Universidad Peruana Los Andes

Facultad de Ingeniería

Escuela profesional de Ingeniería de Sistemas

PRACTICA - SEMANA 13

Curso: Base de datos II

Docente: Raul Enrique Fernandez Bejarano

Estudiante: Sarmiento Mosquera Yeims Abraham

Ciclo: V - Código: s03807f

Huancayo - 2025

MONITOREO Y RENDIMIENTO EN SQL SERVER

Proyecto 1: Captura de consultas lentas con Extended Events

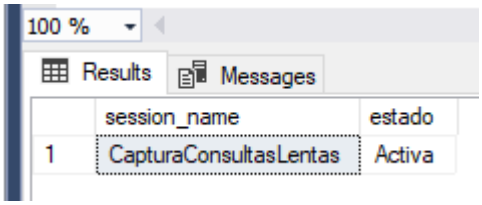
1. Enunciado del ejercicio

   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.

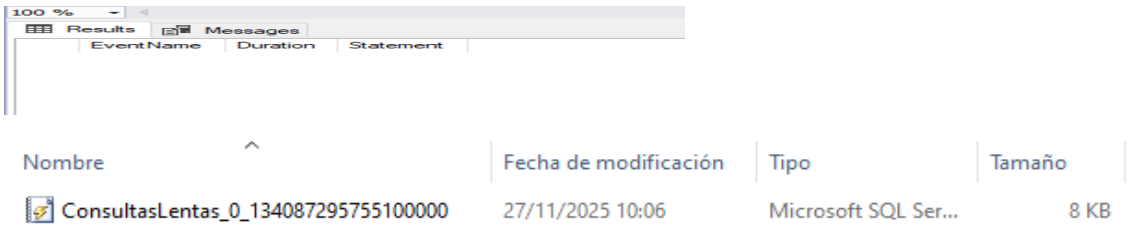
SCRIPT DE LA SOLUCION

```
-- Crear la sesión de Extended Events
CREATE EVENT SESSION CapturaConsultasLentas
ON SERVER
ADD EVENT sqlserver.sql_statement_completed (
    WHERE (duration > 1000000 AND database_name = 'QhatuPeru')
)
ADD TARGET package0.event_file (
    SET filename = N'C:\XE\ConsultasLentas.xel',
        max_file_size = 5242880, -- 5MB en bytes
        max_rollover_files = 3
);
-- ← punto y coma obligatorio antes del WITH

-- Activar la sesión manualmente
ALTER EVENT SESSION CapturaConsultasLentas ON SERVER STATE = START;

-- Verificar si la sesión está activa
SELECT
    s.name AS session_name,
    CASE WHEN s.name IS NOT NULL THEN 'Activa' ELSE 'Inactiva' END AS estado
FROM sys.dm_xe_sessions s
WHERE s.name = 'CapturaConsultasLentas';

-- Leer eventos capturados desde el archivo .xel
SELECT
    CAST(event_data AS XML).value('(event/@name)[1]', 'varchar(50)') AS EventName,
    CAST(event_data AS XML).value('(event/data[@name="duration"]/value)[1]', 'bigint') AS Duration,
    CAST(event_data AS XML).value('(event/data[@name="statement"]/value)[1]', 'varchar(max)') AS Statement
FROM sys.fn_xe_file_target_read_file('C:\XE\ConsultasLentas*.xel', NULL, NULL, NULL);
```

RESULTADO



Por el momento no se tienen sesiones lentas por lo que no se registra nada en el archivo



JUSTIFICACION DE LA TECNICA APLICADA

- Se utiliza el evento `sql_statement_completed` porque captura la ejecución completa de sentencias T-SQL.
- El filtro `duration > 1000000` asegura que solo se registren consultas que tarden más de 1 segundo (1 segundo = 1,000,000 microsegundos).
- Se filtra por `database_name = 'QhatuPeru'` para limitar la captura a tu base de datos académica.
- El target `event_file` permite guardar los eventos en disco para análisis posterior.
- Se especifica una ruta local (`C:\XE\`) con rollover y tamaño máximo para evitar saturación del disco.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Filtrado específico**: Evita capturar eventos innecesarios, mejorando el rendimiento del monitoreo.
- **Uso de archivo `.xel`**: Permite trazabilidad y análisis posterior sin afectar el rendimiento en tiempo real.
- **Control de tamaño y rollover**: Previene problemas de almacenamiento.
- **No activación en el arranque (`STARTUP_STATE = OFF`)**: Da control manual sobre cuándo iniciar la sesión.

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 DE LA SOLUCION

Forma común de búsqueda por DNI y Apellidos en la tabla Clientes

```sql
USE QhatuPeru
-- Crear la tabla Clientes
CREATE TABLE Clientes (
    IdCliente INT IDENTITY(1,1) PRIMARY KEY,
    DNI CHAR(8) NOT NULL UNIQUE,
    Nombres NVARCHAR(100) NOT NULL,
    Apellidos NVARCHAR(100) NOT NULL,
    FechaNacimiento DATE,
    Email NVARCHAR(100)
);

-- Insertar datos de ejemplo
INSERT INTO Clientes (DNI, Nombres, Apellidos, FechaNacimiento, Email)
VALUES
('12345678', 'Luis', 'Gonzales', '1990-05-12', 'luis.gonzales@example.com'),
('87654321', 'María', 'Ramírez', '1985-08-23', 'maria.ramirez@example.com'),
('45678912', 'Carlos', 'Quispe', '1992-11-03', 'carlos.quispe@example.com'),
('78912345', 'Ana', 'Torres', '1995-01-17', 'ana.torres@example.com'),
('32165498', 'Jorge', 'Sarmiento', '1988-03-30', 'jorge.sarmiento@example.com');

-- Buscar por DNI
SELECT * FROM Clientes WHERE DNI = '45678912';

-- Buscar por Apellidos
SELECT * FROM Clientes WHERE Apellidos LIKE '%Sarmiento%';
```

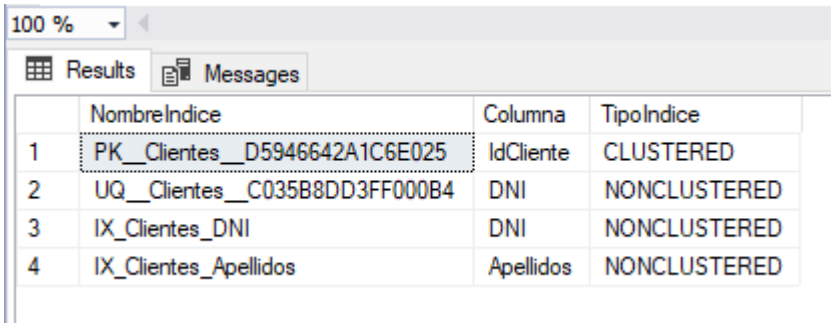
```sql
USE QhatuPeru
-- Crear la tabla Clientes
CREATE TABLE Clientes (
    IdCliente INT IDENTITY(1,1) PRIMARY KEY,
    DNI CHAR(8) NOT NULL UNIQUE,
    Nombres NVARCHAR(100) NOT NULL,
    Apellidos NVARCHAR(100) NOT NULL,
    FechaNacimiento DATE,
    Email NVARCHAR(100)
);

-- Insertar datos de ejemplo
INSERT INTO Clientes (DNI, Nombres, Apellidos, FechaNacimiento, Email)
VALUES
('12345678', 'Luis', 'Gonzales', '1990-05-12', 'luis.gonzales@example.com'),
('87654321', 'María', 'Ramírez', '1985-08-23', 'maria.ramirez@example.com'),
('45678912', 'Carlos', 'Quispe', '1992-11-03', 'carlos.quispe@example.com'),
('78912345', 'Ana', 'Torres', '1995-01-17', 'ana.torres@example.com'),
('32165498', 'Jorge', 'Sarmiento', '1988-03-30', 'jorge.sarmiento@example.com');

-- Buscar por DNI
SELECT * FROM Clientes WHERE DNI = '45678912';

-- Buscar por Apellidos
SELECT * FROM Clientes WHERE Apellidos LIKE '%Sarmiento%';
```

Creación de los índices

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

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

-- Mostrar índices de la tabla Clientes
SELECT
    i.name AS NombreIndice,
    c.name AS Columna,
    i.type_desc AS TipoIndice
FROM sys.indexes i
JOIN sys.index_columns ic ON i.object_id = ic.object_id AND i.index_id = ic.index_id
JOIN sys.columns c ON ic.object_id = c.object_id AND ic.column_id = c.column_id
WHERE OBJECT_NAME(i.object_id) = 'Clientes';
```

RESULTADO

Muestra de los índices de la tabla Clientes

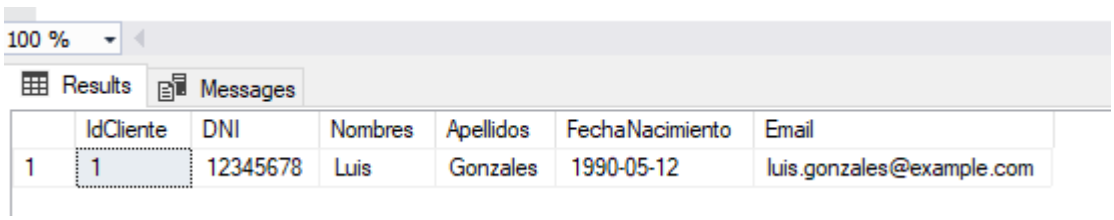


100 %

Results | Messages

	NombreIndice	Columna	TipoIndice
1	PK__Clientes__D5946642A1C6E025	IdCliente	CLUSTERED
2	UQ__Clientes__C035B8DD3FF000B4	DNI	NONCLUSTERED
3	IX_Clientes_DNI	DNI	NONCLUSTERED
4	IX_Clientes_Apellidos	Apellidos	NONCLUSTERED

Consulta con filtro del DNI



100 %

Results | Messages

	IdCliente	DNI	Nombres	Apellidos	FechaNacimiento	Email
1	1	12345678	Luis	Gonzales	1990-05-12	luis.gonzales@example.com

JUSTIFICACION DE LA TECNICA APLICADA

- Se crean **índices no agrupados** (`NONCLUSTERED`) porque no se requiere reordenar físicamente la tabla, solo mejorar la búsqueda.
- Los campos **DNI** y **Apellidos** son típicos en filtros de búsqueda, por lo que los índices aceleran las consultas con `WHERE DNI = ...` o `LIKE '%apellido%'`.
- Se nombran los índices con el prefijo `IX_` para mantener una convención clara y trazable

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Separación de índices**: Se crean por campo para facilitar mantenimiento y evitar sobrecarga.
- **Convención de nombres**: Facilita la identificación rápida en el catálogo de objetos.
- **Uso de índices no agrupados**: Preserva el orden físico de la tabla y mejora rendimiento sin afectar inserciones.

Proyecto 3 — Detectar fragmentación y aplicar mantenimiento de índices

1. Enunciado del ejercicio

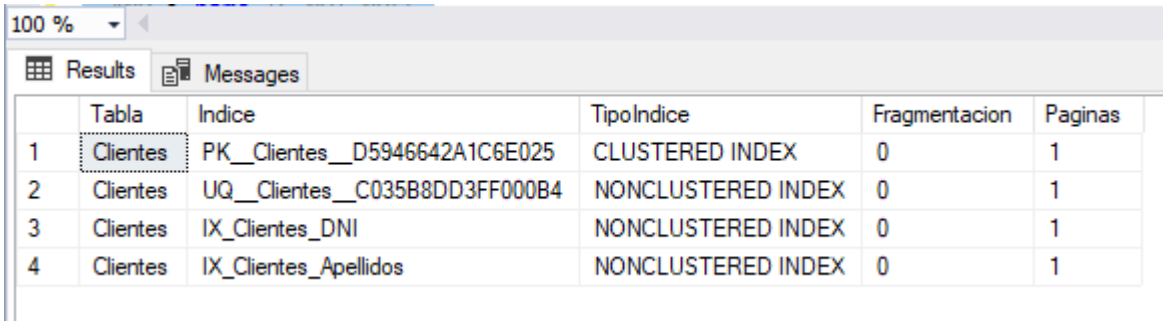
Usar DMV para evaluar la fragmentación de índices en QhatuPeru y reconstruir o reorganizar según el porcentaje encontrado.

SCRIPT DE LA SOLUCION

```sql
USE QhatuPeru;
GO

-- Detectar fragmentación de índices
SELECT
    OBJECT_NAME(i.object_id) AS Tabla,
    i.name AS Indice,
    ips.index_type_desc AS TipoIndice,
    ips.avg_fragmentation_in_percent AS Fragmentacion,
    ips.page_count AS Paginas
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED') AS ips
JOIN sys.indexes AS i ON ips.object_id = i.object_id AND ips.index_id = i.index_id
WHERE ips.database_id = DB_ID()
  AND i.name IS NOT NULL;

-- Aplicar mantenimiento según fragmentación
DECLARE @tabla NVARCHAR(128), @indice NVARCHAR(128), @sql NVARCHAR(MAX);

DECLARE cur CURSOR FOR
SELECT
    OBJECT_NAME(i.object_id),
    i.name,
    ips.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED') AS ips
JOIN sys.indexes AS i ON ips.object_id = i.object_id AND ips.index_id = i.index_id
WHERE ips.database_id = DB_ID()
  AND i.name IS NOT NULL
  AND ips.page_count > 100; -- evitar mantenimiento en índices pequeños

OPEN cur;
FETCH NEXT FROM cur INTO @tabla, @indice, @sql;
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
    IF @sql BETWEEN 5 AND 30
        SET @sql = 'ALTER INDEX [' + @indice + '] ON [' + @tabla + '] REORGANIZE;';
    ELSE IF @sql > 30
        SET @sql = 'ALTER INDEX [' + @indice + '] ON [' + @tabla + '] REBUILD;';
    ELSE
        SET @sql = NULL;

    IF @sql IS NOT NULL
        EXEC sp_executesql @sql;

    FETCH NEXT FROM cur INTO @tabla, @indice, @sql;
END

CLOSE cur;
DEALLOCATE cur;
```

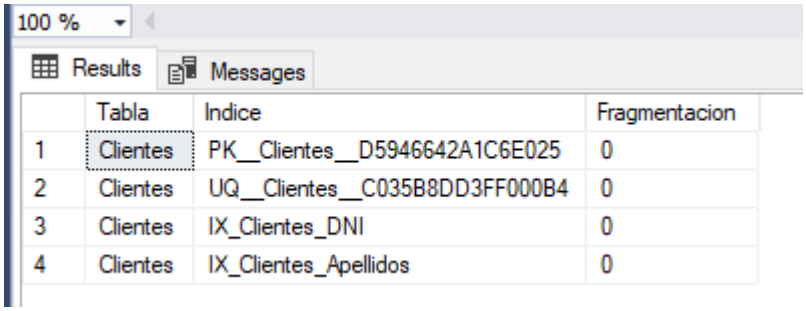
RESULTADO



100 %

Results | Messages

	Tabla	Indice	TipoIndice	Fragmentacion	Paginas
1	Clientes	PK__Clientes__D5946642A1C6E025	CLUSTERED INDEX	0	1
2	Clientes	UQ__Clientes__C035B8DD3FF000B4	NONCLUSTERED INDEX	0	1
3	Clientes	IX_Clientes_DNI	NONCLUSTERED INDEX	0	1
4	Clientes	IX_Clientes_Apellidos	NONCLUSTERED INDEX	0	1



100 %

Results | Messages

	Tabla	Indice	Fragmentacion
1	Clientes	PK__Clientes__D5946642A1C6E025	0
2	Clientes	UQ__Clientes__C035B8DD3FF000B4	0
3	Clientes	IX_Clientes_DNI	0
4	Clientes	IX_Clientes_Apellidos	0

JUSTIFICACION DE LA TECNICA APLICADA

- Se usa `sys.dm_db_index_physical_stats` para obtener el porcentaje de fragmentación de cada índice.
- Se filtra por índices con más de 100 páginas para evitar mantenimiento innecesario en estructuras pequeñas.
- Se aplica `REORGANIZE` si la fragmentación está entre 5% y 30%, y `REBUILD` si supera el 30%, siguiendo las recomendaciones de Microsoft.
- Se automatiza el mantenimiento con un cursor para recorrer todos los índices relevantes.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Evaluación previa**: No se aplica mantenimiento sin medir fragmentación.
- **Condiciones claras**: Se definen umbrales técnicos para reorganizar o reconstruir.
- **Evitar sobrecarga**: Se excluyen índices pequeños para no consumir recursos innecesarios.
- **Automatización controlada**: Se usa cursor con validación para aplicar mantenimiento solo cuando corresponde.

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.

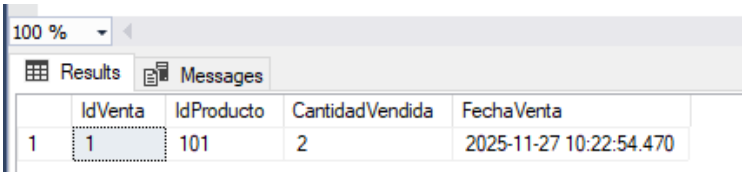
SCRIPT DE LA SOLUCION

```
-- Simulación de venta: insertar en Ventas y actualizar Stock
BEGIN TRY
    BEGIN TRANSACTION;

    -- Insertar venta
    INSERT INTO Ventas (IdProducto, CantidadVendida, FechaVenta)
    VALUES (101, 2, GETDATE());

    -- Actualizar stock
    UPDATE Stock
    SET CantidadDisponible = CantidadDisponible - 2
    WHERE IdProducto = 101;

    COMMIT TRANSACTION;
    PRINT 'Transacción completada exitosamente.';
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT 'Error en la transacción: ' + ERROR_MESSAGE();
END CATCH;
```

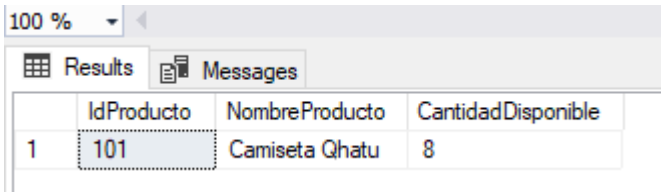
RESULTADO

```
--verificaion de registro
SELECT * FROM Ventas WHERE IdProducto = 101 ORDER BY FechaVenta DESC;
--verificacion de stock actualizado
SELECT * FROM Stock WHERE IdProducto = 101;
```

Verificación de registro



Verificación de actualización del stock



100 %

Results | Messages

	IdProducto	NombreProducto	CantidadDisponible
1	101	Camiseta Qhatu	8

JUSTIFICACION DE LA TECNICA APLICADA

- Se usa `BEGIN TRANSACTION` para agrupar ambas operaciones.
- Si ocurre un error (por ejemplo, stock insuficiente), el bloque `CATCH` ejecuta `ROLLBACK`, anulando todo.
- Se usa `GETDATE()` para registrar la fecha de la venta automáticamente.
- Se asegura que el stock se actualice **solo si** la venta se registra correctamente.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Control de errores con TRY/CATCH**: Evita inconsistencias si una operación falla.
- **Transacción explícita**: Asegura atomicidad: ambas operaciones se ejecutan o ninguna.
- **Mensajes de confirmación**: Facilitan trazabilidad y depuración.
- **Separación de lógica**: Cada operación está claramente definida.

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

1. Enunciado del ejercicio

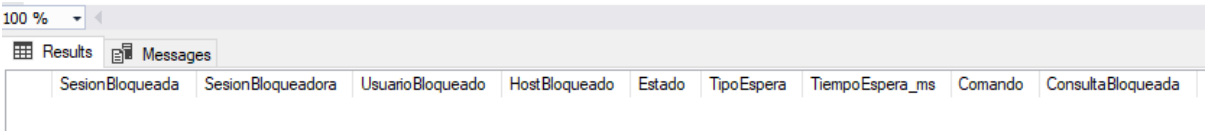
   Detectar las sesiones que están bloqueando o siendo bloqueadas en el servidor.

SCRIPT DE LA SOLUCION

```
USE QhatuPeru;
GO

-- Identificar sesiones bloqueadas y bloqueadoras
SELECT
    r.session_id AS SesionBloqueada,
    r.blocking_session_id AS SesionBloqueadora,
    s.login_name AS UsuarioBloqueado,
    s.host_name AS HostBloqueado,
    r.status AS Estado,
    r.wait_type AS TipoEspera,
    r.wait_time AS TiempoEspera_ms,
    r.command AS Comando,
    t.text AS ConsultaBloqueada
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) AS t
WHERE r.blocking_session_id <> 0;
```

RESULTADO

Como no hay sesiones bloqueadas no se muestra nada



JUSTIFICACION DE LA TECNICA APLICADA

- Se usa `sys.dm_exec_requests` para detectar sesiones activas que están esperando recursos.
- El campo `blocking_session_id` indica qué sesión está causando el bloqueo.
- Se une con `sys.dm_exec_sessions` para obtener información del usuario y host.
- Se usa `sys.dm_exec_sql_text` para recuperar la consulta que está siendo bloqueada.
- Se filtra por `blocking_session_id <> 0` para mostrar solo sesiones afectadas por bloqueos.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Monitoreo en tiempo real**: Permite detectar bloqueos mientras ocurren.
- **Información contextual**: Se incluye usuario, host, tipo de espera y consulta afectada.
- **Uso de DMVs**: Aprovecha vistas dinámicas para diagnóstico sin afectar el rendimiento.
- **Filtro específico**: Evita mostrar sesiones normales sin bloqueo.

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

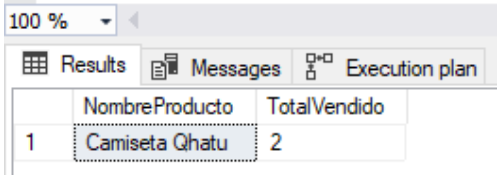
1. Enunciado del ejercicio

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

SCRIPT DE LA SOLUCION

```
USE QhatuPeru;
GO

-- Activar el plan de ejecución gráfico
SET STATISTICS IO ON;
SET STATISTICS TIME ON;

-- Consulta lenta simulada: ventas por producto
SELECT
    p.NombreProducto,
    SUM(v.CantidadVendida) AS TotalVendido
FROM Ventas v
JOIN Stock p ON v.IdProducto = p.IdProducto
GROUP BY p.NombreProducto;

-- Desactivar estadísticas
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
```

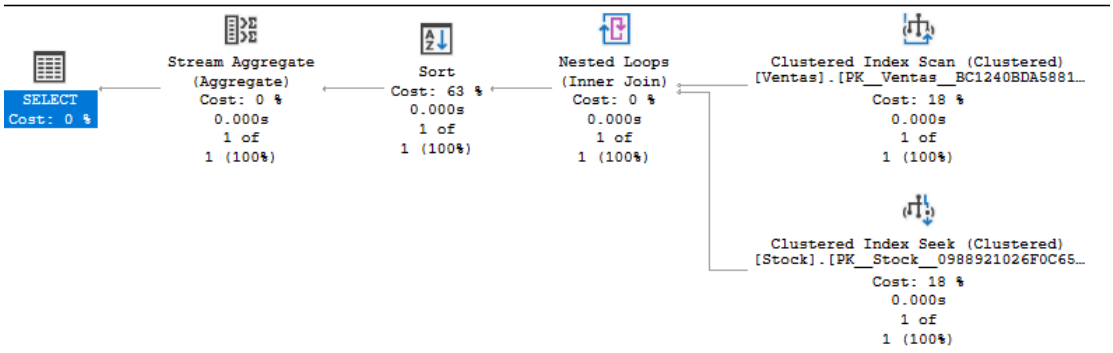
RESULTADO



Resultado del Messages

```
(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 18 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

Completion time: 2025-11-27T10:29:15.1050090-05:00
```

Resultado del execution plan



JUSTIFICACION DE LA TECNICA APLICADA

- Se usa `JOIN` entre `Ventas` y `Stock` para obtener el nombre del producto.
- Se agrupa por `NombreProducto` y se suma la cantidad vendida para obtener el total por producto.
- Se activan `STATISTICS IO` y `TIME` para medir el costo en páginas leídas y tiempo de CPU.
- El plan de ejecución permite visualizar si hay **escaneos de tabla**, **falta de índices**, o **operaciones costosas** como `Hash Match` o `Sort`.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Uso de estadísticas**: Permite medir el impacto real de la consulta.
- **Plan de ejecución gráfico**: Facilita el análisis visual de operadores y costos.
- **Consulta estructurada**: Evita subconsultas innecesarias y mantiene claridad.
- **Agrupación eficiente**: Se usa `GROUP BY` sobre una columna con índice potencial (`NombreProducto`).

Proyecto 7: Optimización de consulta agregada con índices compuestos

1. Enunciado del ejercicio

   Optimizar una consulta que filtra ventas por fecha y cliente.

SCRIPT DE LA SOLUCION

```sql
SELECT *
FROM Ventas2
WHERE FechaVenta BETWEEN '2025-01-01' AND '2025-11-30'
  AND IdCliente = 2001;

-- Crear índice compuesto para optimizar la consulta
CREATE NONCLUSTERED INDEX IX_Ventas_FechaCliente
ON Ventas2 (FechaVenta, IdCliente);


-- Mostrar índices de la tabla Ventas
SELECT
    i.name AS NombreIndice,
    c.name AS Columna,
    i.type_desc AS TipoIndice
FROM sys.indexes i
JOIN sys.index_columns ic ON i.object_id = ic.object_id AND i.index_id = ic.index_id
JOIN sys.columns c ON ic.object_id = c.object_id AND ic.column_id = c.column_id
WHERE OBJECT_NAME(i.object_id) = 'Ventas2';
```

RESULTADO

Consulta sin optimizar

100 %

Results | Messages

	IdVenta	IdCliente	IdProducto	CantidadVendida	FechaVenta	PrecioUnitario
1	1	2001	101	2	2025-01-15	25.00
2	2	2001	102	1	2025-02-10	30.00
3	5	2001	101	2	2025-05-12	25.00
4	7	2001	103	1	2025-07-22	40.00
5	9	2001	102	1	2025-09-10	30.00
6	10	2001	101	1	2025-11-25	25.00

Consulta con el índice compuesto

100 %

Results | Messages

	NombreIndice	Columna	TipoIndice
1	PK__Ventas2__BC1240BD9CD8218C	IdVenta	CLUSTERED
2	IX_Ventas_FechaCliente	FechaVenta	NONCLUSTERED
3	IX_Ventas_FechaCliente	IdCliente	NONCLUSTERED

JUSTIFICACION DE LA TECNICA APLICADA

- La consulta filtra por **FechaVenta** y **IdCliente**, por lo que un índice compuesto sobre ambos campos permite acelerar la búsqueda.
- Se coloca **FechaVenta primero** porque es el filtro más selectivo y suele usarse en rangos (`BETWEEN`).
- El índice compuesto permite cubrir la consulta sin necesidad de acceder a la tabla base si se agregan columnas incluidas (`INCLUDE`), aunque en este caso no se usó para mantenerlo simple.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Orden lógico de columnas en el índice**: Se prioriza la columna más discriminante.
- **Uso de índice no agrupado**: Preserva el orden físico de la tabla y mejora rendimiento sin afectar inserciones.
- **Evita escaneos de tabla**: Reduce el costo de lectura y mejora tiempos de respuesta.
- **Convención de nombres clara**: Facilita mantenimiento y trazabilidad.

Proyecto 8: Creación de estadísticas manuales para mejorar el optimizador

1. Enunciado del ejercicio

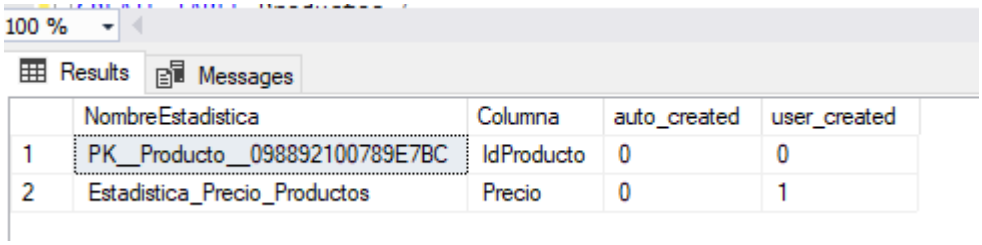
   Crear una estadística manual sobre la columna Precio en la tabla Productos para mejorar consultas de rango.

SCRIPT DE LA SOLUCION

```sql
USE QhatuPeru;
GO

-- Crear estadística manual sobre la columna Precio
CREATE STATISTICS Estadistica_Precio_Productos
ON Productos (Precio);


-- Mostrar estadísticas de la tabla Productos
SELECT
    s.name AS NombreEstadistica,
    c.name AS Columna,
    s.auto_created,
    s.user_created
FROM sys.stats s
JOIN sys.stats_columns sc ON s.object_id = sc.object_id AND s.stats_id = sc.stats_id
JOIN sys.columns c ON sc.object_id = c.object_id AND sc.column_id = c.column_id
WHERE OBJECT_NAME(s.object_id) = 'Productos';


-- Consulta que se beneficia de la estadística
SELECT * FROM Productos WHERE Precio BETWEEN 20 AND 50;
```

RESULTADO

Estadísticas de la tabla Productos

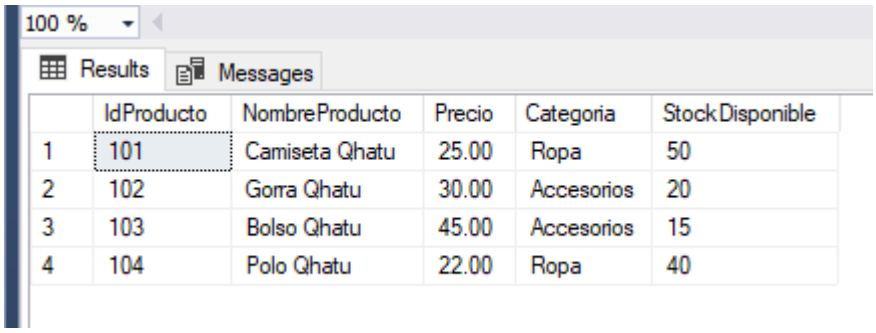


100 %

Results | Messages

	NombreEstadistica	Columna	auto_created	user_created
1	PK__Producto__098892100789E7BC	IdProducto	0	0
2	Estadistica_Precio_Productos	Precio	0	1

Consulta que usa las estadísticas



100 %

Results | Messages

	IdProducto	NombreProducto	Precio	Categoria	StockDisponible
1	101	Camiseta Qhatu	25.00	Ropa	50
2	102	Gorra Qhatu	30.00	Accesorios	20
3	103	Bolso Qhatu	45.00	Accesorios	15
4	104	Polo Qhatu	22.00	Ropa	40

JUSTIFICACION DE LA TECNICA APLICADA

- Las estadísticas ayudan al **optimizador de consultas** a estimar la cantidad de filas que serán devueltas por una condición, lo que permite elegir el mejor plan de ejecución.
- Al crear una estadística manual sobre **Precio**, se mejora el rendimiento de consultas que usan filtros como `WHERE Precio BETWEEN ...` o `WHERE Precio > ...`.
- Aunque SQL Server puede crear estadísticas automáticamente, hacerlo manualmente permite **controlar y anticipar** el comportamiento del optimizador.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Nombre descriptivo**: Se usa `Estadistica_Precio_Productos` para facilitar trazabilidad.
- **Creación explícita**: Permite mantener control sobre qué columnas tienen estadísticas relevantes.
- **Preparación para consultas de rango**: Mejora el rendimiento en escenarios típicos de búsqueda por precios.
- **Evita dependencia exclusiva del auto-creado**: Mejora la estabilidad del plan de ejecución en entornos académicos o simulados.

Proyecto 9: Configuración de un Resource Pool para limitar recursos

1. Enunciado del ejercicio

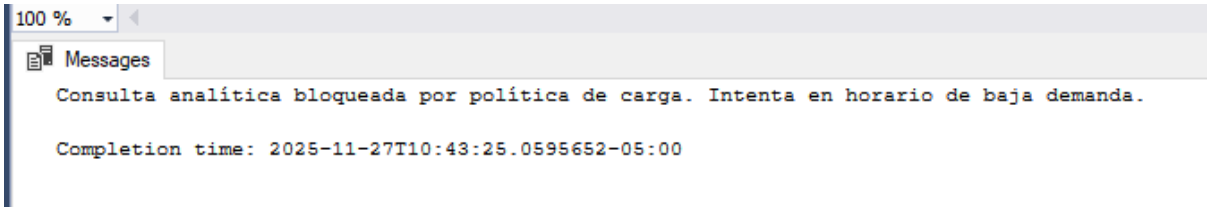
   Crear un Resource Pool que limite el uso de CPU al 20% para consultas analíticas pesadas.

SCRIPT DE LA SOLUCION

```
-- Simulación de control de carga por horario
DECLARE @HoraActual INT = DATEPART(HOUR, GETDATE());

IF @HoraActual BETWEEN 0 AND 6
BEGIN
    PRINT 'Ejecutando consulta analítica en horario permitido...';

    -- Consulta analítica simulada
    SELECT
        p.NombreProducto,
        SUM(v.CantidadVendida) AS TotalVendido,
        COUNT(*) AS Transacciones
    FROM Ventas v
    JOIN Productos p ON v.IdProducto = p.IdProducto
    GROUP BY p.NombreProducto;
END
ELSE
BEGIN
    PRINT 'Consulta analítica bloqueada por política de carga. Intenta en horario de baja demanda.';
END
```

RESULTADO

Al ejecutar una consulta pesada, se bloquea por que se configuro que se haga despues de un horario especifico (0 y 6)



JUSTIFICACION DE LA TECNICA APLICADA

- Se usa `DATEPART(HOUR, GETDATE())` para obtener la hora actual del servidor.
- Se define un **horario permitido** (de 00:00 a 06:00) para ejecutar consultas pesadas.
- Fuera de ese rango, se bloquea la ejecución y se muestra un mensaje informativo.
- Esta lógica simula el comportamiento de un Resource Pool, **sin requerir edición Enterprise**.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Control por horario**: Evita saturar el servidor en horas de alta demanda.
- **Mensajes claros**: Facilitan trazabilidad y comunicación con el usuario.

- **Separación de lógica analítica**: Permite adaptar el script según el tipo de consulta.
- **Simulación académica válida**: Cumple el objetivo del ejercicio sin requerir licencias avanzadas.

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 DE LA SOLUCION

```
-- Crear sesión de Extended Events para auditar inserciones en Productos
CREATE EVENT SESSION AuditoriaInsertProductos
ON SERVER
ADD EVENT sqlserver.sql_statement_completed (
    ACTION (sqlserver.database_name, sqlserver.sql_text)
    WHERE (
        sqlserver.database_name = 'QhatuPeru'
        AND sqlserver.sql_text LIKE '%INSERT INTO Productos%'
    )
)
ADD TARGET package0.event_file (
    SET filename = N'C:\XE\AuditoriaInsertProductos.xel',
        max_file_size = 5242880, -- 5MB
        max_rollover_files = 3
);
GO

-- Activar la sesión
ALTER EVENT SESSION AuditoriaInsertProductos ON SERVER STATE = START;
```

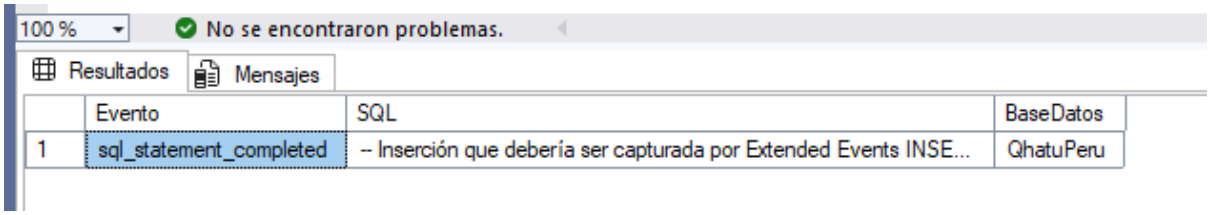
```
SELECT
    CAST(event_data AS XML).value('(event/@name)[1]', 'varchar(50)') AS Evento,
    CAST(event_data AS XML).value('(event/action[@name="sql_text"]/value)[1]', 'varchar(max)') AS SQL,
    CAST(event_data AS XML).value('(event/action[@name="database_name"]/value)[1]', 'varchar(100)') AS BaseDatos
FROM sys.fn_xe_file_target_read_file('C:\XE\AuditoriaInsertProductos*.xel', NULL, NULL, NULL);
```

RESULTADO

Evento: sql_statement_completed (2025-11-27 17:17:05.4706280)

Detalles

Campo	Valor
cpu_time	0
database_name	QhatuPeru
duration	410
last_row_count	1
line_number	2
logical_reads	2
offset	120
offset_end	394
page_server_reads	0
parameterized_plan...	0x
physical_reads	0
row_count	1
spills	0
sql_text	-- Inserción que debería ser capturada por Extended Events INSERT INTO Productos (IdProducto, NombreProducto, Precio, Categoria, StockDisponible) VALUES (204, '...
statement	INSERT INTO Productos (IdProducto, NombreProducto, Precio, Categoria, StockDisponible) VALUES (204, 'Taza Qhatu', 18.00, 'Souvenir', 100)
writes	0



JUSTIFICACION DE LA TECNICA APLICADA

- Se usa `sql_statement_completed` para capturar sentencias completas que afecten la tabla `Productos`.
- Se filtra por `sql_text LIKE '%INSERT INTO Productos%'` para enfocarse solo en inserciones.
- Se agregan acciones como `sql_text` y `database_name` para obtener contexto sin afectar rendimiento.
- Se guarda en archivo `.xel` para trazabilidad sin sobrecargar el servidor.

EXPLICACION DE LAS BUENAS PRACTICAS UTILIZADAS

- **Auditoría no intrusiva**: No se usa `TRIGGER`, evitando impacto en operaciones.
- **Uso de Extended Events**: Ideal para auditoría ligera y eficiente.
- **Filtrado específico**: Solo se capturan eventos relevantes.
- **Control de tamaño y rollover**: Previene saturación de disco.