource Governor

```
ALTER RESOURCE GOVERNOR RECONFIGURE;
```



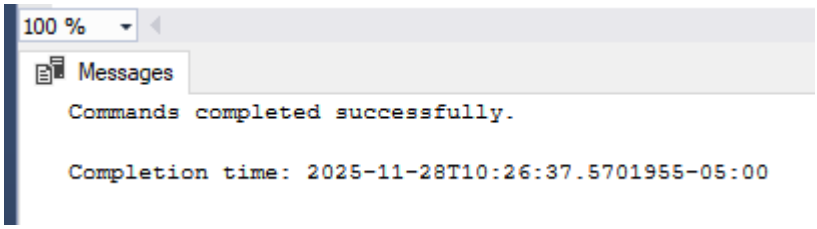
Crear el Resource Pool (máximo 20% CPU)

```
CREATE RESOURCE POOL PoolAnalitico
WITH (
    MAX_CPU_PERCENT = 20,        -- No podrá usar más del 20% de CPU
    MIN_CPU_PERCENT = 0          -- No reservamos CPU, solo limitamos
);
```



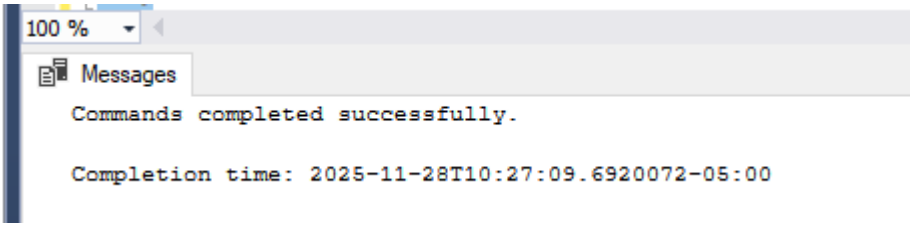
Crear un Workload Group dentro del Pool

```
CREATE WORKLOAD GROUP GrupoAnalitico
USING PoolAnalitico;
```



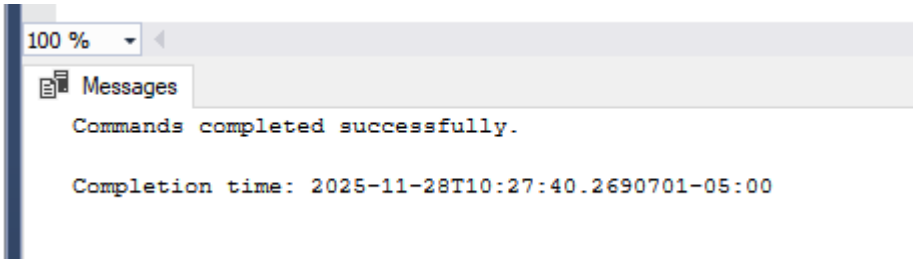
Crear la función clasificadora

```
CREATE FUNCTION dbo.fnClasificarSesion()
RETURNS sysname
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @grupo SYSNAME;

    -- Si la sesión marca contexto analítico, la enviamos al pool
    IF (CONTEXT_INFO() = 0xA1A1A1)
        SET @grupo = 'GrupoAnalitico';
    ELSE
        SET @grupo = 'default';

    RETURN @grupo;
END;
```



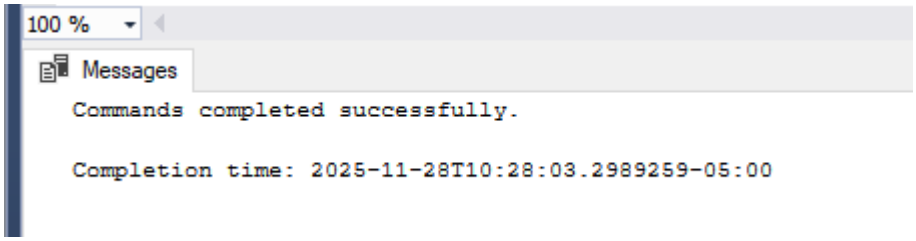
Activar el Resource Governor

```
ALTER RESOUCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = dbo.fnClasificarSesion);

ALTER RESOURCE GOVERNOR RECONFIGURE;
```



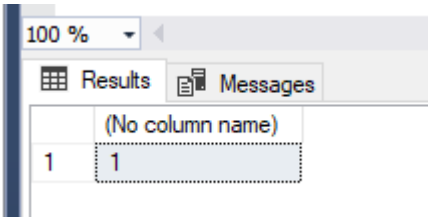
Marcar la sesión como analítica

```
DECLARE @x VARBINARY(128) = 0xA1A1A1;
SET CONTEXT_INFO @x;
```



Ejecutar tu consulta pesada

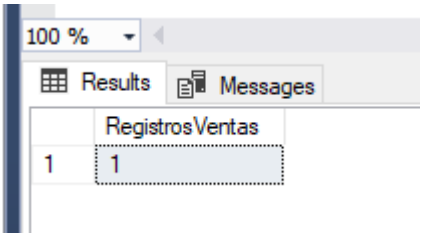
```
SELECT COUNT(*)
FROM Ventas v1
CROSS JOIN Ventas v2
CROSS JOIN Ventas v3;
```



Validar en Sql

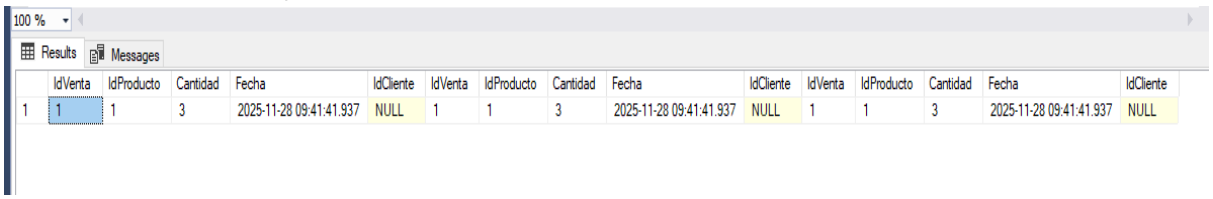
Verifica cuántos registros tiene la tabla Ventas

```
SELECT COUNT(*) AS RegistrosVentas
FROM Ventas;
```



Valida si el CROSS JOIN

```
SELECT TOP 10 *
FROM Ventas v1
CROSS JOIN Ventas v2
CROSS JOIN Ventas v3;
```



JUSTIFICACIÓN TÉCNICA

SQL Server Resource Governor permite controlar:

CPU

Memoria

IO

Separando cargas de trabajo (OLTP, analítica, cargas nocturnas, reportes, etc.).

✓ Limitar consultas analíticas a 20% CPU evita que bloqueen consultas transaccionales críticas.

✓ Se usa un Resource Pool para definir límites.
✓ Se usa un Workload Group para agrupar sesiones.
✓ Se usa una función clasificadora para decidir qué sesión va dónde.
✓ RECONFIGURE aplica los cambios sin reiniciar el servidor.
Este enfoque da control total sobre la asignación de recursos.

📓 3. BUENAS PRÁCTICAS

✓ 1. No limitar arbitrariamente pools críticos
Nunca limites:
default
interno
✓ 2. Usar función clasificadora simple
Reglas complicadas → degradan el rendimiento.
✓ 3. Identificar sesiones por:
CONTEXT_INFO
Aplicación (program_name)
Login (suser_name())
Usuario de base
✓ 4. Auditoría constante
Consultar uso de pools:

```
SELECT *
FROM sys.dm_resource_governor_resource_pools;
```

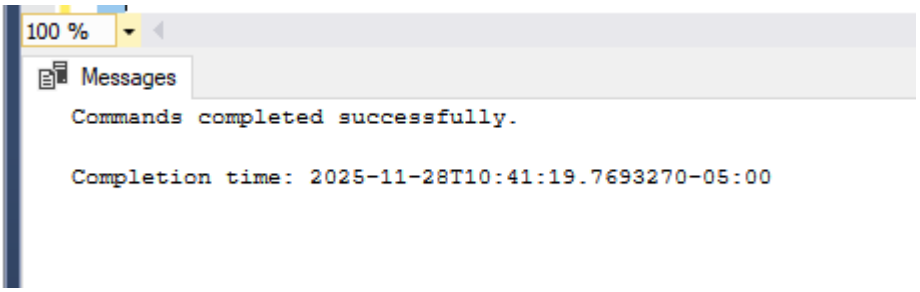
proyecto 10: crear un trigger de auditoria ligera usando extended events
enunciado del ejercicio
Auditar inserciones en la tabla productos, usando extended events sin afectar rendimiento.
-script de la solución en SQL
-Justificación técnica de la solución aplicada
-Explicación de buenas prácticas

Script de la solución en SQL

crear la sesión xe

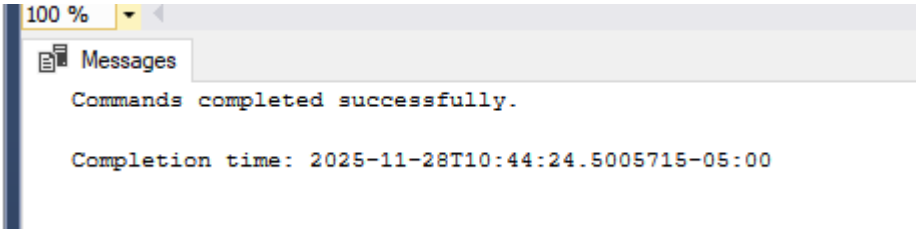
```
CREATE EVENT SESSION Auditoria_Productos_Insert
ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION
```

```
    (
        sqlserver.client_app_name,
        sqlserver.client_hostname,
        sqlserver.server_principal_name,
        sqlserver.sql_text
    )
    WHERE
    (
         sqlserver.sql_text LIKE '%INSERT INTO Productos%'
    )
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\XE_Auditoria\Auditoria_Productos_Insert.xel'
)
WITH
(
    STARTUP_STATE = ON
);
GO
ALTER EVENT SESSION Auditoria_Productos_Insert
ON SERVER STATE = START;
```



versión con filtro por base de datos

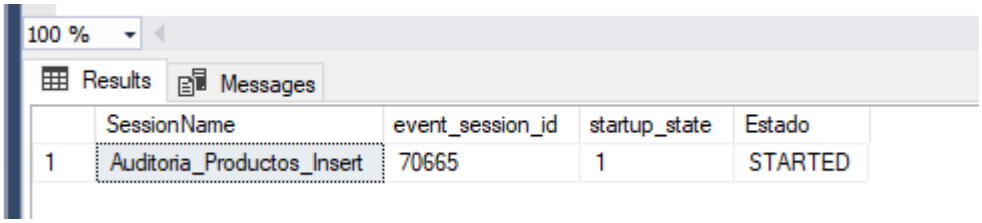
```
CREATE EVENT SESSION Auditoria_Productos_Insert_2
ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION
    (
        sqlserver.client_app_name,
        sqlserver.client_hostname,
        sqlserver.server_principal_name,
        sqlserver.sql_text
    )
    WHERE
    (
         sqlserver.database_name = 'QhatuPeru'
         AND sqlserver.sql_text LIKE '%INSERT INTO Productos%'
    )
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\XE_Auditoria\Auditoria_Productos_Insert_2.xel'
)
WITH
(
    STARTUP_STATE = ON
);
GO
ALTER EVENT SESSION Auditoria_Productos_Insert_2 ON SERVER STATE = START;
GO
```



VALIDAR EN SQL

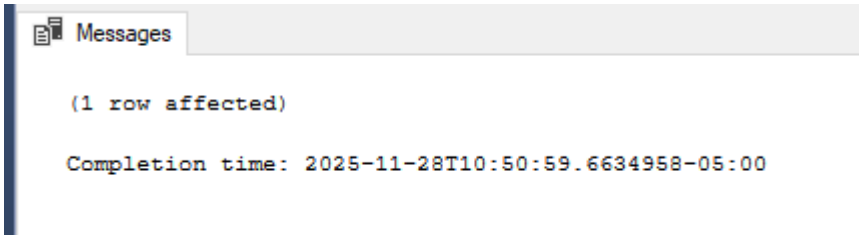
Verificar si la sesión existe

```
-- Verificar si la sesión existe y su estado
SELECT
    s.name AS SessionName,
    s.event_session_id,
    s.startup_state,
    CASE
        WHEN s.event_session_id IN (SELECT event_session_id FROM sys.dm_xe_sessions)
THEN 'STARTED'
        ELSE 'STOPPED'
    END AS Estado
FROM sys.server_event_sessions s
WHERE s.name = 'Auditoria_Productos_Insert';
```



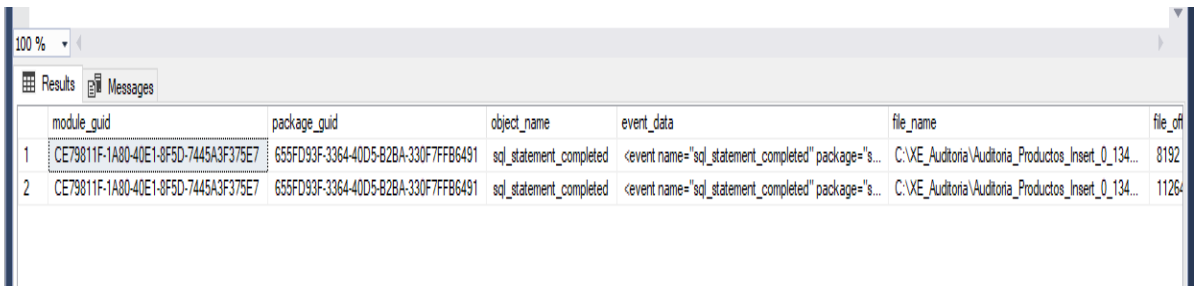
Inserta un registro de prueba en Productos para validar la auditoría:

```
INSERT INTO Productos (Nombre, Stock, Precio)
VALUES ('Producto Test', 1, 99.99);
```



Consulta los eventos capturados en el archivo .xel usando:

```
SELECT *
FROM sys.fn_xe_file_target_read_file(
    'C:\XE_Auditoria\Auditoria_Productos_Insert*.xel', NULL, NULL, NULL
);
```



Justificación Técnica de la Solución

✓ 1. No se usa TRIGGER para auditoría
Los triggers:
Aumentan la latencia de la transacción
Se ejecutan dentro del proceso de INSERT
Consumen CPU y bloqueos
Son difíciles de mantener
Por eso, para auditoría real en producción no se usan triggers, sino Extended Events.

✓ 2. Extended Events es más rápido y eficiente
Extended Events:
Usa buffers en memoria
No impacta la transacción que hace el INSERT
Es asincrónico
Consume menos CPU que SQL Profiler
Puede encenderse y apagarse sin reiniciar servicios
Por eso es la forma moderna y recomendada de auditar.

✓ 3. Filtrado por texto 'INSERT INTO Productos'
Esto permite:
Capturar solo inserciones reales
Reducir la carga de eventos
Evitar ruido innecesario

✓ 4. Se almacenan los datos en archivo .xel
Porque:
No bloquea
No usa tablas
Es ligero
Es fácil de consultar

📌 Buenas Prácticas Aplicadas

✓ 1. No guardar auditoría en tablas
Guardar auditoría en tablas internas causa:
Bloqueos
Crecimiento incontrolado
Lentitud en INSERTs
Por eso se guarda en archivos XE.

✓ 2. Activar la sesión al inicio del servidor
STARTUP_STATE = ON garantiza que la auditoría esté siempre activa.

✓ 3. Registrar solo lo necesario
Solo se captura:
Query ejecutado
Usuario
Hostname
Aplicación
Nada más → mínimo impacto.

✓ 4. Ubicación del archivo fuera de C:\Temp
C:\XE_Auditoria\ es una ruta segura y manejable.