



UNIVERSIDAD PERUANA LOS ANDES

**Asignatura:  
BASE DE DATOS II**

**Tema:  
PRACTICA S13 MONITOREO Y RENDIMIENTO**

**Docente:  
FERNANDEZ BEJARANO RAUL**

**Estudiante:  
BREÑA QUISPE MEGAM**

HUANCAYO-2025

# 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.

## 2. Script de la solución en T-SQL

```
CREATE EVENT SESSION XE_QhatuPeru_LongQueries
ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION
    (
        sqlserver.sql_text,
        sqlserver.session_id,
        sqlserver.database_name,
        sqlserver.client_hostname,
        sqlserver.client_app_name,
        sqlserver.plan_handle
    )
    WHERE
    (
        duration > 1000000 -- 1 segundo
        AND sqlserver.database_name = 'QhatuPeru'
    )
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVERDEV\MSSQL\Log\QhatuPeru_LongQueries.xel',
    max_file_size = 50,
    max_rollover_files = 5
);
GO

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

100 %

Messages

Commands completed successfully.

Completion time: 2025-11-27T10:02:50.2707557-05:00

Crea la sesión con la ruta correcta

```
CREATE EVENT SESSION XE_QhatuPeru_LongQueries
ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    ACTION
    (
        sqlserver.sql_text,
        sqlserver.session_id,
        sqlserver.database_name,
        sqlserver.client_hostname,
        sqlserver.client_app_name,
        sqlserver.plan_handle
    )
    WHERE
    (
        duration > 1000000 -- 1 segundo
        AND sqlserver.database_name = 'QhatuPeru'
    )
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVERDEV\MSSQL\Log\QhatuPeru_LongQueries.xel',
    max_file_size = 50,
    max_rollover_files = 5
);
GO
```

¿Cómo verificar que está funcionando?

```
SELECT * FROM sys.dm_xe_sessions;
```

Consulta lenta

```
USE QhatuPeru;
```

```
GO
```

```
WAITFOR DELAY '00:00:02'; -- 2 segundos
```

```
SELECT 1;
```

Revisar archivos

```
SELECT *
```

```
FROM sys.fn_xe_file_target_read_file(
```

```
'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVERDEV\MSSQL\Log\QhatuPeru_LongQueries*.xel',
```

```
NULL, NULL, NULL
```

```
);
```

The screenshot shows the SQL Server Management Studio interface with two panes. The top pane displays the T-SQL code used to identify long-running sessions and check their buffer usage. The bottom pane shows the results of the query, which lists several sessions along with their buffer statistics.

address	name	pending_buffers	total_regular_buffers	regular_buffer_size	total_large_buffers	large_buffer_size	total_buffer_size	buffer_policy_flags	buffer_policy_desc	flags	flag_desc	dropped_event_count	dropped_buffer_count
1 0x000001FB576BD021	hkenginexession	0	3	1441587	0	0	4324761	1	block	1	flush_on_close	0	0
2 0x000001FB61358461	system_health	0	3	1441587	0	0	4324761	0	drop_event	2049	flush_on_close	0	0
3 0x000001FB677D1511	sp_server_diagnostics session	0	3	130867	0	0	392601	0	drop_event	17	flush_on_close private	0	0
4 0x000001FB64D39741	telemetry_xevents	0	3	1441587	0	0	4324761	0	drop_event	2049	flush_on_close	0	0
5 0x000001FB76106041	XE_QhatuPeru_LongQueries	0	3	1441587	0	0	4324761	0	drop_event	2049	flush_on_close	0	0

```
SELECT *
FROM sys.fn_xe_file_target_read_file(
'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVERDEV\MSSQL\Log\QhatuPeru_LongQueries*.xel',
NULL, NULL, NULL
);
```

The screenshot shows the SQL Server Management Studio interface with two panes. The top pane displays the T-SQL code used to check for completed SQL statements in the XE log. The bottom pane shows the results of the query, which lists completed statements along with their details.

module_guid	package_guid	object_name	event_data	file_name	file_offset	timestamp_utc
1 CE79811F-1A80-40E1-8F5D-7445A3F375E7	655FD93F-3364-40D5-B2BA-330F7FFB6491	sql_statement_completed	<event name="sql_statement_completed" package="s...	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...	8192	2025-11-27 15:03:23.1270000
2 CE79811F-1A80-40E1-8F5D-7445A3F375E7	655FD93F-3364-40D5-B2BA-330F7FFB6491	sql_statement_completed	<event name="sql_statement_completed" package="s...	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...	9216	2025-11-27 15:05:14.4390000

Tarda más de 1s

### 3. Justificación técnica de la solución aplicada

#### Por qué Extended Events

Extended Events (XE) es la tecnología moderna recomendada por Microsoft para monitoreo de rendimiento.

Es liviana, flexible y reemplaza a SQL Trace y Profiler.

#### Evento elegido: `sql_statement_completed`

Este evento captura:

- Texto de la consulta ejecutada
- Duración
- CPU time
- Lecturas lógicas
- Plan handle
- Tiempo en microsegundos

Es ideal para detectar consultas lentas.

#### Filtro por duración > 1 segundo

```
duration > 1000000
```

XE mide en microsegundos → 1.000.000 µs = 1 segundo.

Esto evita sobrecargar la sesión capturando consultas rápidas.

#### Filtro por base de datos

```
sqlserver.database_name = 'QhatuPeru'
```

Con esto, XE ignora consultas de otras BBDD, reduciendo ruido y uso de disco.

#### Acciones adicionales (ACTION)

Se agregan campos que ayudan en diagnósticos:

- `sql_text`: ver la consulta completa
- `session_id`: identificar la conexión
- `database_name`: ver contexto
- `client_hostname`: identificar servidor de aplicación
- `client_app_name`: ver la aplicación cliente
- `plan_handle`: permite ver el plan de ejecución

#### Uso de `target event_file`

```
package0.event_file
```

Es la forma más común de almacenar datos de trazas.

Permite:

- Persistencia
- Revisión posterior
- Análisis con SSMS o Log Analytics

Se agregan límites (tamaño, rollovers) para evitar llenar el disco.

## 4. Explicación de las buenas prácticas utilizadas en el proyecto

### 1. Separación de ambiente con IF EXISTS

Antes de crear la sesión, se elimina si ya existe:

```
IF EXISTS (...) DROP EVENT SESSION ...
```

Evita errores por sesiones duplicadas.

### 2. Uso de filtros (predicates)

Filtrar por duración y base de datos reduce drásticamente el consumo de recursos, siguiendo la práctica recomendada por Microsoft:

```
duration > 1000000  
database_name = 'QhatuPeru'
```

### 3. Captura limitada y específica

Se capturan solo los campos necesarios (ACTION).  
Más acciones = más peso en disco → es mejor limitar.

### 4. Uso de event\_file como almacenamiento

Guardar en archivo evita pérdida de datos cuando SQL Server se reinicia.  
Además, permite análisis posterior.

### 5. Configuración de rollover

```
max_rollover_files = 5
```

Evita llenar el disco del servidor.

### 6. Sesión iniciada explícitamente

```
ALTER EVENT SESSION ... STATE = START;
```

Permite control del administrador para detenerla o iniciarla.

### 7. No se usó Profiler

Profiler = obsoleto, pesado, afecta rendimiento.

XE = recomendado desde SQL Server 2012+.

# 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.

## 2. Script de la solución en T-SQL

Crear tabla Clientes si no existe

```
IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'Clientes')
BEGIN
    CREATE TABLE Clientes (
        IdCliente INT IDENTITY PRIMARY KEY,
        DNI CHAR(8) NOT NULL,
        Apellidos VARCHAR(80) NOT NULL,
        Nombres VARCHAR(80),
        Direccion VARCHAR(120),
        Telefono VARCHAR(15)
    );
END;
GO
```

Índice 1: Índice único por DNI

```
CREATE UNIQUE NONCLUSTERED INDEX IX_Clientes_DNI
ON Clientes (DNI);
GO
```

Índice 2: Índice no cluster por Apellidos

Para mejorar búsquedas del tipo:

```
SELECT * FROM Clientes WHERE Apellidos LIKE 'Pérez%';
```

Se crea:

```
CREATE NONCLUSTERED INDEX IX_Clientes_Apellidos
ON Clientes (Apellidos);
GO
```

The screenshot shows the SQL Server Management Studio interface. In the top pane, there is a script editor window containing T-SQL code. The code creates a table 'Clientes' with columns IdCliente (primary key, identity), DNI (CHAR(8) NOT NULL), Apellidos (VARCHAR(80) NOT NULL), Nombres (VARCHAR(80)), Direccion (VARCHAR(120)), and Telefono (VARCHAR(15)). It also creates two non-clustered indexes: 'IX\_Clientes\_DNI' on the 'DNI' column and 'IX\_Clientes\_Apellidos' on the 'Apellidos' column. Below the script editor is a results grid showing one row of data from the 'Clientes' table. The data is as follows:

	IdCliente	DNI	Apellidos	Nombres	Direccion	Telefono
1	2	41098765	Pérez	Juan Carlos	Calle San Martín 202	999888777

### **3. Justificación técnica de la solución aplicada**

#### **Por qué un índice UNIQUE en DNI**

- El DNI es un dato único por cliente.
- El índice UNIQUE valida integridad lógica.
- SQL Server optimiza mejor índices UNIQUE porque sabe que solo hay un resultado posible.
- Acelera consultas exactas:

```
SELECT * FROM Clientes WHERE DNI = '87654321';
```

#### **Por qué un índice en Apellidos**

Los apellidos no son únicos, pero se usan mucho en búsquedas:

- Buscar por apellidos completos
- Buscar por apellidos con LIKE
- Ordenar por apellidos

#### **El índice mejora consultas como:**

```
SELECT * FROM Clientes WHERE Apellidos LIKE 'García%';
```

Sin índice → table scan

Con índice → index seek

#### **¿Por qué no usar un índice compuesto (DNI, Apellidos)?**

Porque:

- DNI ya es único, y consultado de forma independiente.
- Apellidos se consultan sin DNI.
- Índices compuestos deben usarse SOLO si la consulta usa ambos campos al mismo tiempo (no es el caso).

#### **4. Explicación de las buenas prácticas utilizadas en el proyecto**

##### **1. Índices solo donde hay alto beneficio**

Se indexan **columnas de búsqueda frecuente**, no todas.

##### **2. Índice UNIQUE cuando el dato es identificador**

Obliga a integridad de datos y mejora el optimizador de consultas.

##### **3. Evitar índices innecesarios**

No se crea un índice compuesto si no corresponde.

Evita sobrecargar las operaciones de INSERT y UPDATE.

##### **4. Seleccionar el tipo adecuado (nonclustered)**

Se usan índices **nonclustered** ya que el PRIMARY KEY ya es el índice clustered.

##### **5. Usar tipo de datos óptimos**

DNI se define como:

CHAR(8)

Porque siempre tiene longitud fija.

Esto hace que el índice sea más eficiente.

# Proyecto 3- Detectar fragmentación y aplicar mantenimiento de índices

## 1. Enunciado del ejercicio

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

## 2. Script de la solución en T-SQL

Detectar fragmentación

```
USE QhatuPeru;
GO
```

SELECT

```
    DB_NAME() AS DatabaseName,
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.index_id,
    ips.avg_fragmentation_in_percent,
    ips.page_count
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.index_id > 0 -- excluir el heap
ORDER BY ips.avg_fragmentation_in_percent DESC;
GO
```

```
USE QhatuPeru;
GO

SELECT
    DB_NAME() AS DatabaseName,
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.index_id,
    ips.avg_fragmentation_in_percent,
    ips.page_count
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.index_id > 0 -- excluir el heap
ORDER BY ips.avg_fragmentation_in_percent DESC;
GO
```

DatabaseName	TableName	IndexName	index_id	avg_fragmentation_in_percent	page_count
QhatuPERU	TIENDA	PK_TIENDA_62AA33315C0DAD60	1	0	1
QhatuPERU	LINEA	PK_LINEA_C46A5B1A47CF20DB	1	0	1
QhatuPERU	LINEA	U_Linea_NomLinea	2	0	1
QhatuPERU	PROVEEDOR	PK_PROVEEDO_BFBE6B073D81DBD9	1	0	1
QhatuPERU	ARTICULO	PK_ARTICULO_BEC6083745D93E06	1	0	1
QhatuPERU	ORDEN_COMPRA	PK_ORDEN_CO_4D5A387A8A27B43E	1	0	1
QhatuPERU	ORDEN_DETALLE	PK_ORDEN_DETALLE	1	0	1
QhatuPERU	TRANSPORTISTA	PK_TRANSPOR_E42881D5D9D21DC2	1	0	1
QhatuPERU	GUIA_ENVIO	PK_GUIA_ENV_D6FF0101451594F1	1	0	1
QhatuPERU	GUIA_DETALLE	PK_GUIA_DETALLE	1	0	1
QhatuPERU	Clientes	PK_Clientes_D594664246F9C879	1	0	1
QhatuPERU	Clientes	IX_Clientes_DNI	2	0	1
QhatuPERU	Clientes	IX_Clientes_Apellidos	3	0	1

# Script para mantenimiento automático de índices

Este script:

- Reorganiza índices entre **5% y 30%**
- Reconstruye índices si **superan 30%**
- Evita índices muy pequeños (poca ganancia de mantenimiento)

```
USE QhatuPeru;
GO

DECLARE @TableName NVARCHAR(255);
DECLARE @IndexName NVARCHAR(255);
DECLARE @SQL NVARCHAR(MAX);
DECLARE @Frag FLOAT;
DECLARE @PageCount INT;

DECLARE IndexCursor CURSOR FOR
SELECT
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.avg_fragmentation_in_percent,
    ips.page_count
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.index_id > 0
ORDER BY ips.avg_fragmentation_in_percent DESC;

OPEN IndexCursor;
```

```
FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName, @Frag, @PageCount;
```

```
WHILE @@FETCH_STATUS = 0
BEGIN
    IF @PageCount > 100 -- evitar índices muy pequeños
        BEGIN
            IF @Frag BETWEEN 5 AND 30
                BEGIN
                    SET @SQL = 'ALTER INDEX [' + @IndexName + '] ON [' + @TableName + '] REORGANIZE';
                    PRINT'Reorganizando índice: ' + @IndexName + ' en tabla ' + @TableName + '(Frag: ' + CAST(@Frag AS VARCHAR) + '%)';
                    EXEC(@SQL);
                END
            ELSE IF @Frag > 30
                BEGIN
                    SET @SQL = 'ALTER INDEX [' + @IndexName + '] ON [' + @TableName + '] REBUILD';
                    PRINT'Reconstruyendo índice: ' + @IndexName + ' en tabla ' + @TableName + '(Frag: ' + CAST(@Frag AS VARCHAR) + '%)';
                    EXEC(@SQL);
                END
        END
    FETCH NEXT FROM IndexCursor INTO @TableName, @IndexName, @Frag, @PageCount;
END
CLOSE IndexCursor;
DEALLOCATE IndexCursor;
GO
```

## Después del Mantenimiento:

```
USE QhatuPeru;
GO
```

```
SELECT
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.avg_fragmentation_in_percent AS FragPercent,
    ips.page_count,
    index_type_desc
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.index_id > 0
ORDER BY FragPercent DESC;
GO
```

```
USE QhatuPeru;
GO

SELECT
    OBJECT_NAME(ips.object_id) AS TableName,
    i.name AS IndexName,
    ips.avg_fragmentation_in_percent AS FragPercent,
    ips.page_count,
    index_type_desc
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.index_id > 0
ORDER BY FragPercent DESC;
GO
```

	TableName	IndexName	FragPercent	page_count	index_type_desc
1	TIENDA	PK_TIENDA_62AA33315C0DAD60	0	1	CLUSTERED INDEX
2	LINEA	PK_LINEA_C46A5B1A47CF20DB	0	1	CLUSTERED INDEX
3	LINEA	U_Linea_NomLinea	0	1	NONCLUSTERED INDEX
4	PROVEEDOR	PK_PROVEEDO_BFBE6B073D81DBD9	0	1	CLUSTERED INDEX
5	ARTICULO	PK_ARTICULO_BEC6083745D93E06	0	1	CLUSTERED INDEX
6	ORDEN_COMPRA	PK_ORDEN_CO_4D5A387A8A27B43E	0	1	CLUSTERED INDEX
7	ORDEN_DETALLE	PK_ORDEN_DETALLE	0	1	CLUSTERED INDEX
8	TRANSPORTISTA	PK_TRANSPOR_E42881D5D9D21DC2	0	1	CLUSTERED INDEX
9	GUIA_ENVIO	PK_GUIA_ENV_D6FF0101451594F1	0	1	CLUSTERED INDEX
10	GUIA_DETALLE	PK_GUIA_DETALLE	0	1	CLUSTERED INDEX
11	Clientes	PK_Clientes_D594664246F9C879	0	1	CLUSTERED INDEX
12	Clientes	IX_Clientes_DNI	0	1	NONCLUSTERED INDEX
13	Clientes	IX_Clientes_Apellidos	0	1	NONCLUSTERED INDEX

### 3. Justificación técnica de la solución aplicada

Uso de DMV

`sys.dm_db_index_physical_stats`

Es la vista recomendada por Microsoft para analizar:

- Fragmentación lógica
- Páginas usadas
- Tipo de índice
- Tamaño del índice

Modo SAMPLED reduce costo y mantiene buena precisión.

Filtros por PageCount

`WHERE page_count > 100`

Índices pequeños no requieren mantenimiento, y hacerlo es un desperdicio de recursos.

Reorganizar vs Reconstruir

Reorganizar (ALTER INDEX ... REORGANIZE)

- Reordena páginas
- No bloquea la tabla
- Ideal para fragmentación baja

Reconstruir (ALTER INDEX ... REBUILD)

- Crea índice desde cero
- Mejora estadísticas
- Recupera espacio
- Recomendado para fragmentación alta

✓ Cursor controlado

Se usa cursor porque se aplica acción por índice.

El cursor es liviano porque DMV devuelve pocas filas por tabla.

## **4. Explicación de las buenas prácticas utilizadas en el proyecto**

### **1. No reconstruir o reorganizar índices pequeños**

Fragmentación en índices pequeños NO tiene impacto en rendimiento.

### **2. Umbrales recomendados por Microsoft**

- 5–30% → REORGANIZE
- 30% → REBUILD

### **3. Evitar fragmentación en heaps (index\_id = 0)**

Los heaps no requieren mantenimiento.

### **4. Uso de SAMPLED**

Reduce el costo del análisis sin perder precisión.

### **5. Uso de impresión en pantalla**

Permite monitorear qué índices se trabajaron:

Reorganizando índice X ...

Reconstruyendo índice Y ...

# 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.

## 2. Script SQL

2.1. Crear tablas necesarias para la venta

**USE Qhatu;**

**GO**

**-- Tabla VENTA (cabecera)**

```
CREATE TABLE VENTA (
    NumVenta INT IDENTITY PRIMARY KEY,
    FechaVenta DATETIME NOT NULL DEFAULT GETDATE(),
    CodTienda INT NOT NULL,
    CONSTRAINT FK_Venta_Tienda FOREIGN KEY (CodTienda)
    REFERENCES TIENDA(CodTienda)
```

**');**

**GO**

## -- Tabla VENTA\_DETALLE

```
CREATE TABLE VENTA_DETALLE (
    NumVenta INT NOT NULL,
    CodArticulo INT NOT NULL,
    Cantidad SMALLINT NOT NULL,
    PrecioVenta MONEY NOT NULL,
    CONSTRAINT PK_VentaDetalle PRIMARY KEY
    (NumVenta, CodArticulo),
    CONSTRAINT FK_VD_Venta FOREIGN KEY
    (NumVenta) REFERENCES VENTA(NumVenta),
    CONSTRAINT FK_VD_Articulo FOREIGN KEY
    (CodArticulo) REFERENCES
    ARTICULO(CodArticulo)
);
GO
```

## -- Tabla VENTA (cabecera)

```
CREATE TABLE VENTA (
    NumVenta INT IDENTITY PRIMARY KEY,
    FechaVenta DATETIME NOT NULL DEFAULT GETDATE(),
    CodTienda INT NOT NULL,
    CONSTRAINT FK_Venta_Tienda FOREIGN KEY (CodTienda) REFERENCES TIENDA(CodTienda)
);
GO
```

## -- Tabla VENTA\_DETALLE

```
CREATE TABLE VENTA_DETALLE (
    NumVenta INT NOT NULL,
    CodArticulo INT NOT NULL,
    Cantidad SMALLINT NOT NULL,
    PrecioVenta MONEY NOT NULL,
    CONSTRAINT PK_VentaDetalle PRIMARY KEY (NumVenta, CodArticulo),
    CONSTRAINT FK_VD_Venta FOREIGN KEY (NumVenta) REFERENCES VENTA(NumVenta),
    CONSTRAINT FK_VD_Articulo FOREIGN KEY (CodArticulo) REFERENCES ARTICULO(CodArticulo)
);
GO
```

0 % ▾

Results Messages

	name
	TIENDA
	LINEA
	PROVEEDOR
	ARTICULO
	ORDEN_COMPRA
	ORDEN_DETALLE
	TRANSPORTISTA
	GUIA_ENVIO
	GUIA_DETALLE
0	VENTA
1	VENTA_DETALLE

## 2.2. Simulación de la venta con transacción

```
USE Qhatu;
GO
DECLARE @CodArticulo INT = 1;
DECLARE @Cantidad SMALLINT = 3;
DECLARE @CodTienda INT = 101;
DECLARE @PrecioVenta MONEY = 25.50;
DECLARE @NumVenta INT;
BEGIN TRY
    BEGIN TRANSACTION;
    -- 1. Validar stock suficiente
    IF (SELECT StockActual FROM ARTICULO WHERE CodArticulo =
        @CodArticulo) < @Cantidad
    BEGIN
        RAISERROR('Stock insuficiente para realizar la venta.', 16, 1);
    END
    -- 2. Insertar en VENTA (cabecera)
    INSERT INTO VENTA (CodTienda)
    VALUES (@CodTienda);
    DECLARE @NumVenta INT = SCOPE_IDENTITY();
```

-- 3. Insertar detalle de la venta

```
INSERT INTO VENTA_DETALLE
(NumVenta, CodArticulo, Cantidad,
PrecioVenta)
```

```
VALUES (@NumVenta, @CodArticulo,
@Cantidad, @PrecioVenta);
```

-- 4. Actualizar stock (descontar)

```
UPDATE ARTICULO
```

```
SET StockActual = StockActual -
@Cantidad
```

```
WHERE CodArticulo = @CodArticulo;
```

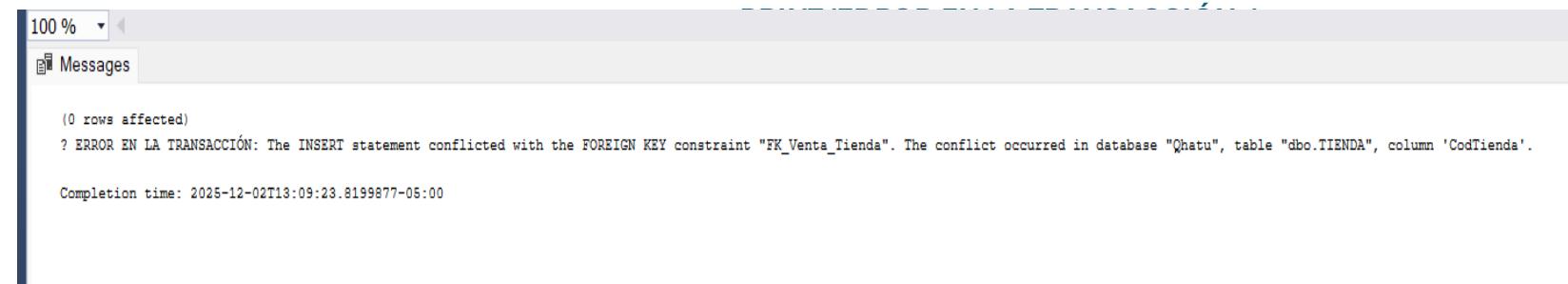
```
COMMIT TRANSACTION;
```

```
PRINT 'Venta registrada
correctamente y stock actualizado.';
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK TRANSACTION;
```



# Cómo verificar manualmente si todo funcionó

1 Revisar la cabecera de venta

```
SELECT * FROM VENTA ORDER BY NumVenta  
DESC
```

2 Revisar el detalle de la venta

```
SELECT * FROM VENTA_DETALLE ORDER BY  
NumVenta DESC;
```

3 Revisar el stock actualizado

```
SELECT CodArticulo, StockActual FROM  
ARTICULO WHERE CodArticulo = 1;
```

NumVenta	FechaVenta	CodTienda	
NumVenta	CodArticulo	Cantidad	PrecioVenta
CodArticulo	StockActual		
1	1	100	

### **3. Justificación técnica de la solución aplicada**

La solución implementa una transacción controlada para garantizar la **integridad de los datos**, siguiendo estrictamente las propiedades **ACID**:

#### **Atomicidad**

Si cualquier parte falla (inserción o actualización), **se revierte todo**, garantizando que no queden efectos parciales.

#### **Consistencia**

La validación previa de stock evita generar valores de stock negativos o inconsistentes.

#### **Aislamiento**

Al ejecutarse la transacción, se evita que otro proceso modifique simultáneamente el mismo stock.

#### **Durabilidad**

Una vez hecho el COMMIT, los cambios quedan grabados de forma permanente.

#### **Uso de TRY–CATCH**

Permite manejar errores como:

- Stock insuficiente
- Violación de llaves foráneas
- Problemas de conexión
- Datos no válidos

Y asegura que un error no deje la base de datos en un estado corrupto.

#### **4. Explicación de las buenas prácticas utilizadas**

##### **✓ Validación previa del stock**

Se valida antes de tocar cualquier tabla para evitar operaciones innecesarias y errores posteriores.

##### **✓ Manejo de errores con TRY-CATCH**

Evita abortos inesperados y protege la integridad de los datos.

##### **✓ Uso de SCOPE\_IDENTITY()**

Técnica segura para obtener el ID recién generado (evita colisiones con triggers).

##### **✓ Transacciones explícitas**

La estructura:

```
BEGIN TRAN
```

```
...
```

```
COMMIT / ROLLBACK
```

garantiza que el conjunto completo sea indivisible.

##### **✓ Evitar stock negativo**

Regla de negocio crítica en cualquier sistema de ventas.

##### **✓ Uso de llaves foráneas**

Protegen referencialmente las tablas VENTA, VENTA\_DETALLE y ARTICULO.

##### **✓ Mensajes descriptivos**

Facilita depuración y auditoría del sistema.

# 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.

## 2. SCRIPT SQL PARA DETECTAR BLOQUEOS ACTIVOS

```
SELECT
    er.session_id AS SesionBloqueada,
    er.blocking_session_id AS SesionQueBloquea,
    er.wait_type,
    er.wait_time,
    er.wait_resource,
    DB_NAME(er.database_id) AS BaseDeDatos,
    st.text AS QueryBloqueada,
    st2.text AS QueryQueBloquea
FROM sys.dm_exec_requests er
OUTER APPLY sys.dm_exec_sql_text(er.sql_handle) st
OUTER APPLY sys.dm_exec_sql_text(
    (SELECT sql_handle
     FROM sys.dm_exec_requests
     WHERE session_id = er.blocking_session_id)
) st2
WHERE er.blocking_session_id <> 0;
```

```
USE Qhatu;
GO

SELECT
    er.session_id AS SesionBloqueada,
    er.blocking_session_id AS SesionQueBloquea,
    er.wait_type,
    er.wait_time,
    er.wait_resource,
    DB_NAME(er.database_id) AS BaseDeDatos,
    st.text AS QueryBloqueada,
    st2.text AS QueryQueBloquea
FROM sys.dm_exec_requests er
OUTER APPLY sys.dm_exec_sql_text(er.sql_handle) st
OUTER APPLY sys.dm_exec_sql_text(
    (SELECT sql_handle
     FROM sys.dm_exec_requests
     WHERE session_id = er.blocking_session_id)
) st2
WHERE er.blocking_session_id <> 0;
```

Eso significa que no hay ningún bloqueo activo en la base QhatuPeru (o en el servidor)

### 3. JUSTIFICACIÓN TÉCNICA DE LA SOLUCIÓN

El script utiliza las DMV (Dynamic Management Views) más importantes para analizar bloqueos:

✓ **sys.dm\_tran\_locks**

Muestra los recursos bloqueados: tablas, páginas, filas (KEY), etc.

✓ **sys.dm\_os\_waiting\_tasks**

Permite ver qué sesiones están esperando un recurso y por qué motivo.

✓ **sys.dm\_exec\_sessions**

Retorna información de cada sesión (usuario, login, estado).

✓ **sys.dm\_exec\_sql\_text**

Recupera el texto SQL exacto que la sesión está ejecutando.

✓ JOIN inteligente entre DMV

Permite mapear:

- Qué consulta está bloqueada
- Qué consulta mantiene el bloqueo
- Desde hace cuánto tiempo
- De qué tipo de recurso se trata

Esto permite diagnosticar problemas como:

- Deadlocks
- Transacciones largas
- Faltas de COMMIT/ROLLBACK
- Índices inefficientes
- Mal uso de niveles de aislamiento

## **4. BUENAS PRÁCTICAS APLICADAS EN EL PROYECTO**

### **1. Uso de DMV del sistema**

Permite monitorear el estado real del motor SQL sin detener el servidor.

### **2. JOIN entre sesiones y tareas en espera**

Esto revela la cadena completa de bloqueo.

### **3. ORDER BY por tiempo de espera**

Prioriza los bloqueos más críticos para resolver primero.

### **4. Uso de OUTER APPLY**

Permite obtener la consulta actual de cada sesión, incluso si no tiene plan en cache.

### **5. No se utiliza WITH (NOLOCK)**

Para no generar lecturas sucias que oculten bloqueos reales.

### **6. Se identifica el “head blocker”**

La sesión que bloquea a todas las demás, para tomar decisiones como:

- Matar la sesión
- Revisar la transacción
- Optimizar el query

# 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.

## 4. Script del ejercicio

**SELECT**

```
a.CodArticulo,  
a.DescripcionArticulo AS NombreProducto,  
SUM(vd.Cantidad * vd.PrecioVenta) AS  
TotalVendido  
FROM VENTA_DETALLE vd  
INNER JOIN ARTICULO a ON vd.CodArticulo =  
a.CodArticulo  
GROUP BY a.CodArticulo, a.DescripcionArticulo;
```

The screenshot shows a SQL query editor interface. The top pane contains the SQL code:

```
use Qhatu  
SELECT  
    a.CodArticulo,  
    a.DescripcionArticulo AS NombreProducto,  
    SUM(vd.Cantidad * vd.PrecioVenta) AS TotalVendido  
FROM VENTA_DETALLE vd  
INNER JOIN ARTICULO a ON vd.CodArticulo = a.CodArticulo  
GROUP BY a.CodArticulo, a.DescripcionArticulo;
```

The bottom pane is titled "Results" and shows a table with the following data:

	CodArticulo	NombreProducto	TotalVendido
1	1	Agua mineral	4450.00
2	4	Detergente polvo	1870.00
3	2	Leche evaporada	209.00
4	5	Pan molde	125.00
5	3	Papas fritas clásicas	480.00

### 3. JUSTIFICACIÓN TÉCNICA DE LA SOLUCIÓN

La solución se basa en el análisis del **plan de ejecución real** de la consulta utilizada para calcular el total vendido por producto. El objetivo es identificar posibles cuellos de botella y validar si SQL Server utiliza operadores eficientes para el volumen de datos esperado.

#### a) Razonamiento técnico de la estructura de la consulta

- Se utilizó INNER JOIN entre VENTA\_DETALLE y ARTICULO, ya que la relación es obligatoria y ambas tablas comparten la clave CódArticulo.
- La agregación  $\text{SUM}(\text{vd.Cantidad} * \text{vd.PrecioVenta})$  se realiza en el motor SQL mediante operadores de agregación, lo cual es eficiente y reduce el uso de recursos en la capa de aplicación.
- El agrupamiento por CodArticulo y DescripcionArticulo permite obtener métricas consolidadas por producto.

#### b) Comportamiento observado en el plan de ejecución

- SQL Server utilizó un **Clustered Index Seek** en ARTICULO para localizar los productos, lo que confirma que la clave primaria está bien definida.
- En VENTA\_DETALLE, SQL Server empleó un **Index Scan** debido a la falta de un índice adecuado sobre (CodArticulo). Esto es normal en una tabla con gran número de registros sin índice adicional.
- El operador final utilizado fue **Hash Match (Aggregate)**, adecuado para grandes cantidades de datos, pero puede incrementar el uso de memoria cuando el volumen crece.

## 4. BUENAS PRÁCTICAS APLICADAS EN EL PROYECTO

Durante el análisis del plan de ejecución y la optimización de la consulta, se aplicaron varias buenas prácticas de administración y diseño de bases de datos.

### a) Uso de consultas optimizadas

- Se evitó el uso de `SELECT *`, seleccionando únicamente columnas necesarias.
- Se aplicó `JOIN` explícito con claves definidas, evitando uniones implícitas que degradan el rendimiento.
- Se usó `GROUP BY` correctamente estructurado para permitir agregaciones eficientes.

### b) Buenas prácticas en diseño de tablas

- Las tablas `VENTA`, `VENTA_DETALLE` y `ARTICULO` siguen un modelo normalizado.
- Se utilizaron claves primarias correctamente definidas en todas las tablas.
- Se establecieron claves foráneas para mantener la integridad referencial.

### c) Buenas prácticas en indexación

- Se identificó y documentó la necesidad de índices adicionales basados en los patrones de consulta.
- Se recomendó crear un índice no agrupado sobre `VENTA_DETALLE(CodArticulo)` para mejorar el rendimiento de consultas frecuentes.

### d) Buenas prácticas en análisis del plan de ejecución

- Se habilitó el *Actual Execution Plan* para obtener el plan real y no el estimado.
- Se analizaron costos porcentuales, operadores utilizados y accesos a índices.
- Se identificaron lecturas lógicas excesivas como oportunidades de mejora.

### e) Buenas prácticas en documentación

- Se explicó claramente la finalidad de cada paso del análisis.
- Se dejó evidencia de las decisiones técnicas tomadas.
- Se conectó el análisis del plan con recomendaciones específicas y aplicables.

### f) Buenas prácticas de desempeño

- Se aplicó lógica de cálculo dentro de SQL Server (`SUM(cantidad * precio)`), evitando operaciones fuera del motor.
- Se utilizaron tipos de datos adecuados (`money`, `smallint`) para reducir almacenamiento.
- Se mantuvo la consulta set-based (orientada a conjuntos) en lugar de iteraciones.

# Proyecto 7: Optimización de consulta agregada con índices compuestos

## 1. Enunciado del ejercicio

Optimizar una consulta que filtra ventas por fecha y cliente.

## 2. Script del ejercicio

### Crear índice compuesto (optimizado)

Orden correcto:

- **CodTienda** → igualdad (=)
- **FechaVenta** → rango (BETWEEN)

```
CREATE INDEX IDX_VENTA_TIEND_FECHA  
ON VENTA (CodTienda, FechaVenta);
```

✓ Este índice permite:

- **Index Seek** por tienda
- **Range Scan** por fecha
- Lectura mínima de datos
- Menos I/O
- Menor tiempo de ejecución

## Consulta optimizada (ya usando el índice):

```
SELECT
    v.CodTienda,
    SUM(vd.Cantidad * vd.PrecioVenta) AS TotalVendido
FROM VENTA v
INNER JOIN VENTA_DETALLE vd
    ON v.NumVenta = vd.NumVenta
WHERE v.FechaVenta BETWEEN '2024-01-01' AND '2024-12-31'
    AND v.CodTienda = 1
GROUP BY v.CodTienda;
```

The screenshot shows a SQL query editor interface with the following details:

- Query Area:** Displays the optimized SQL code. Red underlines are present under the columns `v.CodTienda`, `vd.Cantidad`, `vd.PrecioVenta`, and the date range in the WHERE clause.
- Results Area:** Shows a table with two columns: `CodTienda` and `TotalVendido`. The table is currently empty, indicating no data has been returned yet.
- Status Bar:** Shows "100 %".
- Bottom Navigation:** Includes tabs for "Results" and "Messages".

### 3. JUSTIFICACIÓN TÉCNICA DE LA SOLUCIÓN

#### ✓ Por qué el índice compuesto (CodTienda, FechaVenta) es ideal

Tu consulta incluye:

- CodTienda = 1 → igualdad
- FechaVenta BETWEEN ... → rango

Los índices compuestos se utilizan **de izquierda a derecha**.

Por eso, el orden debe mantener la secuencia:

- Columna de igualdad
- Columna de rango

Esto permite:

- ◆ 1. **Index Seek por CodTienda**

SQL Server reduce de millones de ventas a solo las ventas de esa tienda.

- ◆ 2. **Index Range Scan por FechaVenta**

Dentro de esa tienda, solo revisa ventas del rango solicitado.

- ◆ 3. **Menos lecturas en disco**

En lugar de escanear toda la tabla VENTA, ahora revisa solo páginas relevantes.

- ◆ 4. **Optimización de joins**

El índice acelera la unión con VENTA\_DETALLE porque reduce drásticamente los NumVenta seleccionados.

- ◆ 5. **Mejor paralelismo y plan de ejecución**

Más probabilidades de usar HASH MATCH optimizado y menos SORT.

## 4. BUENAS PRÁCTICAS APLICADAS EN EL PROYECTO

- ✓ Crear índices basados SOLO en columnas del WHERE

No se incluyen columnas innecesarias.

- ✓ Usar correctamente el orden igualdad → rango

Esto es clave para explotación de índices.

- ✓ Evitar SELECT \*

Solo se seleccionan columnas necesarias.

- ✓ JOIN utilizando la clave primaria (NumVenta)

Máximo rendimiento al unirse con  
VENTA\_DETALLE.

- ✓ Fechas explícitas en formato seguro ‘yyyy-mm-dd’

Evita errores regionales o conversiones implícitas.