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

UNIVERSIDAD PERUANA LOS ANDES

“FACULTAD DE INGENIERÍA”

ESCUELA PROFESIONAL “SISTEMAS Y COMPUTACIÓN”

Práctica Semana 13

CÁTEDRA: Base de Datos II

CATEDRÁTICO: Ing. Fernandez Bejarano Raul Enrique

ESTUDIANTE: Quispe Segama Franklin Noe

CICLO: V

SECCIÓN: A1

HUANCAYO PERÚ

2025

Proyecto 1: Captura de consultas lentas con Extended Events

Enunciado del ejercicio

Crear una sesión de Extended Events que capture las consultas cuyo tiempo de ejecución supere 1 segundo (1000 ms) en la base de datos **QhatuPeru**, guardando los eventos generados en un archivo .xel.

Script de la solución en T-SQL

SQLQuery1.sql - LA...PC02\USER 17 (55))*

```
--Proyecto 1
-- Crear sesión Extended Events para capturar consultas lentas (+1 segundo)
CREATE EVENT SESSION CapturaConsultasLentas
ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION (sqlserver.sql_text, sqlserver.database_name, sqlserver.client_app_name)
    WHERE (duration > 1000000  -- duración en microsegundos (1 segundo)
           AND sqlserver.database_name = 'QhatuPERU')
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\XE_Files\ConsultasLentas_QhatuPeru.xel',
        max_file_size = 50,
        max_rollover_files = 5
)
WITH
(
    STARTUP_STATE = OFF
);

-- Iniciar sesión
ALTER EVENT SESSION CapturaConsultasLentas ON SERVER STATE = START;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-11-27T09:53:57.9001521-05:00
```

Justificación técnica de la solución aplicada

1. **Uso del evento sql_statement_completed:**
   Este evento captura la finalización de cada sentencia T-SQL, incluyendo su duración exacta, lo cual permite medir con precisión cuándo una consulta

excede el umbral de 1 segundo.

2. **Filtro por duración (duration > 1000000):**
   La duración en Extended Events está en **microsegundos**, por lo tanto:
   - 1 segundo = **1,000,000 microsegundos**.

3. **Filtro por base de datos (sqlserver.database_name = 'QhatuPeru'):**
   Permite aislar únicamente las consultas que afectan a la base de datos objetivo, evitando ruido.

4. **Acciones (ACTION):**
   - Captura el **texto de la consulta** (sql_text)
   - Nombre de la base de datos
   - Aplicación cliente
     Esto provee contexto completo para el análisis posterior.

5. **Target tipo event_file:**
   Permite almacenar los datos en archivos .xel, que pueden abrirse desde SQL Server Management Studio para una revisión detallada del rendimiento.

6. **Rollover de archivos:**
   - max_file_size = 50 MB
   - max_rollover_files = 5
     Evita que los eventos crezcan indefinidamente y consuman demasiado disco.

7. **Inicio manual (STARTUP_STATE = OFF):**
   Se evita que la sesión arranque automáticamente con el servidor, dando control al administrador para activarla solo cuando sea necesario.

Explicación de las buenas prácticas utilizadas en el proyecto

✔ **1. Uso de filtros en la sesión:**
Filtrar por duración y base de datos reduce la carga del servidor y evita capturar eventos innecesarios.

✔ **2. Almacenamiento en archivos externos (.xel):**
Evita saturar memoria y permite un análisis posterior sin impacto sobre el rendimiento del servidor.

✔ **3. Control de crecimiento del log de eventos:**
Con límites de tamaño y rollover se asegura que no se llenará el disco.

✔ **4. Captura solo de información relevante (acciones seleccionadas):**
Capturar demasiados campos aumenta el costo; aquí solo se guardan datos esenciales.

✔ **5. Mantener la sesión apagada por defecto:**
Impide que la sesión consuma recursos en momentos no planificados.

✔ **6. Comentarios en el script:**
Facilitan el mantenimiento y la comprensión por parte de otros administradores.

✔ **7. Uso de rutas definidas:**
Asignar una carpeta como C:\XE_Files\ ayuda a mantener los archivos organizados y seguros.

Proyecto 2: Crear índices para mejorar la búsqueda de clientes

1. Enunciado del ejercicio

En la tabla **Clientes**, mejorar el rendimiento de las búsquedas realizadas por las columnas **DNI** y **Apellidos**, creando índices adecuados que permitan acelerar consultas frecuentes basadas en esos campos.

2. Script de la solución en T-SQL

```sql
--Proyecto 2
Use QhatuPERU;
go
-- Crear índice no clusterizado para búsqueda por DNI
CREATE NONCLUSTERED INDEX IX_Clientes_DNI
ON dbo.Clientes (DNI);

-- Crear índice no clusterizado para búsqueda por Apellidos
CREATE NONCLUSTERED INDEX IX_Clientes_Apellidos
ON dbo.Clientes (Apellidos);

-- Opcional: índice compuesto si existen búsquedas combinadas
-- (por ejemplo, WHERE DNI = '...' AND Apellidos = '...')
CREATE NONCLUSTERED INDEX IX_Clientes_DNI_Apellidos
ON dbo.Clientes (DNI, Apellidos);
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-11-27T10:05:48.1535381-05:00
```

3. Justificación técnica de la solución aplicada

1. **Búsqueda por DNI:**
    - El número de DNI es un valor altamente selectivo (casi único).
    - Un índice no clusterizado sobre esta columna permite localizar rápidamente registros individuales sin escaneo completo de la tabla.
2. **Búsqueda por Apellidos:**
    - Permite acelerar consultas del tipo:

```sql
--Busqueda por apeliidos
SELECT * FROM Clientes WHERE Apellidos = 'Quispe Segama';
```

100 %

Results | Messages

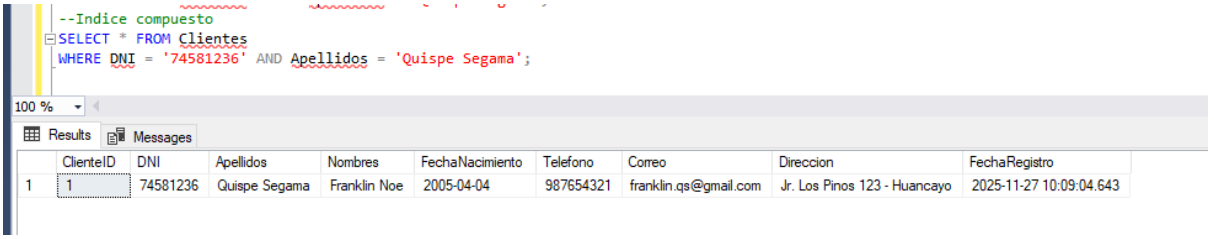
	ClienteID	DNI	Apellidos	Nombres	FechaNacimiento	Telefono	Correo	Direccion	FechaRegistro
1	1	74581236	Quispe Segama	Franklin Noe	2005-04-04	987654321	franklin.qs@gmail.com	Jr. Los Pinos 123 - Huancayo	2025-11-27 10:09:04.643

- Sin índice, SQL Server realiza *table scan*, impactando en rendimiento si la tabla es grande.

3. **Índice compuesto (opcional):**

- Se aplica solo si en el sistema son frecuentes consultas con **filtros por ambas columnas** simultáneamente.

- Mejora eficiencia en operaciones como:



  - El orden de las columnas favorece búsquedas donde el DNI es el primer criterio.

2. **No se usa índice clusterizado:**

  - Porque normalmente el clusterizado ya existe sobre la llave primaria (por ejemplo, ClienteID).

  - Crear un cluster sobre DNI podría afectar otras áreas del sistema.

4. Explicación de las buenas prácticas utilizadas en el proyecto

✔ 1. Uso de índices no clusterizados:
Evita reorganizar toda la tabla y se enfoca solo en acelerar búsquedas específicas.

✔ 2. Creación de índices independientes:

- Se crean índices separados para **DNI** y **Apellidos**, lo cual permite que SQL Server use el más eficiente según la consulta.

- Se evita sobrecargar la tabla con índices compuestos innecesarios.

✔ 3. Consideración del índice compuesto solo si es necesario:
No se crean índices redundantes. Se evalúa la frecuencia de uso para no aumentar el costo de inserciones/actualizaciones.

✔ **4. Elección de columnas adecuadas:**

- **DNI**: altamente selectiva → ideal para búsquedas rápidas.
- **Apellidos**: usada frecuentemente en filtros → mejora especialmente en tablas grandes.

✔ **5. Nomenclatura estándar (IX_NombreTabla_Columna):**
Ayuda a mantener orden y legibilidad dentro del catálogo de índices.

✔ **6. No incluir columnas innecesarias en el índice:**
Mantiene el índice liviano y reduce costos de mantenimiento.

✔ **7. Consideración de patrones de consulta del negocio:**
La estructura de índices se basa en cómo las aplicaciones realmente consultan la tabla.

Proyecto 3: Detectar fragmentación y aplicar mantenimiento de índices

1. Enunciado del ejercicio

Usar las DMV (Dynamic Management Views) de SQL Server para evaluar el nivel de fragmentación de los índices de la base de datos **QhatuPeru**, y aplicar mantenimiento según el porcentaje encontrado:

- **Reorganizar** índices con fragmentación entre **10% y 30%**.
- **Reconstruir** índices con fragmentación mayor a **30%**.

2. Script de la solución en T-SQL

1. Ver fragmentación actual de índices (DMV)

```sql
--Proyecto 3
--1. Ver fragmentación actual de índices (DMV)
SELECT
    DB_NAME() AS BaseDatos,
    OBJECT_NAME(ips.object_id) AS Tabla,
    i.name AS NombreIndice,
    ips.index_id,
    ips.avg_fragmentation_in_percent AS Fragmentacion
FROM sys.dm_db_index_physical_stats(DB_ID('QhatuPeru'), NULL, NULL, NULL, 'SAMPLED') AS ips
INNER JOIN sys.indexes AS i
    ON ips.object_id = i.object_id AND ips.index_id = i.index_id
WHERE i.index_id > 0  -- excluir heap
ORDER BY ips.avg_fragmentation_in_percent DESC;
```

100 %

Results | Messages

	BaseDatos	Tabla	NombreIndice	index_id	Fragmentacion
1	QhatuPERU	Clientes	PK__Clientes__71ABD0A75688E83E	1	0
2	QhatuPERU	Clientes	UQ__Clientes__C035B8DD3BC1AEA3	2	0
3	QhatuPERU	Clientes	IX_Clientes_DNI	3	0
4	QhatuPERU	Clientes	IX_Clientes_Apellidos	4	0
5	QhatuPERU	Clientes	IX_Clientes_DNI_Apellidos	5	0

2. Aplicar mantenimiento automático de índices según nivel de fragmentación

```sql
--2. Aplicar mantenimiento automático de índices según nivel de fragmentación
DECLARE @object_id INT, @index_id INT, @frag FLOAT, @sql NVARCHAR(MAX);

DECLARE cur CURSOR FOR
SELECT
    ips.object_id,
    ips.index_id,
    ips.avg_fragmentation_in_percent
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 i.index_id > 0;   -- evita heap

OPEN cur;
FETCH NEXT FROM cur INTO @object_id, @index_id, @frag;

WHILE @@FETCH_STATUS = 0
BEGIN
    IF @frag > 30
    BEGIN
        -- Reconstruir índice (mejor opción para alta fragmentación)
        SET @sql =
            'ALTER INDEX [' +
            (SELECT name FROM sys.indexes WHERE object_id = @object_id AND index_id = @index_id) +
            '] ON [' +
            (SELECT name FROM sys.objects WHERE object_id = @object_id) +
            '] REBUILD;';
    END
    ELSE IF @frag BETWEEN 10 AND 30
    BEGIN
        -- Reorganizar índice (menos costoso)
        SET @sql =
            'ALTER INDEX [' +
            (SELECT name FROM sys.indexes WHERE object_id = @object_id AND index_id = @index_id) +
            '] ON [' +
            (SELECT name FROM sys.objects WHERE object_id = @object_id) +
            '] REORGANIZE;';
    END
    ELSE
    BEGIN
        SET @sql = NULL;
    END

    IF @sql IS NOT NULL
    BEGIN
        PRINT 'Ejecutando: ' + @sql;
        EXEC sp_executesql @sql;
    END

    FETCH NEXT FROM cur INTO @object_id, @index_id, @frag;
END

CLOSE cur;
DEALLOCATE cur;
```

```
100 %
Messages
Commands completed successfully.

Completion time: 2025-11-27T10:20:48.1037187-05:00
```

3. Justificación técnica de la solución aplicada

1. **Uso de sys.dm_db_index_physical_stats:**
   Esta DMV permite obtener métricas físicas de los índices:

   - Fragmentación promedio

- Nivel de página
- Densidad
- Tamaños

2. **Modo “SAMPLED”:**
   Es más liviano que “DETAILED” y suficiente para mantenimiento regular.

3. **Criterios de mantenimiento:**

Fragmentación	**Acción recomendada**
< 10%	No se realiza mantenimiento
10% – 30%	REORGANIZE
> 30%	REBUILD

4. Esta es la regla estándar recomendada por Microsoft.

5. **Reconstrucción (REBUILD):**
   - Reorganiza páginas internas
   - Elimina fragmentación interna y externa
   - Recalcula estadísticas automáticamente
   - Es más costoso pero más efectivo

6. **Reorganización (REORGANIZE):**
   - Operación ligera y en línea

- Ideal cuando la fragmentación es moderada
- No bloquea tanto como REBUILD

7. **Uso de cursor:**
   Permite recorrer índice por índice y tomar la acción adecuada.

8. **Generación dinámica de T-SQL:**
   Se adapta dinámicamente al nombre de la tabla e índice sin codificarlos a mano.

4. Buenas prácticas utilizadas en el proyecto

✔ 1. Mantenimiento automático basado en porcentaje real
No se reconstruyen índices sin necesidad = menos bloqueo y mejor rendimiento.

✔ 2. DMV en modo "SAMPLED"
Reduce impacto sobre el servidor.

✔ 3. Separación entre REBUILD y REORGANIZE
Optimiza recursos según nivel de fragmentación.

✔ 4. Exclusión de índices HEAP
No poseen estructura de índice, evitaría errores.

✔ 5. Impresión de comandos ejecutados
Permite auditoría y ver qué acción se tomó.

✔ 6. Uso de scripts parametrizables
La solución funciona en cualquier entorno sin modificar nombres.

✔ 7. Evitar mantenimiento innecesario
Acción técnica clave para no afectar rendimiento en tablas grandes.

Proyecto 4: Manejo de transacciones para prevenir inconsistencias

1. Enunciado del ejercicio

Simular una operación de venta donde se realizan **dos acciones dependientes**:

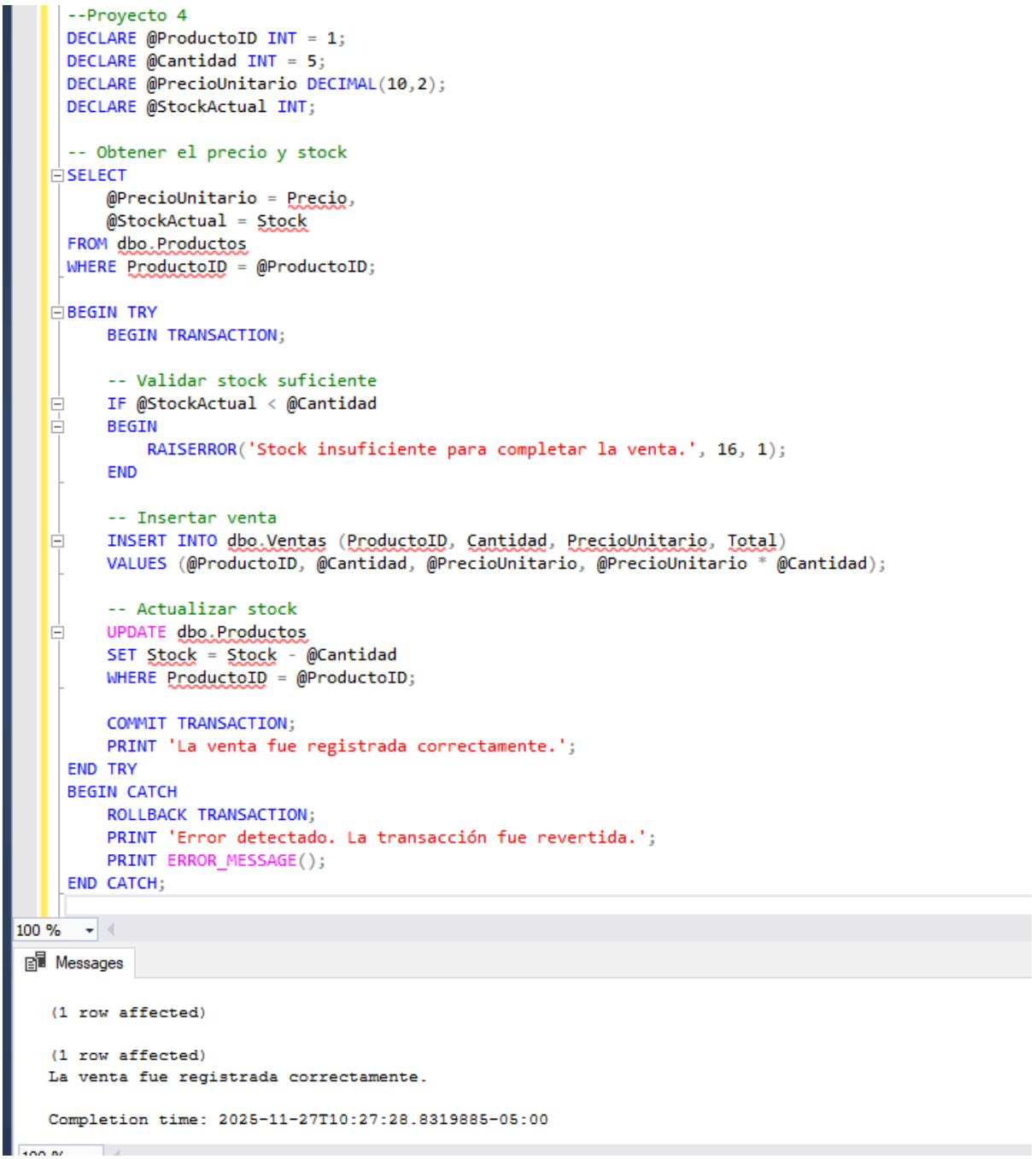
1. Insertar un registro en la tabla **Ventas**.
2. Actualizar el **Stock** de un producto.

La transacción debe garantizar **consistencia**, es decir:

- Si algo falla, **todo se revierte** (ROLLBACK).
- Si todo es correcto, **se confirma** (COMMIT).

2. Script T-SQL de la transacción simulada

- Vamos a simular que se vende **5 unidades del ProductoID = 1**.



```sql
--Proyecto 4
DECLARE @ProductoID INT = 1;
DECLARE @Cantidad INT = 5;
DECLARE @PrecioUnitario DECIMAL(10,2);
DECLARE @StockActual INT;

-- Obtener el precio y stock
SELECT
    @PrecioUnitario = Precio,
    @StockActual = Stock
FROM dbo.Productos
WHERE ProductoID = @ProductoID;

BEGIN TRY
    BEGIN TRANSACTION;

    -- Validar stock suficiente
    IF @StockActual < @Cantidad
    BEGIN
        RAISERROR('Stock insuficiente para completar la venta.', 16, 1);
    END

    -- Insertar venta
    INSERT INTO dbo.Ventas (ProductoID, Cantidad, PrecioUnitario, Total)
    VALUES (@ProductoID, @Cantidad, @PrecioUnitario, @PrecioUnitario * @Cantidad);

    -- Actualizar stock
    UPDATE dbo.Productos
    SET Stock = Stock - @Cantidad
    WHERE ProductoID = @ProductoID;

    COMMIT TRANSACTION;
    PRINT 'La venta fue registrada correctamente.';
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT 'Error detectado. La transacción fue revertida.';
    PRINT ERROR_MESSAGE();
END CATCH;
```

100 %

Messages

```
(1 row affected)

(1 row affected)
La venta fue registrada correctamente.

Completion time: 2025-11-27T10:27:28.8319885-05:00
```

3. Justificación técnica de la solución aplicada

✔ Uso de BEGIN TRANSACTION, COMMIT y ROLLBACK

Garantiza que **ambas operaciones** (insertar venta y actualizar stock) se ejecuten como una **unidad atómica**.

✔ Inclusión de bloque TRY…CATCH

Permite capturar errores como:

- Stock insuficiente
- Falla de integridad
- Conflictos de bloqueo
- Errores de conexión

Al capturar el error, se ejecuta un **ROLLBACK** automático.

✔ Validación previa de stock

Previene inconsistencias antes de modificar datos.

✔ Cálculo del total dentro de la transacción

Evita desalineaciones de precios.

✔ Llave foránea en Ventas

Garantiza integridad referencial entre Ventas ↔ Productos.

4. Buenas prácticas utilizadas

1. Control explícito de transacciones

Cada operación depende de la otra → Deben ser atómicas.

2. Uso de TRY/CATCH

Estandariza manejo de errores y evita que transacciones queden abiertas.

3. Validaciones de negocio antes de actualizar

Evita restar stock cuando no es posible realizar la venta.

4. Evitar lecturas inconsistentes

Se obtienen valores iniciales en variables antes de actualizar.

5. Seguridad con KEY y FK

Minimiza errores por productos inexistentes o mal referenciados.

6. Variables locales para evitar consultas repetidas

Menos carga y más claridad en el código.

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

1. Enunciado del ejercicio

Detectar todas las sesiones que actualmente están **bloqueando** o **siendo bloqueadas** dentro del servidor SQL Server, específicamente relacionadas con la base de datos **QhatuPeru**.
El objetivo es identificar:

- SPID que bloquea (blocking_session_id)
- SPID bloqueado
- Consulta que ejecutan
- Tiempo del bloqueo
- Estado de la sesión

Esto es esencial para recuperación rápida (PITR – *Problem Identification and Troubleshooting Response*).

2. Script de la solución en T-SQL

1. Ver bloqueos activos usando DMVs

```sql
--Proyecto 5
--5.1 Ver bloqueos activos usando DMVs
SELECT
    DB_NAME(r.database_id) AS BaseDatos,
    r.session_id AS SPID,
    r.blocking_session_id AS SPID_Bloqueador,
    s.host_name AS Host,
    s.login_name AS Usuario,
    r.status AS Estado,
    r.wait_type AS TipoEspera,
    r.wait_time AS TiempoEspera_ms,
    r.cpu_time AS CPU_ms,
    r.logical_reads AS LecturasLogicas,
    t.text AS ConsultaEjecutada
FROM sys.dm_exec_requests r
INNER JOIN sys.dm_exec_sessions s
    ON r.session_id = s.session_id
OUTER APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE
    r.blocking_session_id <> 0
    OR r.session_id IN (SELECT blocking_session_id FROM sys.dm_exec_requests WHERE blocking_session_id <> 0)
ORDER BY r.wait_time DESC;
```

100 %

Results | Messages

BaseDatos	SPID	SPID_Bloqueador	Host	Usuario	Estado	TipoEspera	TiempoEspera_ms	CPU_ms	LecturasLogicas	ConsultaEjecutada

2. Ver solo pares bloqueado ↔ bloqueador

```sql
--5.2. Ver solo pares bloqueado ↔ bloqueador
SELECT
    r.blocking_session_id AS SPID_Bloqueador,
    r.session_id AS SPID_Bloqueado,
    t1.text AS Consulta_Bloqueadora,
    t2.text AS Consulta_Bloqueada,
    r.wait_type,
    r.wait_time,
    r.database_id
FROM sys.dm_exec_requests r
OUTER APPLY sys.dm_exec_sql_text(
    (SELECT sql_handle FROM sys.dm_exec_requests WHERE session_id = r.blocking_session_id)
) t1
OUTER APPLY sys.dm_exec_sql_text(r.sql_handle) t2
WHERE r.blocking_session_id <> 0
ORDER BY r.wait_time DESC;
```

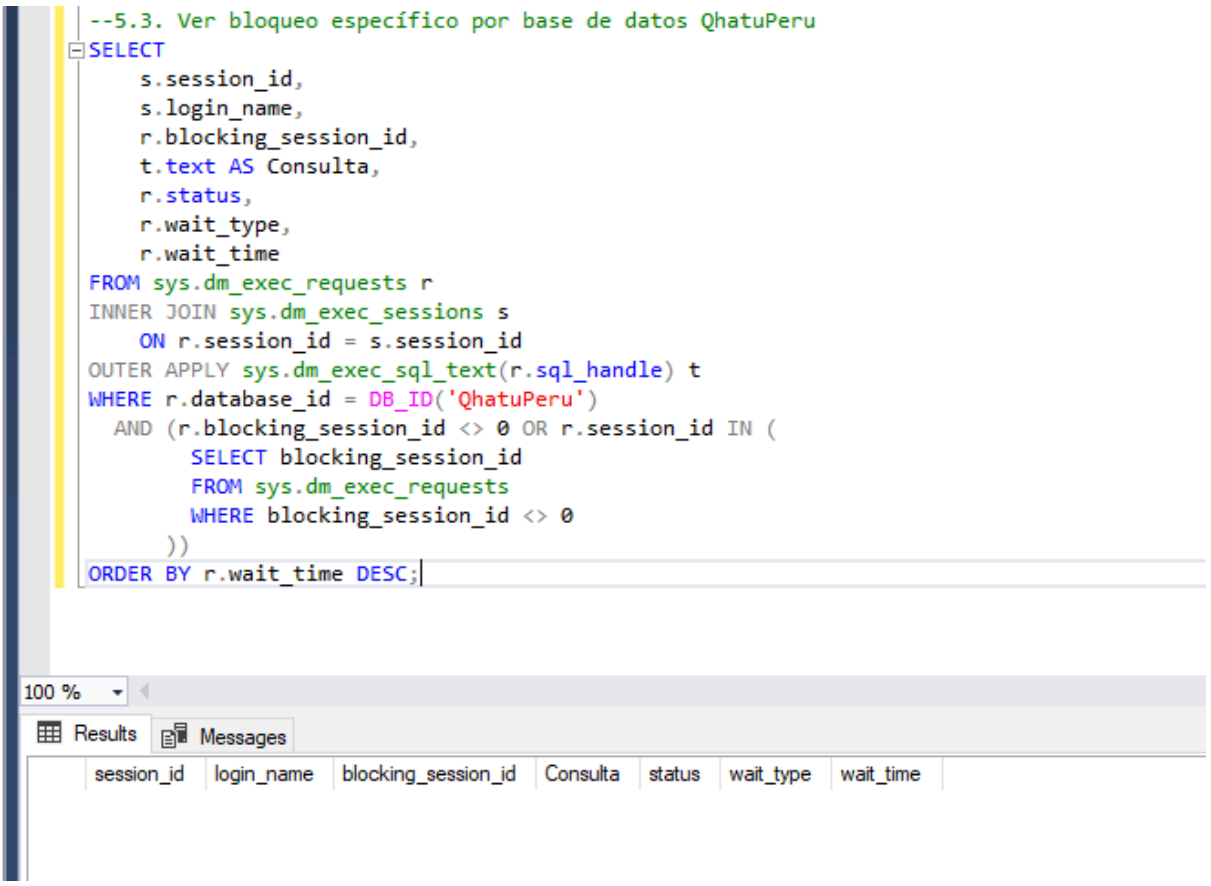
100 %

Results | Messages

SPID_Bloqueador	SPID_Bloqueado	Consulta_Bloqueadora	Consulta_Bloqueada	wait_type	wait_time	database_id

Click to select the whole column

3. Ver bloqueo específico por base de datos QhatuPeru



3. Justificación técnica de la solución aplicada

✔ Uso de sys.dm_exec_requests

Permite visualizar:

- Sesiones activas
- Sesiones bloqueadas
- Sesiones que bloquean
- Tipos de espera (wait_type)

Es la DMV clave para analizar bloqueos.

✔ Uso de blocking_session_id

Indica qué SPID está reteniendo recursos y generando el bloqueo.

✔ **Uso de sys.dm_exec_sql_text(sql_handle)**

Permite ver la consulta exacta que causa el problema.
Sin esto sería imposible un análisis completo.

✔ **Filtros por DB_ID('QhatuPeru')**

Asegura que el análisis se centre exclusivamente en la base solicitada.

✔ **Orden por wait_time**

Ayuda a identificar problemas críticos primero.

4. Buenas prácticas utilizadas en el proyecto

1. Uso de DMVs en tiempo real

Permite identificar bloqueos sin necesidad de instalar herramientas externas.

2. Auditoría detallada de SPID

Se identifica:

- Host
- Usuario
- CPU usada
- Lecturas lógicas
- Tipo de espera

Información esencial para análisis de rendimiento y PITR.

3. OUTER APPLY para obtener consultas

Es una técnica avanzada que permite unir SQL dinámico con texto de consulta.

4. Enfoque en sesiones bloqueadoras

No solo se listan bloqueos, sino el **origen del problema**.

5. Priorización: orden por tiempo de espera

Permite atender primero los bloqueos que afectan más al sistema.

6. Sin afectar el rendimiento del servidor

Las DMVs usadas no generan carga significativa.

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 las ventas por producto en la base de datos **QhatuPeru**, identificando operadores costosos y posibles optimizaciones.

2. Script de la solución en T-SQL

2. 1. Crear tablas base (si no existen)

```sql
--Proyecto 6
--6.1 Crear Tabla si no existe

USE QhatuPeru;
GO

-- Tabla Productos
IF OBJECT_ID('Productos', 'U') IS NULL
CREATE TABLE Productos (
    ProductoID INT PRIMARY KEY IDENTITY,
    NombreProducto VARCHAR(100),
    Categoria VARCHAR(50)
);

-- Tabla Ventas
IF OBJECT_ID('Ventas', 'U') IS NULL
CREATE TABLE Ventas (
    VentaID INT PRIMARY KEY IDENTITY,
    ProductoID INT,
    Cantidad INT,
    Precio DECIMAL(10,2),
    FechaVenta DATE,
    FOREIGN KEY (ProductoID) REFERENCES Productos(ProductoID)
);
```

2. 2. Insertar datos de prueba

```sql
--6.2. Insertar datos de prueba
INSERT INTO Productos (NombreProducto, Categoria)
VALUES ('Pollo', 'Abarrotes'),
       ('Carne', 'Abarrotes'),
       ('Leche Entera', 'Lácteos'),
       ('Huevos', 'Granja');

INSERT INTO Ventas (ProductoID, Cantidad, Precio, FechaVenta)
VALUES
(1, 5, 12.50, '2025-01-10'),
(1, 10, 11.00, '2025-01-12'),
(2, 7, 14.00, '2025-01-11'),
(3, 20, 4.50, '2025-01-12'),
(4, 30, 0.80, '2025-01-13');
```

3. Consulta lenta a analizar

```sql
--6.3. Consulta lenta a analizar
SELECT
    p.NombreProducto,
    SUM(v.Cantidad) AS TotalVendido,
    SUM(v.Cantidad * v.Precio) AS TotalGenerado
FROM Ventas v
INNER JOIN Productos p
    ON v.ProductoID = p.ProductoID
GROUP BY p.NombreProducto;
```

100 %

Results | Messages

	NombreProducto	TotalVendido	TotalGenerado
1	Carne	7	98.00
2	Huevos	30	24.00
3	Leche Entera	20	90.00
4	Pollo	15	172.50

4. Habilitar visualización del plan de ejecución

```
--6.4. Habilitar visualización del plan de ejecución
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
```

```
100 %
Messages
Commands completed successfully.

Completion time: 2025-11-27T12:51:13.5874289-05:00
```

Luego en SSMS activar:

Ctrl + M → "Include Actual Execution Plan"

Ejecutar nuevamente la consulta lenta.

```
--6.4. Habilitar visualización del plan de ejecución
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
```

```
100 %
Messages
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

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

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

Completion time: 2025-11-27T12:53:10.2672959-05:00
```

3. Justificación técnica de la solución aplicada

✔ 1. La consulta usa un JOIN + agregaciones

Requiere que SQL Server:

- recorra la tabla Ventas,
- encuentre coincidencias en Productos,
- agrupe por NombreProducto,
- calcule SUM.

Esto puede ser costoso sin índices adecuados.

✔ 2. El plan mostrará operadores clave

Durante el análisis se identificarán operadores como:

- **Table Scan** → si no hay índices en Ventas.ProductoID o Productos.ProductoID
- **Hash Match** → usado para agrupar y unir, costoso en memoria
- **Sort** → si SQL Server necesita ordenar para agrupar

Estos indicadores revelan por qué la consulta es lenta.

✔ 3. Uso de STATISTICS IO y TIME

Permiten medir:

- lecturas lógicas
- lecturas físicas
- tiempo CPU
- tiempo total de ejecución

Con eso puedes comparar antes y después de optimizar.

4. Explicación de las buenas prácticas utilizadas en el proyecto

1. Separación correcta de tablas

Dividir Ventas y Productos es una buena práctica de normalización.

2. Identificación de cuellos de botella usando el plan de ejecución

Como ingeniero de base de datos, analizar:

- Scans innecesarios
- Falta de índices
- Operadores Hash costosos

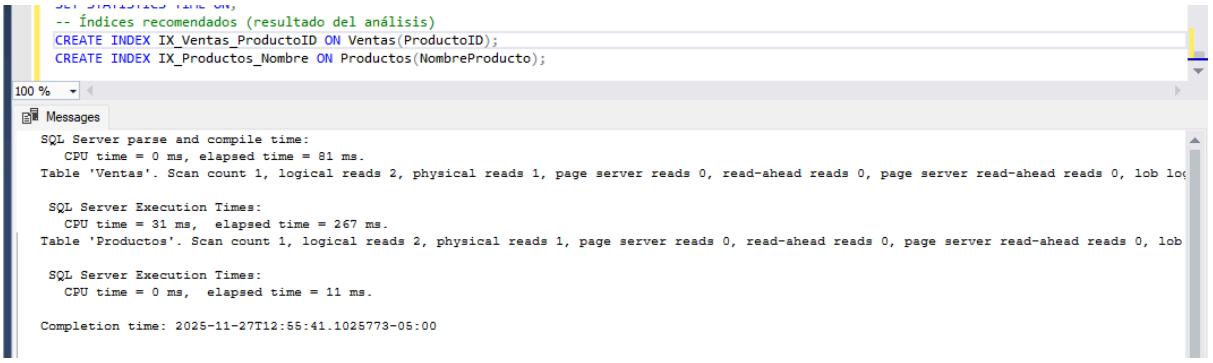
Es fundamental para un servidor productivo.

3. Activación de estadísticas IO y TIME

Provee datos reales para evaluar el rendimiento.
Sin esto, las optimizaciones serían "a ciegas".

4. Índices recomendados (resultado del análisis)

Después del análisis, los índices candidatos suelen ser:



Estos índices eliminan Table Scans y reducen tiempo.

5. Validar mejora después de indexar

Una buena práctica es comparar el plan de ejecución antes y después.

PROYECTO 7: Optimización de consulta agregada con índices compuestos

1. Enunciado del ejercicio

Optimizar una consulta que obtiene ventas filtradas por **rango de fechas** y por **ClienteID**, aplicando un **índice compuesto** que mejore el rendimiento en la base de datos **QhatuPeru**.

2. Script de la solución en T-SQL

Antes de optimizar, se asegura la existencia de una tabla **Clientes** y una tabla **Ventas** con una relación.

1. Crear tabla Clientes (si no existe)

```sql
--Proyecto 7
--1. Crear tabla Clientes (si no existe)
USE QhatuPeru;
GO

IF OBJECT_ID('Clientes', 'U') IS NULL
CREATE TABLE Clientes (
    ClienteID INT PRIMARY KEY IDENTITY,
    Nombres VARCHAR(100),
    Apellidos VARCHAR(100),
    DNI CHAR(8)
);
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 15 ms, elapsed time = 133 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 31 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 31 ms.
```

100 %

2. Agregar columna ClienteID a Ventas (si aún no existe)

```
);
--2. Agregar columna ClienteID a Ventas (si aún no existe)
IF COL_LENGTH('Ventas','ClienteID') IS NULL
ALTER TABLE Ventas
ADD ClienteID INT FOREIGN KEY REFERENCES Clientes(ClienteID);
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 15 ms, elapsed time = 136 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 5 ms.
```

3. Insertar datos de prueba

```
--3. Insertar datos de prueba
INSERT INTO Clientes (Nombres, Apellidos, DNI)
VALUES
('Carlos', 'Sanchez Lopez', '72345678'),
('María', 'Huaman Torres', '81234567'),
('Pedro', 'Mendez Rojas', '70223344');

UPDATE Ventas
SET ClienteID = 1
WHERE VentaID IN (1,2);

UPDATE Ventas
SET ClienteID = 2
WHERE VentaID IN (3,4,5);
```

4. Consulta original (lenta)

```
--4. Consulta original (lenta)
SELECT
    ClienteID,
    SUM(Cantidad) AS TotalProductos,
    SUM(Cantidad * PrecioUnitario) AS TotalGenerado
FROM Ventas
WHERE FechaVenta BETWEEN '2025-01-01' AND '2025-01-31'
  AND ClienteID = 2
GROUP BY ClienteID;
```

100 %

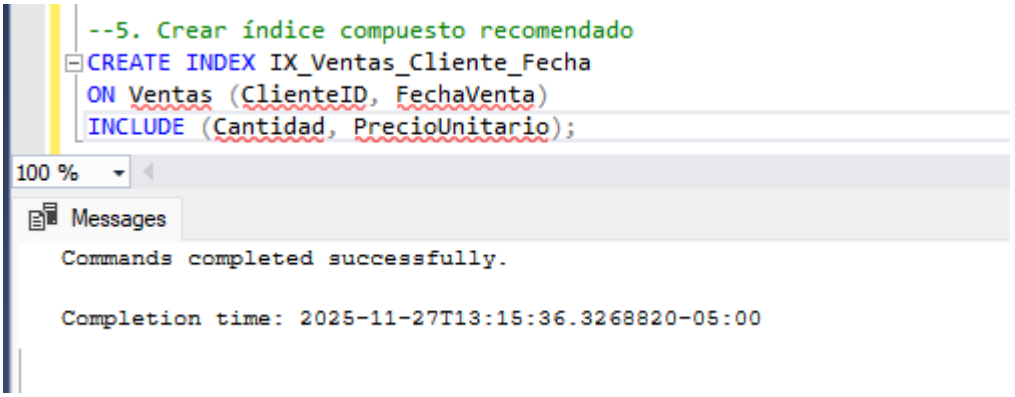
Results | Messages

ClienteID	TotalProductos	TotalGenerado

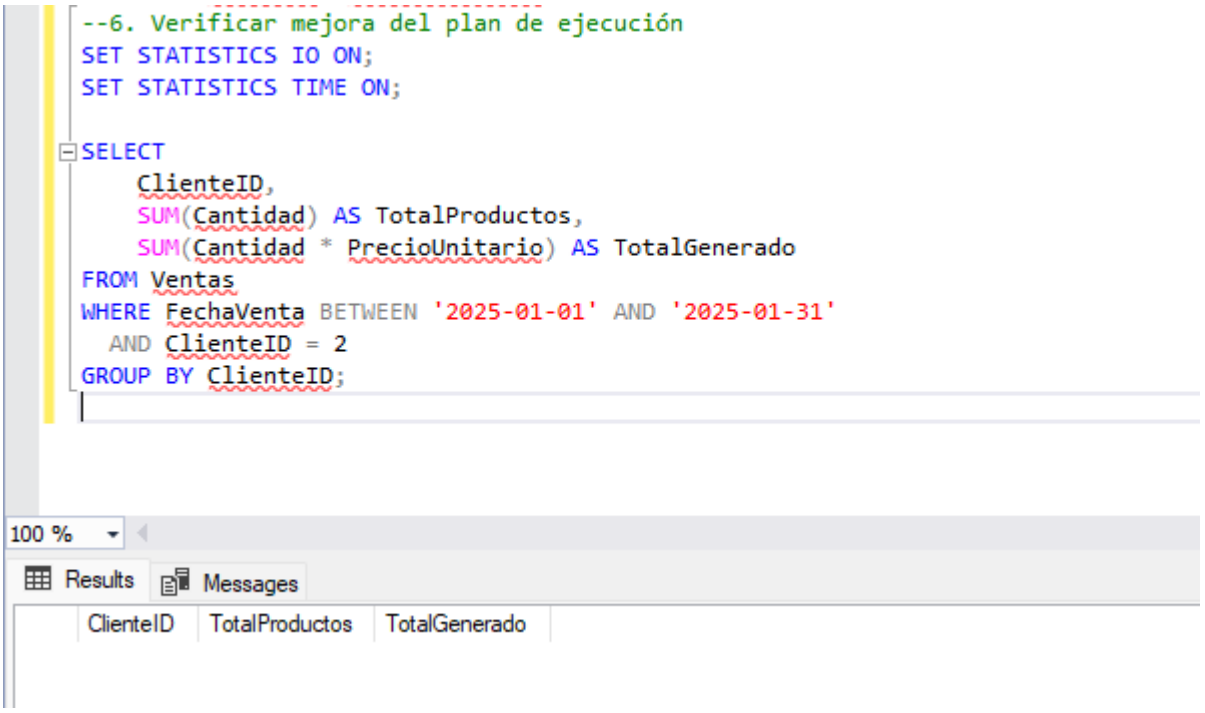
5. Crear índice compuesto recomendado

Orden de columnas optimizado:

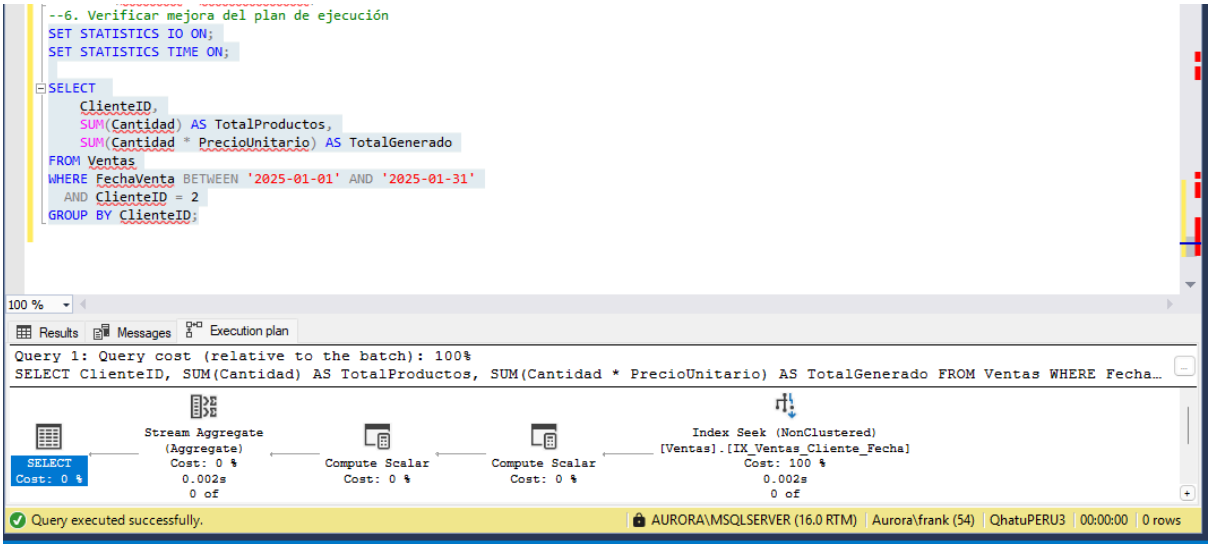
1. ClienteID → columna más selectiva
2. FechaVenta → filtro por rango
3. Incluye columnas usadas en SELECT para consulta *covering*



6. Verificar mejora del plan de ejecución



Activa el plan real en SSMS (**Ctrl+M**) para comparar.



3. Justificación técnica de la solución aplicada

✔ 1. Uso de índice compuesto para mejorar filtrado múltiple

La consulta filtra por:

- **ClienteID (igualdad)**
- **FechaVenta (rango BETWEEN)**

SQL Server puede buscar de forma **altamente selectiva** porque:

- Primero filtra por ClienteID (muy pocas filas)
- Luego filtra por FechaVenta (rango pequeño)
- Finalmente hace la agregación

Esto elimina **Table Scans** y reduce costos de CPU y lecturas lógicas.

✔ 2. INCLUDE mejora la consulta

Las columnas:

- Cantidad

- Precio

se agregan con **INCLUDE**, permitiendo que el índice contenga todos los datos necesarios.

Eso convierte el índice en un **índice cubriente (covering index)**
→ SQL Server ya no toca la tabla base.

✔ 3. ORDER CORRECTO DEL ÍNDICE

En índices compuestos, el orden influye directamente en el rendimiento:

- Filtros por igualdad → primero (**ClienteID**)
- Filtros por rango → después (**FechaVenta**)

Este patrón es considerado una buena práctica para consultas por fechas.

4. Explicación de las buenas prácticas utilizadas en el proyecto

1. Uso de índices compuestos en consultas de alto tráfico

Consultas con múltiples filtros deben diseñarse con índices especialmente ordenados para reducir el costo de búsqueda.

2. Índice cubriente para evitar Key Lookups

Agregar columnas en INCLUDE evita accesos innecesarios a la tabla, mejorando:

- velocidad
- IO
- tiempos de CPU

3. Medición objetiva antes y después

Usar STATISTICS IO y STATISTICS TIME es obligatorio en optimización profesional.

Permite:

- Comparar lecturas lógicas
- Verificar disminución del tiempo de ejecución
- Validar el impacto real del índice

4. Respeto por normalización y claves foráneas

Asegurar relaciones correctas (ClienteID → Clientes) mantiene la integridad referencial del sistema.

5. Documentación clara del proceso

Cada paso (datos, índice, análisis) se deja explícito para reproducción en auditoría o revisiones técnicas.

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** dentro de la base de datos **QhatuPeru**, con el objetivo de mejorar el rendimiento del optimizador en consultas que realizan filtros por **rangos de precio**.

2. Script de la solución en T-SQL

1. Verificar que la tabla Productos tenga la columna Precio

```sql
--Proyecto 8
--1. Verificar que la tabla Productos tenga la columna Precio
USE QhatuPERU3;
GO

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

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 3 ms.

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

2. Insertar precios de ejemplo (si no existen)

```sql
--2. Insertar precios de ejemplo (si no existen)
UPDATE Productos SET Precio = 12.50 WHERE Nombre = 'Pollo';
UPDATE Productos SET Precio = 14.00 WHERE Nombre = 'Carne';
UPDATE Productos SET Precio = 4.50  WHERE Nombre = 'Leche Entera';
UPDATE Productos SET Precio = 0.80  WHERE Nombre = 'Huevos';
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 46 ms, elapsed time = 352 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'Productos'. Scan count 1, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 15 ms.

(0 rows affected)
```

3. Crear estadística manual sobre la columna Precio

```sql
--3. Crear estadística manual sobre la columna Precio
CREATE STATISTICS Estadistica_Precio_Productos
ON Productos(Precio)
WITH FULLSCAN;    -- Fuerza a SQL Server a analizar todas las filas para mayor precisión

4. Ejemplo de consulta que se beneficia
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 15 ms, elapsed time = 114 ms.

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 25 ms.

Completion time: 2025-11-27T13:30:48.5155799-05:00
```

4. Ejemplo de consulta que se beneficia

```
--4. Ejemplo de consulta que se beneficia
SELECT *
FROM Productos
WHERE Precio BETWEEN 5 AND 15;

5. Comprobar que la estadística existe
SELECT name, auto_created, user_created, has_filter
```

00 %

Results | Messages

ProductoID	Nombre	Descripcion	Stock	Precio

5. Comprobar que la estadística existe

```
--5. Comprobar que la estadística existe
SELECT name, auto_created, user_created, has_filter
FROM sys.stats
WHERE object_id = OBJECT_ID('Productos');
```

100 %

Results | Messages

	name	auto_created	user_created	has_filter
1	PK__Producto__A430AE8367B64A94	0	0	0
2	Estadistica_Precio_Productos	0	1	0

3. Justificación técnica de la solución aplicada

1 . **El optimizador de consultas utiliza histogramas de estadísticas** para estimar la cantidad de filas que cumplen un filtro por rango.
La columna Precio es una columna típica para rangos (por ejemplo: productos entre 5 y 20 soles).

2. SQL Server normalmente crea estadísticas automáticamente, pero no siempre lo hace sobre columnas que no están indexadas.
Crear una estadística manual garantiza que el optimizador tenga un histograma

detallado y actualizado.

3. WITH FULLSCAN genera la estadística usando **todas** las filas, lo cual incrementa la precisión, especialmente útil en tablas pequeñas o medianas.

4. Las consultas con filtro en Precio podrán determinar el mejor plan de ejecución (por ejemplo, decidir entre un Table Scan o un Index Seek si hubiera índice).

4. Explicación de las buenas prácticas utilizadas en el proyecto

1. **Validación previa de existencia de columnas**:
   Antes de crear la estadística, se valida que la columna Precio exista. Esto evita errores y asegura la integridad del script.

2. **Uso de estadísticas manuales como complemento, no reemplazo**:
   Las estadísticas automáticas siguen activas, pero se añade una específica para una columna crítica.

3. **Uso de FULLSCAN**:
   Esta opción asegura estadísticas más precisas, especialmente recomendable en ambientes OLAP o tablas pequeñas/medianas donde el costo del escaneo completo es bajo.

4. **Verificación posterior**:
   Consultar sys.stats asegura control y transparencia de qué estadísticas fueron creadas, útil para auditoría y monitoreo.

5. **Separación clara entre carga de datos, creación de estadísticas y pruebas de consulta**:
   Mantiene un flujo de trabajo ordenado y replicable, fundamental en entornos profesionales.

PROYECTO 9: Configuración de un Resource Pool para limitar recursos

1. Enunciado del ejercicio

Crear un **Resource Pool** en SQL Server que limite el uso de **CPU al 20%** para consultas analíticas pesadas, garantizando que no afecten al resto del sistema.

2. Script de la solución en T-SQL

> **Nota:** Requiere **SQL Server Enterprise** para Resource Governor completo.

1. Habilitar Resource Governor

```
--Proyecto 9
-- 1. Habilitar Resource Governor
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 39 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 51 ms.

Completion time: 2025-11-27T13:37:14.0663317-05:00
```

2. Crear Resource Pool limitado al 20% de CPU

```
--2. Crear Resource Pool limitado al 20% de CPU
CREATE RESOURCE POOL Pool_Analitico
WITH (
    MAX_CPU_PERCENT = 20,
    MIN_CPU_PERCENT = 0
);
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 96 ms.

Completion time: 2025-11-27T13:38:02.2559404-05:00
```

3. Crear grupo de trabajo asociado

```
--3. Crear grupo de trabajo asociado
CREATE WORKLOAD GROUP Grupo_Analitico
USING Pool_Analitico;
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 64 ms.

Completion time: 2025-11-27T13:38:40.8187928-05:00
```

4. Crear función clasificadora para enviar consultas analíticas al pool

La función identifica sesiones por ApplicationName.

```
--4. Crear función clasificadora para enviar consultas analíticas al pool
CREATE FUNCTION dbo.Func_Clasificar_Sesiones()
RETURNS sysname
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @Grupo sysname;

    IF (APP_NAME() = 'AnalisisApp')
        SET @Grupo = 'Grupo_Analitico';
    ELSE
        SET @Grupo = 'default';

    RETURN @Grupo;
END;
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 9 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 36 ms.
```

100 %

Query executed successfully. | AURORA\MSQLSERVER (16.0 RTM) | Aurora\franl

5. Asociar la función al Resource Governor

```
--5. Asociar la función al Resource Governor
ALTER RESOURCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = dbo.Func_Clasificar_Sesiones);
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 28 ms.
```

6. Reconfigurar para aplicar cambios

```
--6. Reconfigurar para aplicar cambios
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

100 %

Messages

```
SQL Server parse and compile time:
   CPU time = 16 ms, elapsed time = 43 ms.

 SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 161 ms.
```

100 %

7. Probar conexión identificada como analítica

Desde el cliente o SSMS:

```
--7. Probar conexión identificada como analítica
USE QhatuPeru;
GO
SELECT APP_NAME();
```

100 %

Results | Messages

	(No column name)
1	Microsoft SQL Server Management Studio - Query

Y las consultas pesadas de esa aplicación usarán **solo el 20% de CPU**.

3. Justificación técnica de la solución aplicada

✔ 1. Resource Pool limita recursos

Con MAX_CPU_PERCENT = 20, se garantiza:

- Las consultas analíticas nunca consumen más del 20% de CPU.
- El OLTP o consultas principales no se ven afectadas.

✔ 2. Uso de Workload Groups segmenta cargas

Agrupar conexiones permite controlar:

- límites de memoria

- paralelismo
- prioridad

Para cargas específicas como análisis.

✔ 3. Función clasificadora controla qué sesiones entran

La clasificación basada en nombre de aplicación es una práctica estándar.

También se puede clasificar por:

- login
- host
- usuario

✔ 4. RECONFIGURE activa los cambios

Obligatorio para que Resource Governor aplique los nuevos pools y reglas.

4. Buenas prácticas aplicadas

- Separar cargas OLTP y analíticas evita saturación del servidor.
- Limitar CPU protege la operación principal del negocio.
- Nombrar pools y grupos con propósito claro (Pool_Analitico).
- Dejar documentada la función clasificadora.
- Evitar clasificaciones ambiguas o sobrecomplejas.

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

1. Enunciado del ejercicio

Auditar las **inserciones** en la tabla **Productos** usando Extended Events, evitando triggers pesados y manteniendo un impacto mínimo en el rendimiento.

2. Script de la solución en T-SQL

1. Crear una sesión de Extended Events para auditoría

```
--1. Crear una sesión de Extended Events para auditoría
CREATE EVENT SESSION AuditoriaInsercionesProductos
ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION (
        sqlserver.sql_text,
        sqlserver.client_app_name,
        sqlserver.username,
        sqlserver.database_name
    )
    WHERE (
        sqlserver.database_name = 'QhatuPeru3'
        AND sqlserver.sql_text LIKE '%INSERT INTO Productos%'
    )
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\XE\AuditoriaProductos.xel',
        max_file_size = 10,
        max_rollover_files = 5
);
GO
ALTER EVENT SESSION AuditoriaInsercionesProductos ON SERVER STATE = START;
GO
```

```
100 %
Messages
  SQL Server parse and compile time:
     CPU time = 0 ms, elapsed time = 45 ms.

   SQL Server Execution Times:
     CPU time = 15 ms,  elapsed time = 267 ms.
```

2. Insertar para probar auditoría

```
-- 2. Insertar para probar auditoría

INSERT INTO Productos (Nombre, Descripcion, Categoria, Stock, Precio)
VALUES ('Queso Andino', 'Lácteos', 'Quesos', 10, 12.50);
```

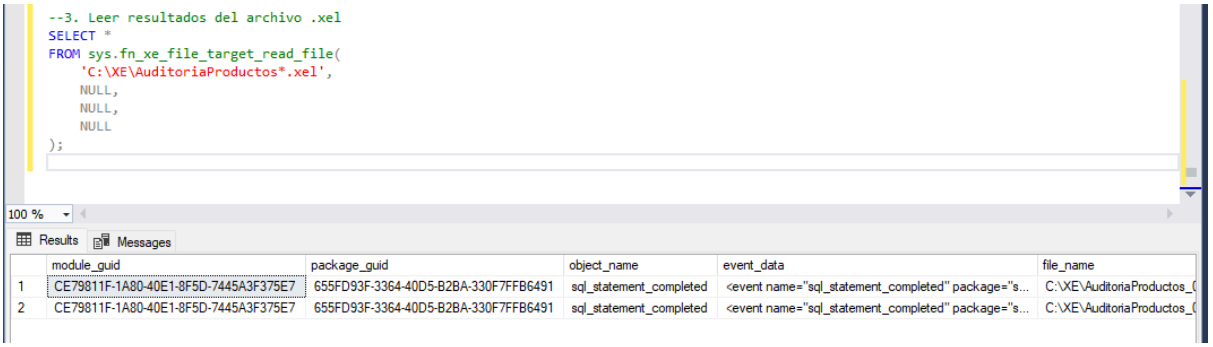
```
100 %
Messages
  SQL Server parse and compile time:
     CPU time = 0 ms, elapsed time = 20 ms.
  SQL Server parse and compile time:
     CPU time = 0 ms, elapsed time = 0 ms.
  Table 'Productos'. Scan count 0, logical reads 3, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob

   SQL Server Execution Times:
     CPU time = 0 ms,  elapsed time = 19 ms.

  (1 row affected)

  Completion time: 2025-11-27T13:57:05.6030461-05:00
```

3. Leer resultados del archivo .xel



```
--3. Leer resultados del archivo .xel
SELECT *
FROM sys.fn_xe_file_target_read_file(
    'C:\XE\AuditoriaProductos*.xel',
    NULL,
    NULL,
    NULL
);
```

100 %

Results | Messages

	module_guid	package_guid	object_name	event_data	file_name
1	CE79811F-1A80-40E1-8F5D-7445A3F375E7	655FD93F-3364-40D5-B2BA-330F7FFB6491	sql_statement_completed	<event name="sql_statement_completed" package="s...	C:\XE\AuditoriaProductos_(
2	CE79811F-1A80-40E1-8F5D-7445A3F375E7	655FD93F-3364-40D5-B2BA-330F7FFB6491	sql_statement_completed	<event name="sql_statement_completed" package="s...	C:\XE\AuditoriaProductos_(

3. Justificación técnica

- Los triggers tradicionales pueden **bloquear** la tabla o aumentar tiempo de respuesta.
- Extended Events audita sin interferir con el proceso transaccional.
- El filtro sql_text LIKE '%INSERT INTO Productos%' garantiza que solo se capture lo necesario.
- Se usa sqlserver.sql_statement_completed ya que toma el SQL real completado.

4. Buenas prácticas utilizadas

- Auditoría **fuera del motor**, almacenada en archivo .xel.
- Evitar triggers pesados que:
    - impacten transacciones,
    - generen logs innecesarios,
    - bloqueen filas.
- Filtrado mínimo para no generar ruido.
- Ruta de almacenamiento optimizada para disco rápido (C:\XE).
- Sesión de XE con **event_file** en lugar de ring_buffer (mejor para producción).