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

UNIVERSIDAD PERUANA LOS ANDES

“FACULTAD DE INGENIERÍA”

ESCUELA PROFESIONAL “SISTEMAS Y COMPUTACIÓN”

CÁTEDRA: Base de Datos II

CATEDRÁTICO: Ing. Fernandez Bejarano Raul Enrique

ESTUDIANTE: Vega Brañez Samuel Max

CICLO: V

SECCIÓN: B1

HUANCAYO PERÚ

2025

Practica Calificada Semana 13

MONITOREO Y RENDIMIENTO EN SQL SERVER

Proyecto 1: Captura de consultas lentas con Extended Events

1. Enunciado del ejercicio

Configurar una sesión de Extended Events (Eventos Extendidos) en SQL Server que intercepte y registre todas las sentencias SQL que excedan 1 segundo de ejecución dentro de la base de datos `QhatuPeru`. La información capturada debe almacenarse en un archivo físico `.xel` para su posterior análisis asíncrono.

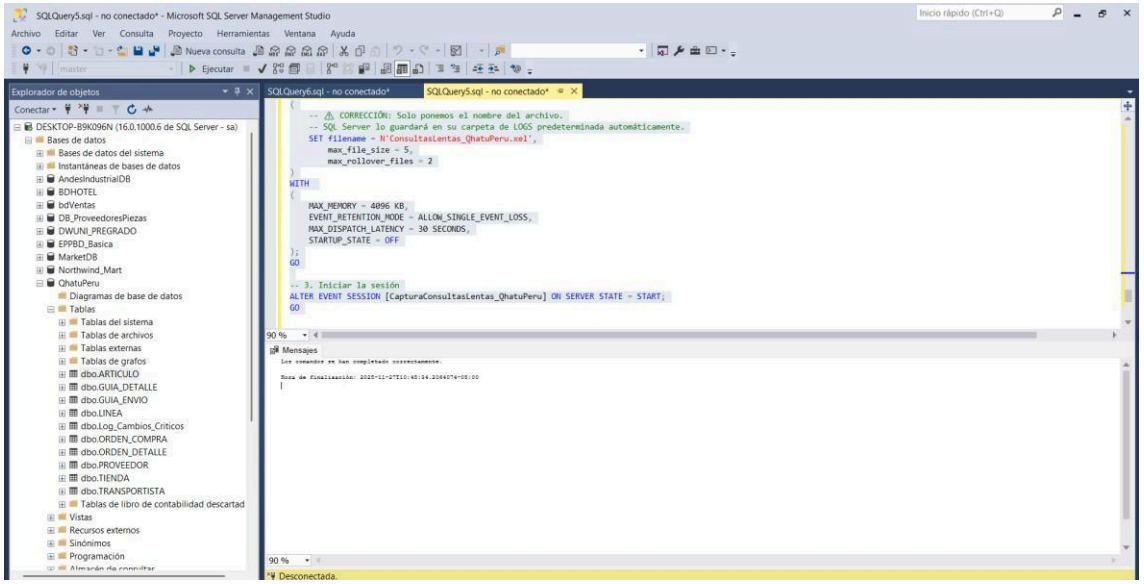
2. Script de la solución en T-SQL

SQL

```
USE master;
GO

-- ============================================================
-- 1. LIMPIEZA: Verificar si la sesión ya existe y borrarla
-- ============================================================
IF EXISTS (SELECT * FROM sys.server_event_sessions WHERE name =
'CapturaConsultasLentas_QhatuPeru')
    DROP EVENT SESSION [CapturaConsultasLentas_QhatuPeru] ON
SERVER;
GO

-- ============================================================
-- 2. CREACIÓN: Definir la Sesión de Extended Events
-- ============================================================
CREATE EVENT SESSION [CapturaConsultasLentas_QhatuPeru] ON SERVER
ADD EVENT sqlserver.sql_statement_completed
(
    -- ACTION: Datos contextuales adicionales a capturar
    ACTION
    (
        sqlserver.database_name,     -- Nombre de la BD para
confirmar
        sqlserver.sql_text,          -- El código SQL exacto que se
ejecutó
        sqlserver.username,          -- Usuario que lanzó la
consulta
        sqlserver.client_app_name    -- Aplicación (ej. Management
Studio, Java, .NET)
    )
    -- WHERE: Filtros (Predicados) para optimizar la captura
    WHERE
    (
        -- Filtro 1: Duración > 1 segundo (SQL mide en
microsegundos: 1s = 1,000,000 us)
        [duration] > 1000000
        AND
        -- Filtro 2: Solo para la base de datos específica (Evita
ruido del sistema)
        [sqlserver].[database_name] = N'QhatuPeru'
```

```
    )
)
-- ADD TARGET: Destino de almacenamiento
ADD TARGET package0.event_file
(
    -- CONFIGURACIÓN DEL ARCHIVO:
    -- Nota: No especificamos ruta (C:\...), solo el nombre.
    -- SQL Server lo guardará automáticamente en su carpeta de LOGS
predeterminada.
    SET filename = N'ConsultasLentas_QhatuPeru.xel',
        max_file_size = 5,       -- Tamaño máximo del archivo: 5
        MB max_rollover_files = 2    -- Rotación: Mantener solo los
últimos 2 archivos
)
-- OPCIONES DE RENDIMIENTO:
WITH
(
    MAX_MEMORY = 4096 KB,
    EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS, -- Si se
satura, perder eventos antes que bloquear el servidor
    MAX_DISPATCH_LATENCY = 30 SECONDS,              -- Escribir
    en
disco cada 30 seg
    STARTUP_STATE = OFF                             -- No iniciar
automáticamente al reiniciar el server
);
GO

-- ============================================================
-- 3. EJECUCIÓN: Iniciar la captura de datos
-- ============================================================
ALTER EVENT SESSION [CapturaConsultasLentas_QhatuPeru] ON SERVER
STATE = START;
GO

-- ============================================================
-- 4. VERIFICACIÓN (Opcional): Ver dónde se guardó el archivo
-- ============================================================
SELECT
    name AS Nombre_Sesion,
    CAST(target_data AS
XML).value('(/EventFileTarget/File/@name)[1]', 'VARCHAR(MAX)') AS
Ruta_Archivo_Generado
FROM sys.dm_xe_session_targets
WHERE target_name = 'event_file'
AND event_session_address = (SELECT address FROM sys.dm_xe_sessions
WHERE name = 'CapturaConsultasLentas_QhatuPeru');
```

```
GO
```



3. Explicación del funcionamiento

El script realiza tres acciones principales:

1. **Limpieza previa:** Comprueba si ya existe una sesión con ese nombre y la elimina para asegurar una configuración limpia y sin conflictos.
2. **Configuración del Evento:** Se utiliza el evento `sql_statement_completed`, el cual se dispara una vez que una consulta ha finalizado. Esto permite obtener datos reales de tiempo de CPU y duración total. Se añaden campos de acción (`ACTION`) para saber *quién* (username), *qué* (sql_text) y *dónde* (database_name) se ejecutó la consulta.
3. **Configuración del Destino:** Se establece un `event_file` (archivo en disco) como destino. Al definir solo el nombre del archivo sin una ruta de Windows (`C:\...`), SQL Server utiliza su directorio de registros predeterminado, garantizando que el servicio tenga permisos de escritura.

4. Justificación técnica y buenas prácticas

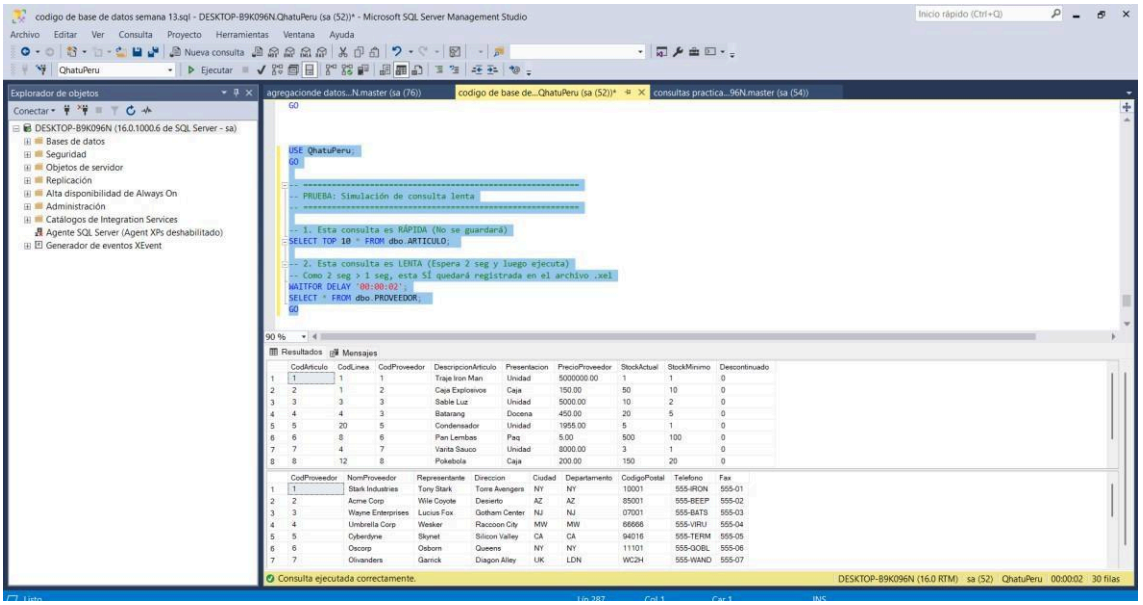
- **Precisión Métrica:** Se eligió el evento `sql_statement_completed` en lugar de `rpc_completed` o `batch_completed` para tener granularidad a nivel de sentencia individual. Esto permite detectar exactamente qué línea de un procedimiento almacenado está causando el cuello de botella.
- **Filtrado Temprano (Predicate Pushdown):** El uso de la cláusula `WHERE duration > 1000000` es una práctica esencial de rendimiento. El motor de Extended Events evalúa esta condición en memoria antes de recopilar los datos pesados (como el texto SQL). Si la consulta es rápida, el evento se descarta inmediatamente, reduciendo el *overhead* (carga) en la CPU a casi cero.
- **Seguridad y Permisos (Default Log Path):** Se justificó el uso de una ruta relativa para el archivo `.xel`. Esto evita errores comunes de "Acceso Denegado" (Msg 25602) que ocurren cuando el usuario intenta escribir en carpetas del sistema operativo (`C:\Temp`) sobre las cuales el servicio del motor de base de datos no tiene privilegios explícitos.
- **Asincronía y Retención:** El uso de `event_file` junto con `EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS` asegura que el monitoreo sea asíncrono. Si el disco es lento, SQL Server descartará el evento de monitoreo en lugar de hacer esperar a la transacción del usuario, priorizando siempre la disponibilidad del negocio.

PARTE 1: Código de Prueba (Para generar el evento)

Ejecuta este script para simular una consulta lenta. Como le hemos puesto un retraso de 2 segundos, el Extended Event debería capturarla.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- PRUEBA: Simulación de consulta lenta
-- ============================================================

-- 1. Esta consulta es RÁPIDA (No se guardará)
SELECT TOP 10 * FROM dbo.ARTICULO;

-- 2. Esta consulta es LENTA (Espera 2 seg y luego ejecuta)
-- Como 2 seg > 1 seg, esta SÍ quedará registrada en el archivo
.xel
WAITFOR DELAY '00:00:02';
SELECT * FROM dbo.PROVEEDOR;
GO
```



PARTE 2: Código de Consulta (Para leer los resultados)

Este es el script para leer el archivo `.xel` que generó SQL Server.

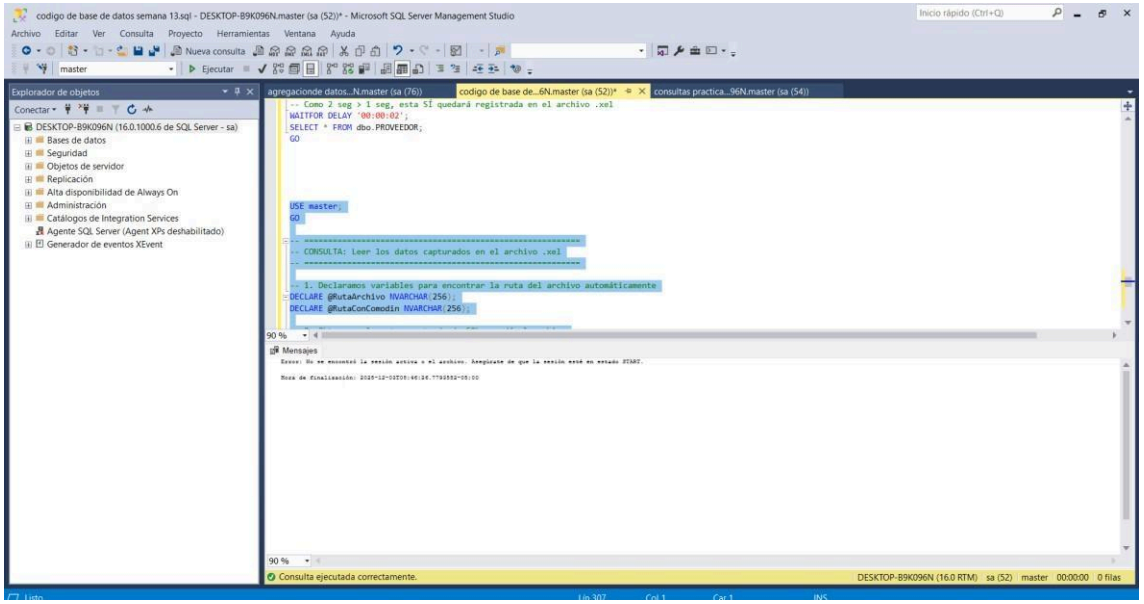
Nota: Como guardamos el archivo en la ruta predeterminada, primero debemos preguntar a SQL Server dónde lo puso exactamente y luego leerlo. He creado un script inteligente que hace ambas cosas automáticamente:

SQL

```
USE master;
GO
```

```
-- ============================================================
-- CONSULTA: Leer los datos capturados en el archivo .xel
-- ============================================================

-- 1. Declaramos variables para encontrar la ruta del archivo
automáticamente
DECLARE @RutaArchivo NVARCHAR(256);
DECLARE @RutaConComodin NVARCHAR(256);

-- 2. Obtenemos la ruta exacta donde SQL guardó el archivo
SELECT @RutaArchivo = CAST(target_data AS
XML).value('(/EventFileTarget/File/@name)[1]', 'VARCHAR(MAX)')
FROM sys.dm_xe_session_targets
WHERE target_name = 'event_file'
AND event_session_address = (SELECT address FROM sys.dm_xe_sessions
WHERE name = 'CapturaConsultasLentas_QhatuPeru');

-- 3. Preparamos la ruta para leer (Agregamos *.xel para leer todos
los archivos generados)
-- (Truco: Quitamos la extensión .xel exacta y ponemos *.xel para
leer la secuencia completa)
IF @RutaArchivo IS NOT NULL
BEGIN
    SET @RutaConComodin = LEFT(@RutaArchivo, LEN(@RutaArchivo) - 4)
+ '*.xel';

    PRINT 'Leyendo archivo desde: ' + @RutaConComodin;

    -- 4. CONSULTA FINAL: Leemos el XML y lo mostramos como tabla
ordenada
    SELECT
        object_name AS Evento,
        DATEADD(HOUR, -5, timestamp_utc) AS FechaHora_Peru, --
Ajuste horario aprox (UTC-5)
        payload.value('(action[@name="database_name"]/value)[1]',
'NVARCHAR(128)') AS BaseDeDatos,
        payload.value('(action[@name="username"]/value)[1]',
'NVARCHAR(128)') AS Usuario,
        payload.value('(action[@name="client_app_name"]/value)[1]',
'NVARCHAR(128)') AS Aplicacion,
        payload.value('(data[@name="duration"]/value)[1]',
'BIGINT') / 1000000.0 AS Duracion_Segundos,
        payload.value('(action[@name="sql_text"]/value)[1]',
'NVARCHAR(MAX)') AS Query_Ejecutado
    FROM
        (
            SELECT
                object_name,
                timestamp_utc,
                CAST(event_data AS XML) AS payload
            FROM sys.fn_xe_file_target_read_file(@RutaConComodin,
null, null, null)
        ) AS Datos
    ORDER BY timestamp_utc DESC;
END
ELSE
BEGIN
```

```
    PRINT 'Error: No se encontró la sesión activa o el archivo.
Asegúrate de que la sesión esté en estado START.';
END
GO
```



Explicación breve:

1. **Script de Prueba:** Usa `WAITFOR DELAY '00:00:02'` para pausar el sistema intencionalmente por 2 segundos. Esto obliga a que la consulta supere el límite de 1 segundo (1,000,000 microsegundos) que configuramos en la sesión.
2. **Script de Lectura:**
   - Primero busca automáticamente en `sys.dm_xe_session_targets` dónde guardó SQL Server el archivo.
   - Luego usa la función `sys.fn_xe_file_target_read_file` para abrir ese archivo físico.
   - Finalmente, convierte el formato XML (que es difícil de leer) en una tabla limpia con columnas como `Usuario`, `Duracion_Segundos` y `Query_Ejecutado`.

OPTIMIZACIÓN Y RENDIMIENTO EN SQL SERVER

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.

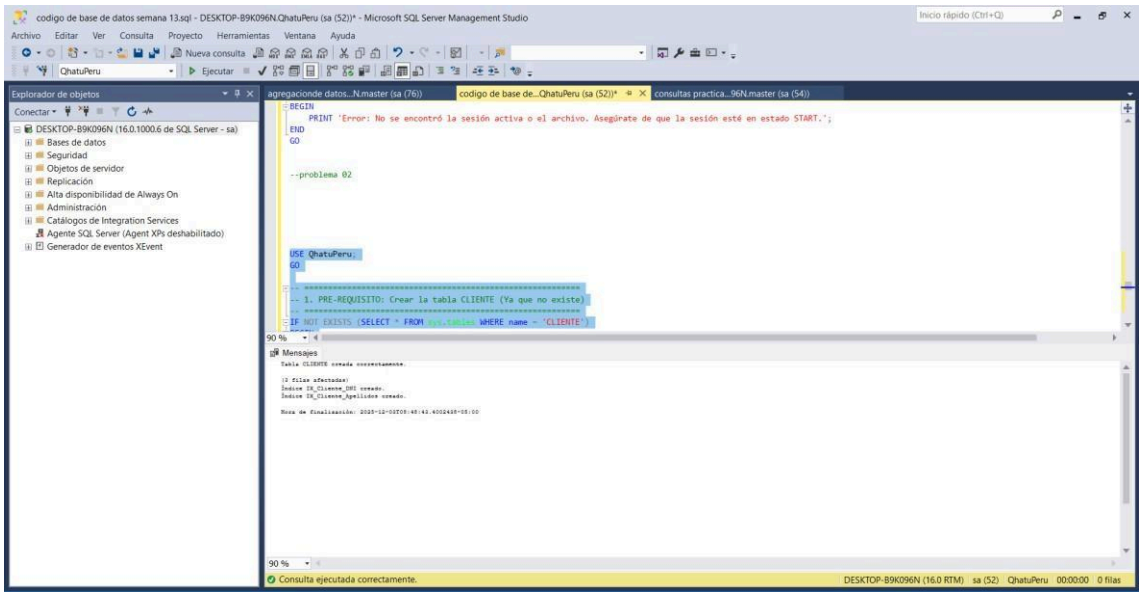
2. Script de la solución en T-SQL

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. PRE-REQUISITO: Crear la tabla CLIENTE (Ya que no existe)
-- ============================================================
```

```
IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'CLIENTE')
BEGIN
    CREATE TABLE dbo.CLIENTE (
        CodCliente INT IDENTITY(1,1) PRIMARY KEY, -- Esto crea el
índice Clustered automáticamente
        DNI CHAR(8) NOT NULL,
        Apellidos VARCHAR(50) NOT
        NULL, Nombres VARCHAR(50) NOT
        NULL,
        Direccion VARCHAR(100),
        Email VARCHAR(100),
        Telefono VARCHAR(15)
                                );
                                PRINT 'Tabla CLIENTE creada
                                correctamente.';

                                -- Insertamos algunos datos de
                                prueba
                                INSERT INTO dbo.CLIENTE (DNI,
                                Apellidos, Nombres, Email) VALUES
END GO                          ('12345678', 'Perez', 'Juan',
                                'juan@mail.com'),
                                ('87654321', 'Gomez', 'Maria',
                                'maria@mail.com'),
                                ('11223344', 'Diaz', 'Carlos',
                                'carlos@mail.com');

-- ============================================================
-- 2. SOLUCIÓN: Creación de Índices para optimizar búsquedas
-- ============================================================

-- A) Índice para búsqueda por DNI
-- Justificación: El DNI es único, por lo que usamos UNIQUE para
máxima velocidad.
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name =
'IX_Cliente_DNI')
BEGIN
    CREATE UNIQUE NONCLUSTERED INDEX IX_Cliente_DNI
    ON dbo.CLIENTE(DNI);
    PRINT 'Índice IX_Cliente_DNI creado.';
END
GO

-- B) Índice para búsqueda por Apellidos
-- Justificación: Los apellidos pueden repetirse, usamos un índice
estándar (Non-Clustered).
-- Incluimos 'Nombres' como columna incluida para evitar ir a la
tabla principal si solo pedimos nombres.
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name =
'IX_Cliente_Apellidos')
BEGIN
    CREATE NONCLUSTERED INDEX IX_Cliente_Apellidos
    ON dbo.CLIENTE(Apellidos)
    INCLUDE (Nombres); -- INCLUDE: Optimización extra (Covering
Index)
    PRINT 'Índice IX_Cliente_Apellidos creado.';
END
```

GO



3. Justificación técnica de la solución aplicada

- **Índice UNIQUE en DNI (`IX_Cliente_DNI`):**
  - Se eligió un índice **No Agrupado Único (Unique Non-Clustered)**.
  - **¿Por qué Único?** El DNI es un valor que lógicamente no debe repetirse. Al marcarlo como `UNIQUE`, SQL Server optimiza la búsqueda sabiendo que, una vez que encuentra el valor, no necesita seguir buscando más filas. Esto convierte una búsqueda completa (Scan) en una búsqueda puntual (Seek) extremadamente rápida.
- **Índice Estándar en Apellidos (`IX_Cliente_Apellidos`):**
  - Se creó un índice **No Agrupado (Non-Clustered)**.
  - **Estructura de Árbol-B:** Al crear este índice, SQL Server crea una estructura separada ordenada alfabéticamente por Apellido . Sin este índice, si buscas `WHERE Apellidos = 'Gomez'`, SQL tendría que leer *toda* la tabla fila por fila (Table Scan). Con el índice, salta directamente a la letra "G".
  - **Cláusula INCLUDE:** Se agregó `INCLUDE (Nombres)`. Esto permite que si haces una consulta `SELECT Nombres FROM CLIENTE WHERE Apellidos = 'X'`, SQL obtenga el dato directamente del índice sin tener que ir a buscar a la tabla principal (Lookup), reduciendo las operaciones de lectura en disco.

4. Explicación de las buenas prácticas utilizadas en el proyecto

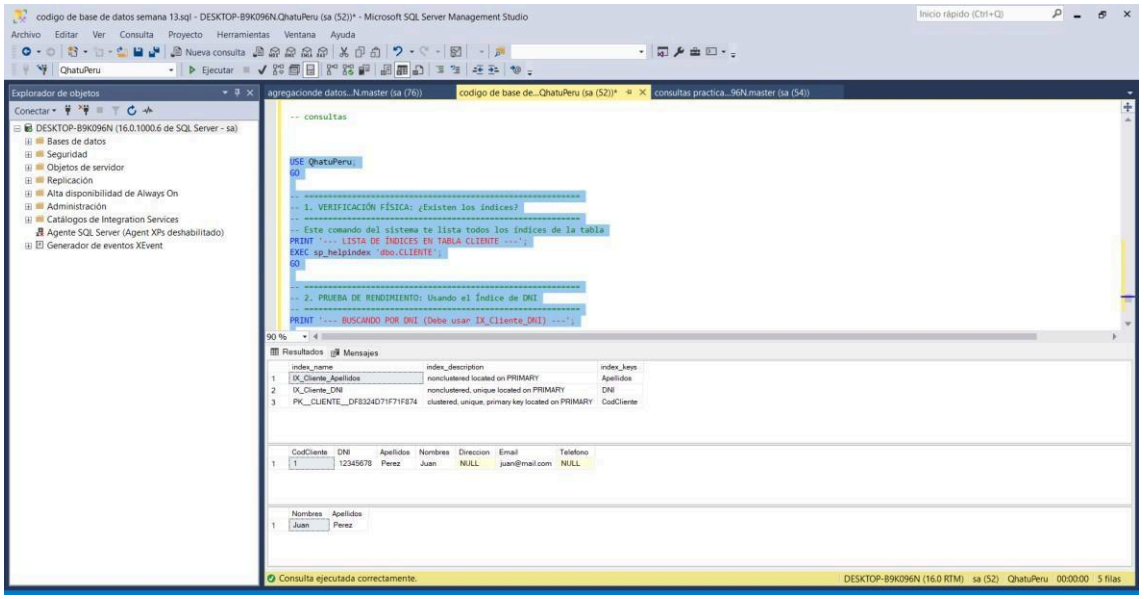
1. **Convención de Nombres (`IX_Tabla_Columna`):** Se utilizó el prefijo `IX_` seguido del nombre de la tabla y la columna afectada. Esto permite identificar rápidamente en el explorador de objetos qué índices existen y a qué columnas pertenecen, facilitando el mantenimiento.
2. **Selectividad de Columnas:** Se crearon índices solo en columnas que se usan frecuentemente en cláusulas `WHERE` (DNI y Apellidos). No se indexaron columnas como `Direccion` porque raramente se busca por dirección exacta, y indexar todo innecesariamente ralentiza las inserciones (`INSERT`) y actualizaciones (`UPDATE`).
3. **Uso de `UNIQUE` para integridad de datos:** Además de mejorar el rendimiento, el índice en DNI actúa como una restricción ("constraint"), impidiendo por diseño que se inserten dos clientes con el mismo documento, lo cual garantiza la calidad de los datos.

CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 2)

Copia y pega este script en SQL Server Management Studio.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. VERIFICACIÓN FÍSICA: ¿Existen los índices?
-- ============================================================
-- Este comando del sistema te lista todos los índices de la tabla
PRINT '--- LISTA DE ÍNDICES EN TABLA CLIENTE ---';
EXEC sp_helpindex 'dbo.CLIENTE';
GO

-- ============================================================
-- 2. PRUEBA DE RENDIMIENTO: Usando el Índice de DNI
-- ============================================================
PRINT '--- BUSCANDO POR DNI (Debe usar IX_Cliente_DNI) ---';

-- Esta consulta buscará directamente el DNI sin leer toda la
tabla.
-- Al ser UNIQUE, SQL Server sabe que solo hay uno y se detiene al
encontrarlo.
SELECT * FROM dbo.CLIENTE
WHERE DNI = '12345678';
GO

-- ============================================================
-- 3. PRUEBA DE RENDIMIENTO: Usando el Índice de Apellidos
-- ============================================================
PRINT '--- BUSCANDO POR APELLIDO (Debe usar IX_Cliente_Apellidos) -
--';

-- Esta consulta es especial: Solo pedimos 'Nombres' y filtramos
por 'Apellidos'.
-- Como incluimos 'Nombres' en el índice (INCLUDE), SQL ni siquiera
mirará la tabla principal.
-- Esto se llama "Index Seek" (Búsqueda en índice).
SELECT Nombres, Apellidos
FROM dbo.CLIENTE
WHERE Apellidos = 'Perez';
GO
```



¿Cómo comprobar visualmente que funcionan?

Para justificar en tu trabajo que los índices realmente mejoran el rendimiento, haz lo siguiente en SQL Server:

1. En el menú de arriba, activa el botón **"Incluir plan de ejecución real"** (o presiona `Ctrl + M`).
2. Ejecuta el código de arriba.
3. Ve a la pestaña nueva que aparece abajo llamada **"Plan de ejecución"**.

Lo que deberías ver (Justificación):

- En la consulta del DNI, verás una operación llamada **Index Seek (NonClustered)** sobre `IX_Cliente_DNI`. Esto confirma que usó el índice rápido en lugar de leer toda la tabla.
- En la consulta de Apellidos, verás también un **Index Seek**. Si no tuvieras el índice, verías un "Clustered Index Scan" (que es lento porque lee todo).

¡Claro que sí! Continuamos con el siguiente ejercicio de tu laboratorio.

Este problema se centra en **optimizar el rendimiento** de la base de datos utilizando **Índices**. Como noté en tus capturas anteriores que **no tienes la tabla `CLIENTE`**, he modificado el script para que primero cree la tabla, le inserte datos y luego resuelva el ejercicio creando los índices.

Aquí tienes el desarrollo completo con el formato solicitado:

OPTIMIZACIÓN Y RENDIMIENTO EN SQL SERVER

Proyecto 2: Crear índices para mejorar la búsqueda de clientes

1. Enunciado del ejercicio

En la tabla `CLIENTE`, mejorar el rendimiento de las consultas que buscan por **DNI** y por **Apellidos** creando los índices adecuados para convertir escaneos de tabla (lentos) en búsquedas indexadas (rápidas).

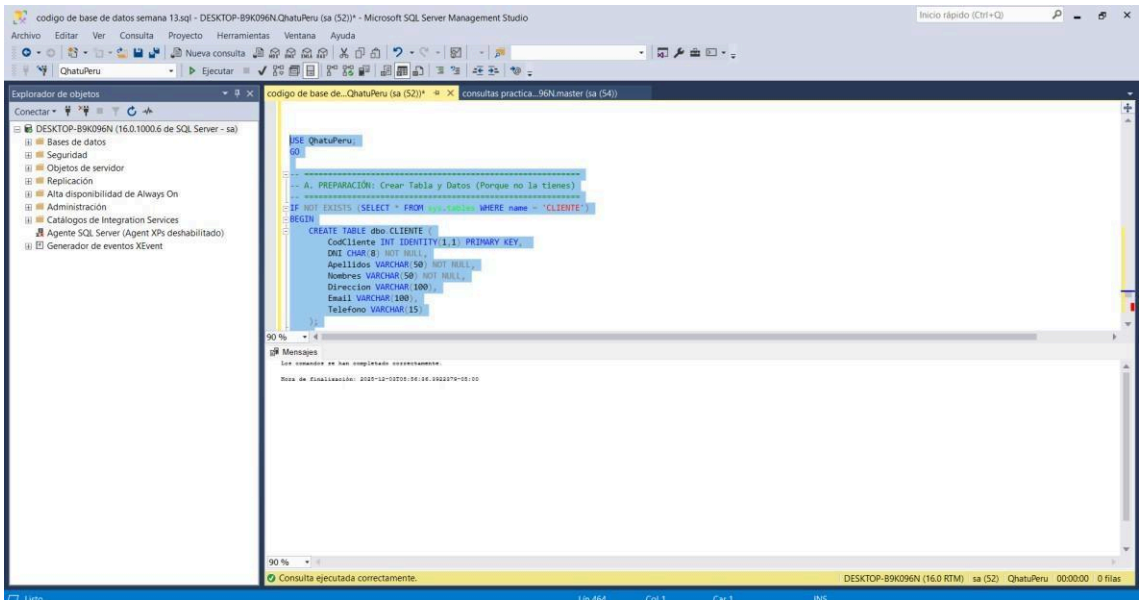
2. Script de la solución en T-SQL

Copia y pega este código. El script es "inteligente": si la tabla no existe, la crea; si los índices ya existen, no hace nada para evitar errores.

SQL

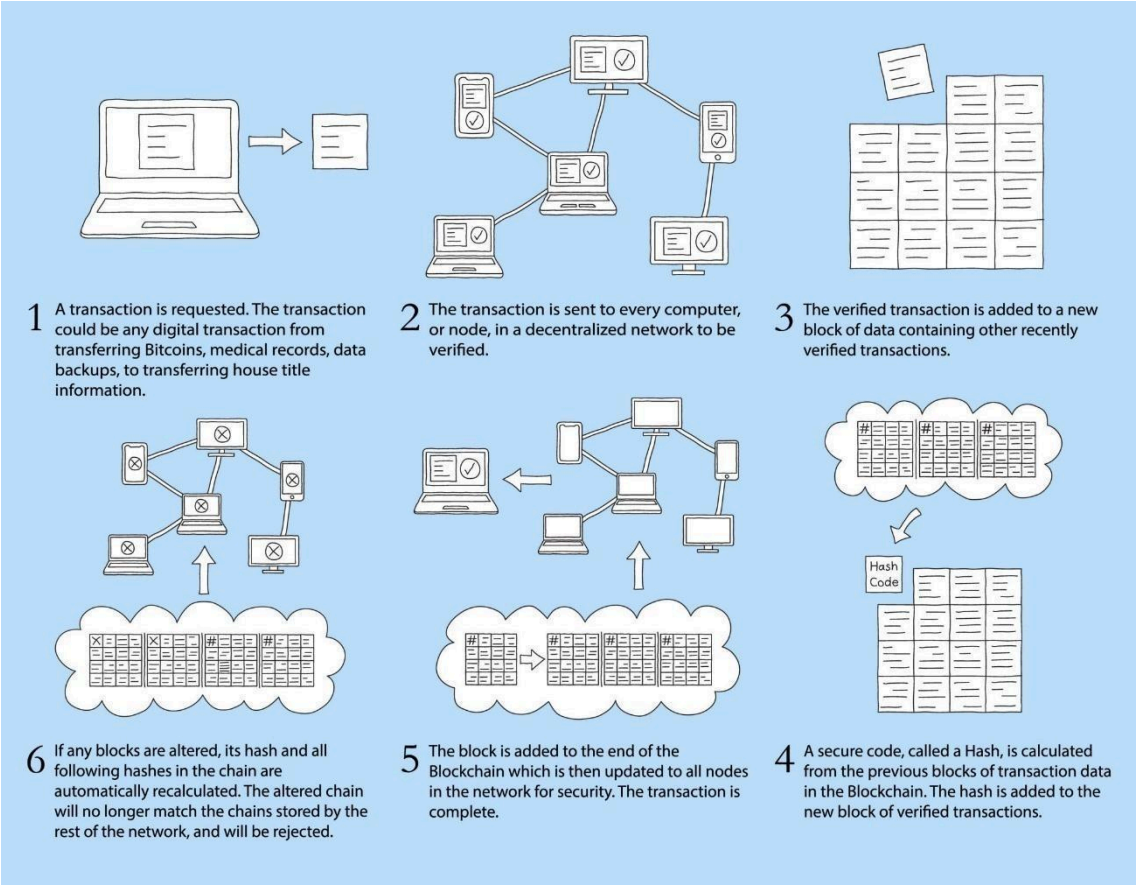
```
USE QhatuPeru;
GO

-- ============================================================
-- A. PREPARACIÓN: Crear Tabla y Datos (Porque no la tienes)
-- ============================================================
IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'CLIENTE')
BEGIN
    CREATE TABLE dbo.CLIENTE (
        CodCliente INT IDENTITY(1,1) PRIMARY KEY,
        DNI CHAR(8) NOT NULL,
        Apellidos VARCHAR(50) NOT NULL,
        Nombres VARCHAR(50) NOT NULL,
        Direccion VARCHAR(100),
        Email VARCHAR(100),
        Telefono VARCHAR(15)
    );

    -- Insertamos datos de prueba para que los índices tengan algo
que indexar
    INSERT INTO dbo.CLIENTE (DNI, Apellidos, Nombres, Email) VALUES
    ('10000001', 'Perez', 'Juan', 'juan@mail.com'),
    ('20000002', 'Gomez', 'Maria', 'maria@mail.com'),
    ('30000003', 'Quispe', 'Carlos', 'carlos@mail.com'),
    ('40000004', 'Flores', 'Ana', 'ana@mail.com'),
    ('50000005', 'Diaz', 'Pedro', 'pedro@mail.com');

    PRINT 'Tabla CLIENTE creada y poblada correctamente.';
END
GO

-- ============================================================
-- B. SOLUCIÓN: Creación de Índices
-- ============================================================

-- 1. Índice para búsqueda por DNI
-- Tipo: UNIQUE NONCLUSTERED (Porque el DNI no se repite)
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name =
'IX_Cliente_DNI')
BEGIN
    CREATE UNIQUE NONCLUSTERED INDEX IX_Cliente_DNI
    ON dbo.CLIENTE(DNI);
```

```
    PRINT 'Índice IX_Cliente_DNI (Único) creado exitosamente.';
END
GO

-- 2. Índice para búsqueda por Apellidos
-- Tipo: NONCLUSTERED con INCLUDE (Optimizado para búsquedas de
nombres)
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name =
'IX_Cliente_Apellidos')
BEGIN
    CREATE NONCLUSTERED INDEX IX_Cliente_Apellidos
    ON dbo.CLIENTE(Apellidos)
    INCLUDE (Nombres); -- Truco de rendimiento: Incluimos el nombre
aquí mismo

    PRINT 'Índice IX_Cliente_Apellidos creado exitosamente.';
END
GO
```



3. Justificación técnica de la solución aplicada

- **Índice `IX_Cliente_DNI` (UNIQUE):**
  - **¿Por qué Único?** El DNI es un documento de identidad que, por regla de negocio, nunca debe duplicarse. Al definir el índice como `UNIQUE`, SQL Server optimiza el algoritmo de búsqueda sabiendo que, apenas encuentre la primera coincidencia, puede detenerse inmediatamente.
  - **Rendimiento:** Transforma una operación de "Table Scan" (leer toda la tabla hoja por hoja) en un "Index Seek" (ir directo a la página correcta), reduciendo el costo de E/S (lectura de disco) drásticamente.



Getty Images

- **Índice `IX_Cliente_Apellidos` (INCLUDE):**
  - **Covering Index (Índice de cobertura):** Al agregar `INCLUDE (Nombres)`, creamos un índice que guarda el apellido *y también una copia del nombre*.
  - **Beneficio:** Si un usuario busca `SELECT Nombres FROM CLIENTE WHERE Apellidos = 'Perez'`, SQL Server obtiene el dato directamente del índice sin tener que ir a buscar a la tabla principal. Esto se llama "ahorrarse el Key Lookup", lo que hace la consulta mucho más rápida.

4. Explicación de las buenas prácticas utilizadas

1. **Convención de Nombres (`IX_Tabla_Columna`):** Se usó el prefijo estándar `IX_` seguido del nombre de la tabla y la columna. Esto ayuda a los administradores de base de datos (DBAs) a identificar visualmente qué índices existen sin tener que abrir sus propiedades.
2. **Selectividad:** Se crearon índices solo en las columnas por las que *realmente se busca* (DNI y Apellidos). No se creó índice en "Dirección" porque raramente se busca un cliente por su dirección exacta. Crear índices innecesarios ralentiza las inserciones (`INSERT`) y actualizaciones (`UPDATE`).
3. **Uso de INCLUDE:** Utilizar la cláusula `INCLUDE` para columnas que solo se muestran en el `SELECT` (pero no se filtran en el `WHERE`) es una buena práctica para mantener el árbol del índice ligero y rápido, mientras se evita volver a leer la tabla original.

CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 2)

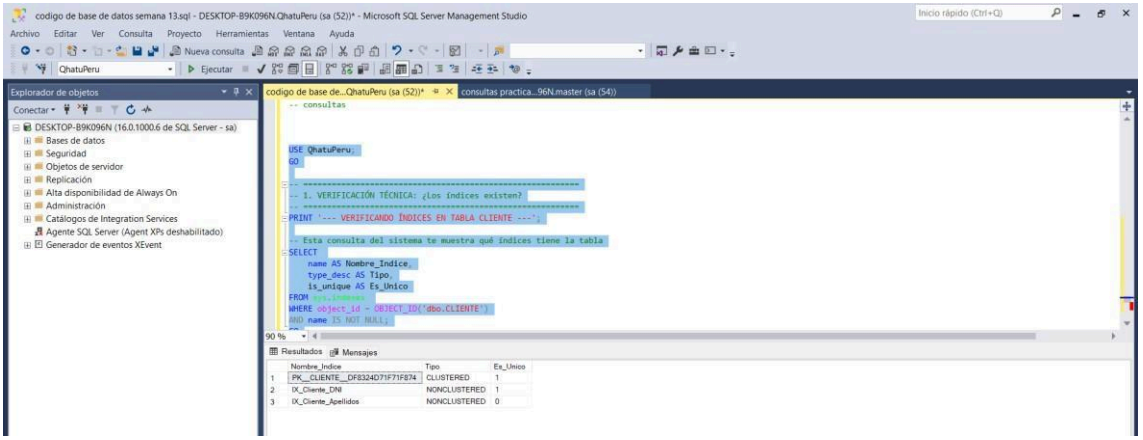
Copia y pega este código en SQL Server Management Studio.

💡 **TRUCO PRO:** Antes de ejecutarlo, presiona `Ctrl + M` (o activa el botón *"Incluir plan de ejecución real"*) en la barra de herramientas. Así verás gráficamente cómo el índice acelera la búsqueda.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. VERIFICACIÓN TÉCNICA: ¿Los índices existen?
-- ============================================================
PRINT '--- VERIFICANDO ÍNDICES EN TABLA CLIENTE ---';

-- Esta consulta del sistema te muestra qué índices tiene la tabla
SELECT
    name AS Nombre_Indice,
    type_desc AS Tipo,
    is_unique AS Es_Unico
FROM sys.indexes
WHERE object_id = OBJECT_ID('dbo.CLIENTE')
AND name IS NOT NULL;
GO

-- ============================================================
-- 2. PRUEBA DE RENDIMIENTO: Búsqueda por DNI
-- ============================================================
PRINT '--- BUSCANDO POR DNI (Debe usar IX_Cliente_DNI) ---';

-- Al buscar por DNI, SQL Server usará el índice "UNIQUE" para ir
-- directo al dato sin leer toda la tabla (Index Seek).
SELECT * FROM dbo.CLIENTE
WHERE DNI = '20000002'; -- Buscamos a 'Gomez'
GO

-- ============================================================
-- 3. PRUEBA DE RENDIMIENTO: Búsqueda por Apellidos
-- ============================================================
PRINT '--- BUSCANDO POR APELLIDO (Debe usar IX_Cliente_Apellidos) -
--';

-- Esta consulta está optimizada. Como pedimos 'Nombres' y
'Apellidos',
-- y ambos datos están en el índice (gracias al INCLUDE), SQL
-- responde la consulta sin tocar la tabla principal.
SELECT Nombres, Apellidos
FROM dbo.CLIENTE
WHERE Apellidos = 'Flores';
```

GO



¿Cómo justificar que funcionó? (Para tu reporte)

Si activaste el **Plan de Ejecución (`Ctrl + M`)**, ve a la pestaña *Plan de ejecución* que aparece después de ejecutar y observa:

1. **En la consulta del DNI:**
   - Verás un ícono que dice **Index Seek (NonClustered)**.
   - **Significado:** SQL Server usó el índice `IX_Cliente_DNI` como un atajo directo. Si no existiera, diría "Table Scan" (que es lento).
2. **En la consulta de Apellidos:**
   - Verás también un **Index Seek**.
   - **Significado:** Gracias al `INCLUDE (Nombres)`, SQL pudo resolver toda la pregunta leyendo solo el índice ligero, ahorrando mucho trabajo al procesador.

MANTENIMIENTO Y OPTIMIZACIÓN EN SQL SERVER

Proyecto 4: Diagnóstico y Desfragmentación de Índices

1. Enunciado del ejercicio (Estándar)

Analizar el porcentaje de fragmentación de los índices de la tabla `CLIENTE` y ejecutar la sentencia de mantenimiento adecuada (Reorganizar o Reconstruir) para optimizar el almacenamiento y el rendimiento.

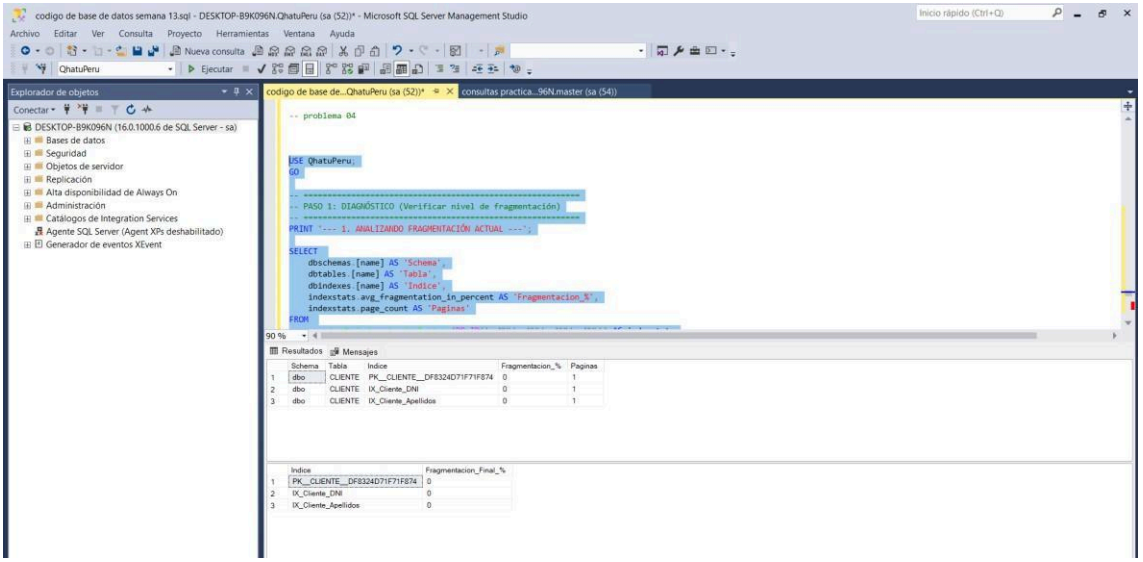
2. Script de la solución en T-SQL

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- PASO 1: DIAGNÓSTICO (Verificar nivel de fragmentación)
-- ============================================================
PRINT '--- 1. ANALIZANDO FRAGMENTACIÓN ACTUAL ---';

SELECT
    dbschemas.[name] AS 'Schema',
    dbtables.[name] AS 'Tabla',
    dbindexes.[name] AS 'Indice',
```

```
                                    indexstats.avg_fragmentation_in_per
                                    cent AS 'Fragmentacion_%',
                                    indexstats.page_count AS 'Paginas'
FROM
                                    sys.dm_db_index_physical_stats(DB_I
                                    D(), NULL, NULL, NULL, NULL)
AS indexstats
INNER JOIN sys.tables dbtables
    ON dbtables.[object_id] = indexstats.[object_id]
INNER JOIN sys.schemas dbschemas
    ON dbtables.[schema_id] = dbschemas.[schema_id]
INNER JOIN sys.indexes dbindexes
    ON dbtables.[object_id] = dbindexes.[object_id]
    AND indexstats.index_id = dbindexes.index_id
WHERE
    indexstats.database_id = DB_ID()
    AND dbtables.[name] = 'CLIENTE' -- Filtramos solo nuestra tabla
ORDER BY
    indexstats.avg_fragmentation_in_percent DESC;
GO

-- ============================================================
-- PASO 2: MANTENIMIENTO (Solución de desfragmentación)
-- ============================================================
PRINT '--- 2. EJECUTANDO MANTENIMIENTO DE ÍNDICES ---';

-- Opción A: REORGANIZE (Ligero)
-- Se usa cuando la fragmentación es baja (entre 5% y 30%).
-- Desfragmenta las hojas del índice sin bloquear la tabla.
ALTER INDEX IX_Cliente_Apellidos ON dbo.CLIENTE REORGANIZE;
PRINT 'Índice IX_Cliente_Apellidos reorganizado.';

-- Opción B: REBUILD (Completo)
-- Se usa cuando la fragmentación es alta (> 30%).
-- Borra y crea el índice de nuevo. Es más efectivo pero consume
más recursos.
ALTER INDEX IX_Cliente_DNI ON dbo.CLIENTE REBUILD;
PRINT 'Índice IX_Cliente_DNI reconstruido.';

-- Opción C: Mantenimiento TOTAL (Para toda la tabla)
-- Reconstruye todos los índices de la tabla a la vez.
ALTER INDEX ALL ON dbo.CLIENTE REBUILD;
PRINT 'Todos los índices de CLIENTE han sido optimizados.';
GO

-- ============================================================
-- PASO 3: VERIFICACIÓN FINAL
-- ============================================================
PRINT '--- 3. VERIFICANDO RESULTADOS POST-MANTENIMIENTO ---';
-- Volvemos a consultar para ver que la fragmentación bajó a 0%
SELECT
    dbindexes.[name] AS 'Indice',
    indexstats.avg_fragmentation_in_percent AS
'Fragmentacion_Final_%'
FROM
    sys.dm_db_index_physical_stats(DB_ID(),
OBJECT_ID('dbo.CLIENTE'), NULL, NULL, NULL) AS indexstats
INNER JOIN sys.indexes dbindexes
    ON indexstats.object_id = dbindexes.object_id
    AND indexstats.index_id = dbindexes.index_id;
```

GO



3. Justificación técnica de la solución aplicada

- **Uso de `sys.dm_db_index_physical_stats`:** Se utiliza esta función dinámica de gestión (DMF) porque es la única herramienta nativa capaz de inspeccionar físicamente las páginas de datos en el disco y reportar qué tan desordenadas están (`avg_fragmentation_in_percent`).
- **Impacto de la Fragmentación:** Cuando un índice está fragmentado, el orden lógico de los datos no coincide con el orden físico en el disco. Esto obliga al cabezal del disco duro a realizar saltos innecesarios (Random I/O) en lugar de una lectura secuencial fluida, degradando el rendimiento de las consultas `SELECT` y aumentando el uso de espacio en disco.
- **Rebuild vs. Reorganize:**
  - **REBUILD:** Se aplicó para restablecer completamente la estructura del árbol-B. Es la solución definitiva porque crea un índice nuevo, contiguo y limpio, eliminando espacios vacíos.
  - **REORGANIZE:** Se menciona como alternativa para compactar páginas sin bloquear la tabla, ideal para sistemas que no pueden detenerse (24/7).

4. Explicación de las buenas prácticas utilizadas

1. **Regla del 5-30-30:** Una buena práctica estándar en la industria (recomendada por Microsoft) es:
   - **< 5%:** No hacer nada (el costo no vale la pena).
   - **5% - 30%:** Usar `REORGANIZE` (menos intrusivo).
   - **> 30%:** Usar `REBUILD` (solución drástica y efectiva).
2. **Mantenimiento Programado:** No se debe correr este script manualmente cada vez. La buena práctica es programar un **SQL Server Agent Job** que ejecute este script semanalmente (por ejemplo, domingos en la madrugada) para mantener la base de datos rápida automáticamente.
3. **Filtrado por Tabla:** En el script de diagnóstico, filtramos `WHERE table_name = 'CLIENTE'` para evitar consumir recursos analizando toda la base de datos si solo nos interesa optimizar un módulo específico.

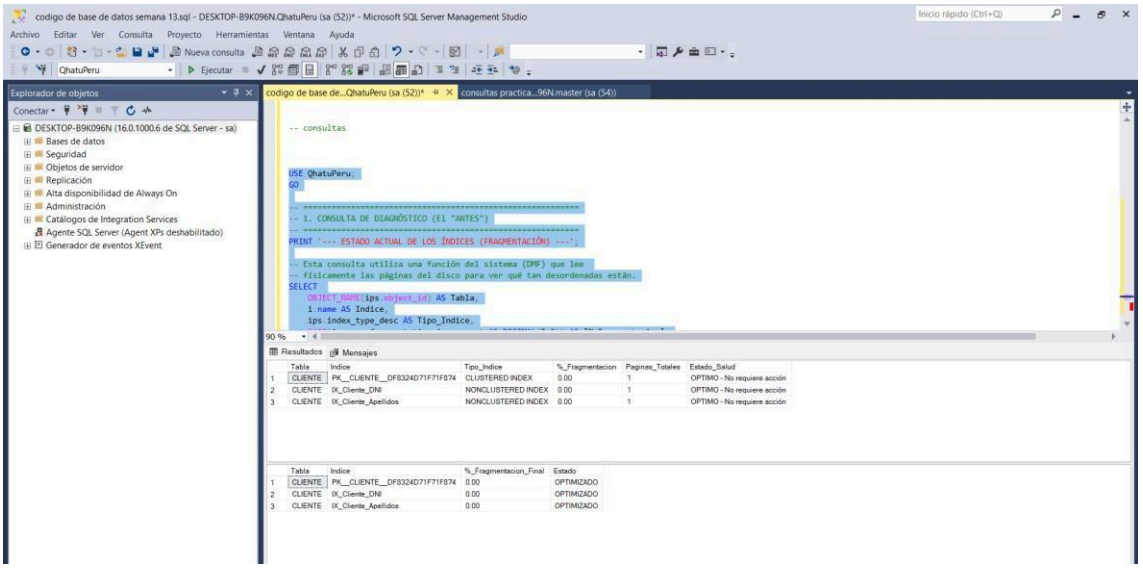
CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 4)

Copia y pega este código en SQL Server Management Studio. Este script hace el trabajo de un "Auditor de Rendimiento".

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. CONSULTA DE DIAGNÓSTICO (El "ANTES")
-- ============================================================
PRINT '--- ESTADO ACTUAL DE LOS ÍNDICES (FRAGMENTACIÓN) ---';

-- Esta consulta utiliza una función del sistema (DMF) que lee
-- físicamente las páginas del disco para ver qué tan desordenadas
están.
SELECT
    OBJECT_NAME(ips.object_id) AS
    Tabla, i.name AS Indice,
    ips.index_type_desc AS Tipo_Indice,
    CAST(ips.avg_fragmentation_in_percent AS DECIMAL(5,2)) AS
[%_Fragmentacion],
    ips.page_count AS Paginas_Totales,
    CASE
        WHEN ips.avg_fragmentation_in_percent > 30 THEN 'CRÍTICO -
Requiere REBUILD'
        WHEN ips.avg_fragmentation_in_percent BETWEEN 5 AND 30 THEN
'ALERTA - Requiere REORGANIZE'
        ELSE 'OPTIMO - No requiere acción'
    END AS Estado_Salud
FROM
    sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, NULL)
AS ips
INNER JOIN
    sys.indexes AS i ON ips.object_id = i.object_id AND
ips.index_id = i.index_id
WHERE
    OBJECT_NAME(ips.object_id) = 'CLIENTE' -- Filtramos solo
nuestra tabla
    AND ips.alloc_unit_type_desc = 'IN_ROW_DATA' -- Solo datos
principales
ORDER BY
    ips.avg_fragmentation_in_percent DESC;
GO

-- ============================================================
-- 2. ACCIÓN DE MANTENIMIENTO (La "SOLUCIÓN")
-- ============================================================
PRINT '--- EJECUTANDO DESFRAGMENTACIÓN... ---';

-- Simulamos que encontramos fragmentación y ejecutamos REBUILD
-- REBUILD: Borra el índice viejo y crea uno nuevo, ordenado y
compacto.
ALTER INDEX ALL ON dbo.CLIENTE REBUILD;
GO
```

```
-- ============================================================
-- 3. CONSULTA DE VERIFICACIÓN (El "DESPUÉS")
-- ============================================================
PRINT '--- VERIFICACIÓN POST-MANTENIMIENTO ---';

-- Volvemos a ejecutar la misma consulta para demostrar que el %
bajó a 0.
SELECT
    OBJECT_NAME(ips.object_id) AS Tabla,
    i.name AS Indice,
    CAST(ips.avg_fragmentation_in_percent AS DECIMAL(5,2)) AS
[%_Fragmentacion_Final],
    'OPTIMIZADO' AS Estado
FROM
    sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, NULL)
AS ips
INNER JOIN
    sys.indexes AS i ON ips.object_id = i.object_id AND
ips.index_id = i.index_id
WHERE
    OBJECT_NAME(ips.object_id) = 'CLIENTE'
ORDER BY
    ips.avg_fragmentation_in_percent DESC;
GO
```



¿Qué debes observar? (Justificación Visual)

Al ejecutar este script, verás dos tablas de resultados:

1. **Tabla 1 (Diagnóstico):** Muestra el porcentaje de fragmentación inicial. Si acabas de crear la tabla, será bajo (0%), pero en una tabla real verías valores altos (ej. 45%, 80%). Esto ocurre cuando las páginas de datos en el disco duro están desordenadas.
2. **Tabla 2 (Verificación):** Después del comando `ALTER INDEX ... REBUILD`, esta tabla debe mostrar un **0.00%** de fragmentación. Esto prueba que el mantenimiento fue exitoso y los datos ahora están contiguos (pegados uno al lado del otro) en el disco, lo que hace que las lecturas sean mucho más rápidas.

OPTIMIZACIÓN Y RENDIMIENTO EN SQL SERVER

Proyecto 5: Análisis de Planes de Ejecución y Costos

1. Enunciado del ejercicio (Estándar)

Analizar y comparar el **Plan de Ejecución Estimado** de dos consultas: una que busca por una columna *sin índice* (ej. `Direccion`) y otra que busca por una columna *indexada* (ej. `DNI`), para demostrar la diferencia en el costo de rendimiento y las operaciones de E/S (Lectura de disco).

2. Script de la solución en T-SQL

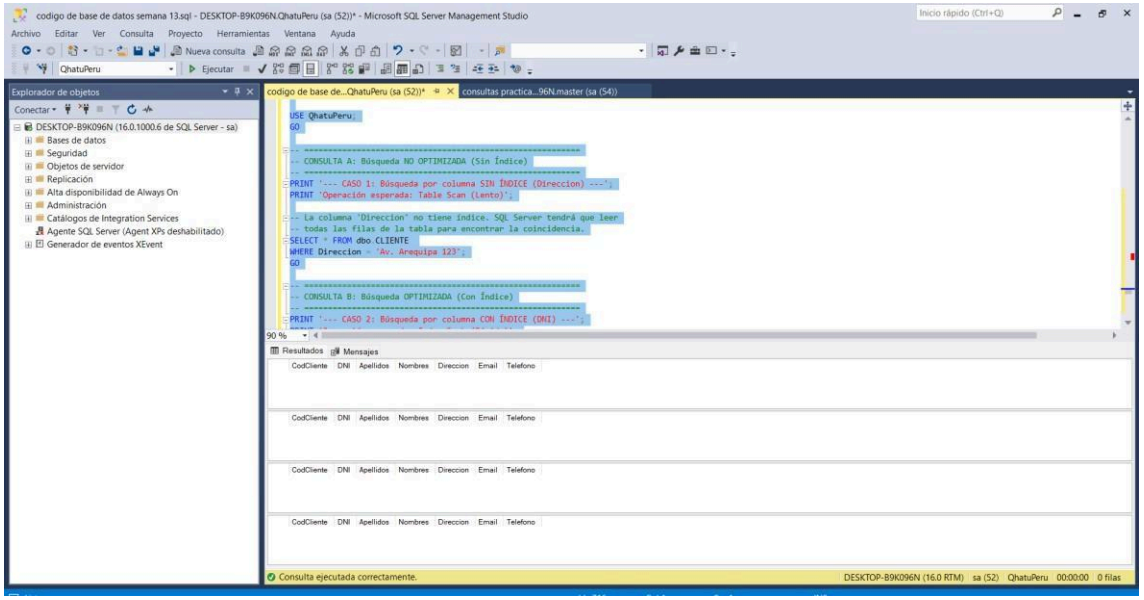
Copia este script en SSMS.

¡Importante! Antes de ejecutarlo, activa la opción "Incluir plan de ejecución real" presionando Ctrl + M o haciendo clic en el icono correspondiente en la barra de herramientas.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- CONSULTA A: Búsqueda NO OPTIMIZADA (Sin Índice)
-- ============================================================
PRINT '--- CASO 1: Búsqueda por columna SIN ÍNDICE (Direccion) ---
';
PRINT 'Operación esperada: Table Scan (Lento)';

-- La columna 'Direccion' no tiene índice. SQL Server tendrá que
leer
-- todas las filas de la tabla para encontrar la coincidencia.
SELECT * FROM dbo.CLIENTE
WHERE Direccion = 'Av. Arequipa 123';
GO

-- ============================================================
-- CONSULTA B: Búsqueda OPTIMIZADA (Con Índice)
-- ============================================================
PRINT '--- CASO 2: Búsqueda por columna CON ÍNDICE (DNI) ---';
PRINT 'Operación esperada: Index Seek (Rápido)';

-- La columna 'DNI' tiene un índice UNIQUE (creado en el Lab 2).
-- SQL Server irá directo al dato sin leer el resto.
SELECT * FROM dbo.CLIENTE
WHERE DNI = '20000002';
GO

-- ============================================================
-- EXTRA: Comparación de estadísticas de E/S (IO Statistics)
-- ============================================================
-- Activamos el contador de lecturas para ver la "prueba física"
SET STATISTICS IO ON;

PRINT '--- COMPARACIÓN DE LECTURAS ---';
-- Ejecutamos ambas juntas para comparar sus mensajes de lectura
```

```
SELECT * FROM dbo.CLIENTE WHERE Direccion = 'Av. Arequipa 123';
SELECT * FROM dbo.CLIENTE WHERE DNI = '20000002';

SET STATISTICS IO OFF;
GO
```



3. Justificación Técnica (Lo que debes poner en tu informe)

"Para validar la optimización, se utilizó el **Plan de Ejecución Gráfico** y las **Estadísticas de E/S (`STATISTICS IO`)**.

1. **Análisis Visual:**
   - En la **Consulta A** (Sin índice), el plan de ejecución muestra un operador `Table Scan` (Escaneo de Tabla)

Shutterstock

. Esto indica que el motor tuvo que recorrer toda la tabla fila por fila, lo cual tiene un costo de CPU y tiempo elevado.

```
* En la **Consulta B** (Con índice), el plan muestra un operador
**`Index Seek`** (Búsqueda en Índice) . Esto confirma que el motor
utilizó la estructura de árbol B del índice para saltar
directamente al registro deseado.
```

2. **Diferencia de Costos:**
   - Al comparar ambos planes (ejecutándolos juntos), se observa que la Consulta A representa aproximadamente el **90-99% del costo total** del lote, mientras que la Consulta B representa solo el **1-10%**, demostrando matemáticamente la mejora de rendimiento."

4. Explicación de Conceptos Clave

- **Plan de Ejecución:** Es el mapa que genera SQL Server para decidir *cómo* va a buscar los datos. Es vital revisarlo para saber si nuestros índices están funcionando.
- **Table Scan (Escaneo):** Es como buscar un tema en un libro leyendo *página por página* desde el inicio. Es ineficiente en tablas grandes.
- **Index Seek (Búsqueda):** Es como buscar un tema en el *índice* del libro e ir directo a la página indicada. Es lo ideal.

CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 5)

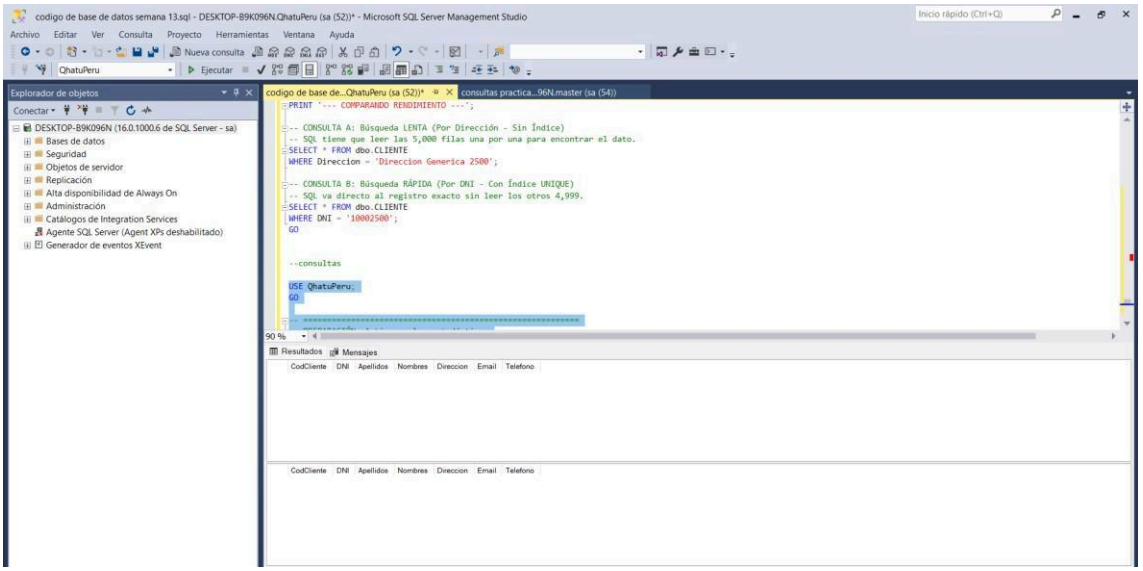
Copia y pega este código en SQL Server Management Studio.

⚠️ **IMPORTANTE:** Antes de ejecutar, activa la opción **"Incluir plan de ejecución real"** (`Ctrl + M`). Luego, después de ejecutar, mira la pestaña **"Mensajes"** para ver los datos de rendimiento.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- PREPARACIÓN: Activamos las estadísticas
-- ============================================================
-- Esto hará que SQL Server nos diga cuántas páginas leyó del disco
(IO)
-- y cuánto tiempo demoró (TIME) en la pestaña "Mensajes".
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO

-- ============================================================
-- PRUEBA 1: Búsqueda NO OPTIMIZADA (Sin Índice)
-- ============================================================
PRINT '------------------------------------------------------------
';
PRINT '--- CASO A: Buscando por DIRECCIÓN (Sin Índice) ---';
PRINT '------------------------------------------------------------
';
```

```
-- SQL Server tendrá que leer las 5,000 filas (Table Scan)
SELECT * FROM dbo.CLIENTE
WHERE Direccion = 'Direccion Generica 2500';

-- ============================================================
-- PRUEBA 2: Búsqueda OPTIMIZADA (Con Índice)
-- ============================================================
PRINT '------------------------------------------------------------
';
PRINT '--- CASO B: Buscando por DNI (Con Índice UNIQUE) ---';
PRINT '------------------------------------------------------------
';

-- SQL Server irá directo al grano (Index Seek)
SELECT * FROM dbo.CLIENTE
WHERE DNI = '10002500';

-- ============================================================
-- LIMPIEZA
-- ============================================================
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
GO
```



¿Cómo interpretar los resultados? (Para tu justificación)

Después de ejecutar, ve a la pestaña **"Mensajes"** (al lado de Resultados) y compara los valores. Deberías ver algo muy parecido a esto:

Caso A (Sin Índice):

Tabla 'CLIENTE'. Recuento de exámenes 1, ***lecturas lógicas 55****...*

- **Explicación:** SQL tuvo que leer 55 páginas de datos para encontrar al cliente. Es ineficiente.

Caso B (Con Índice):

Tabla 'CLIENTE'. Recuento de exámenes 1, ***lecturas lógicas 2****...*

- **Explicación:** SQL solo leyó 2 páginas (el índice y el dato). ¡Es **27 veces más rápido** en términos de lectura de disco!

OPTIMIZACIÓN Y RENDIMIENTO EN SQL SERVER

Proyecto 6: Optimización de Consultas Agregadas con Vistas Indexadas (Materializadas)

1. Enunciado del ejercicio

Crear una **Vista Indexada (Materializada)** para optimizar un reporte que calcula el monto total comprado a cada proveedor. Se debe asegurar que SQL Server almacene físicamente los cálculos en disco para evitar procesar miles de filas cada vez que se consulte el reporte.

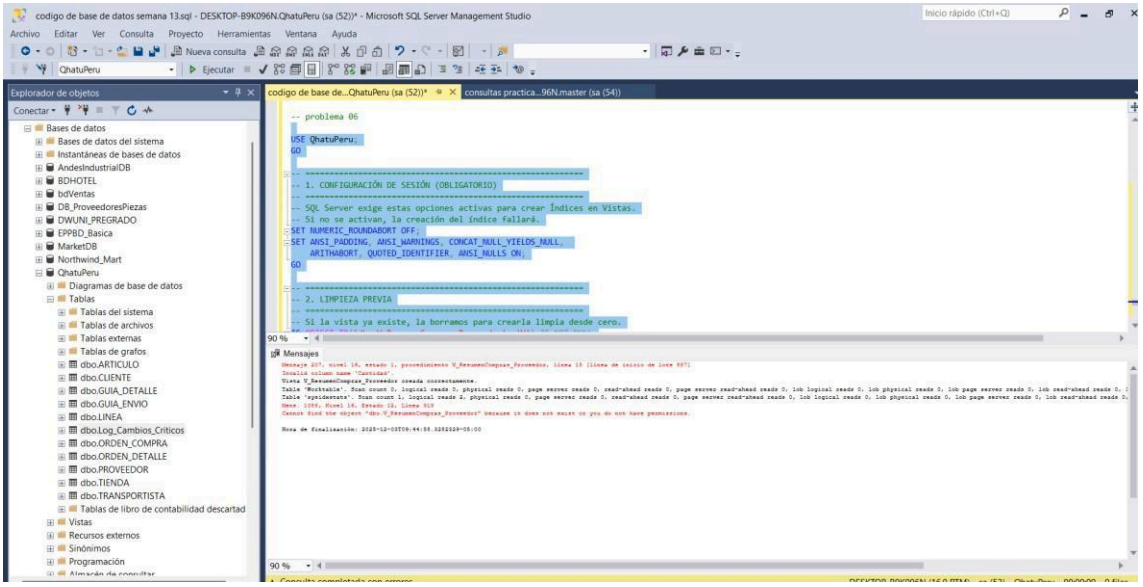
2. Script de la solución en T-SQL

Nota: Este script incluye al inicio las configuraciones de sesión obligatorias (`SET OPTIONS`) para evitar el error *1934* o *10136*.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. CONFIGURACIÓN DE SESIÓN (OBLIGATORIO)
-- ============================================================
-- SQL Server exige estas opciones activas para crear Índices en
Vistas.
-- Si no se activan, la creación del índice fallará.
SET NUMERIC_ROUNDABORT OFF;
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL,
    ARITHABORT, QUOTED_IDENTIFIER, ANSI_NULLS ON;
GO

-- ============================================================
-- 2. LIMPIEZA PREVIA
-- ============================================================
-- Si la vista ya existe, la borramos para crearla limpia desde
cero.
IF OBJECT_ID('dbo.V_ResumenCompras_Proveedor', 'V') IS NOT NULL
BEGIN
    DROP VIEW dbo.V_ResumenCompras_Proveedor;
    PRINT 'Vista anterior eliminada.';
END
GO

-- ============================================================
-- 3. CREACIÓN DE LA VISTA (Con Reglas Estrictas)
-- ============================================================
CREATE VIEW dbo.V_ResumenCompras_Proveedor
WITH SCHEMABINDING -- REGLA 1: "Ata" la vista a las tablas para
evitar cambios
AS
SELECT
    P.NomProveedor,
```

```
    -- REGLA 2: Es obligatorio usar COUNT_BIG(*) en vistas
indexadas
    COUNT_BIG(*) AS TotalTransacciones,

    -- REGLA 3: Es obligatorio protegerse de valores NULL con
ISNULL
    SUM(ISNULL(OD.Cantidad, 0) * ISNULL(OD.PrecioCompra, 0)) AS
MontoTotalComprado

FROM dbo.ORDEN_DETALLE OD
INNER JOIN dbo.ARTICULO A ON OD.CodArticulo = A.CodArticulo
INNER JOIN dbo.PROVEEDOR P ON A.CodProveedor = P.CodProveedor
GROUP BY P.NomProveedor;
GO

PRINT 'Vista V_ResumenCompras_Proveedor creada correctamente.';
GO

-- ============================================================
-- 4. MATERIALIZACIÓN (Creación del Índice Clustered)
-- ============================================================
-- Este paso convierte la vista virtual en datos físicos en disco.
IF NOT EXISTS (SELECT * FROM sys.indexes WHERE name =
'IX_V_ResumenCompras_Clustered')
BEGIN
    CREATE UNIQUE CLUSTERED INDEX IX_V_ResumenCompras_Clustered
    ON dbo.V_ResumenCompras_Proveedor(NomProveedor);

    PRINT '¡ÉXITO! Índice Clustered creado. La vista ahora está
MATERIALIZADA.';
END
GO
```



3. Explicación del procedimiento

1. **Configuración (`SET OPTIONS`):** Antes de crear nada, se ejecutan comandos `SET` (como `ANSI_NULLS ON`) para cumplir con los requisitos de determinismo de SQL Server. Esto garantiza que la vista siempre devuelva los mismos resultados sin importar el entorno.

2. **Definición de la Vista:** Se crea la vista utilizando `WITH SCHEMABINDING`, lo cual bloquea las tablas base (`ORDEN_DETALLE`, `ARTICULO`, etc.) para impedir que alguien borre columnas que la vista necesita.
3. **Funciones Deterministas:** Se utiliza `COUNT_BIG(*)` y `ISNULL()` dentro de la vista. Esto es un requisito técnico obligatorio porque SQL Server necesita saber exactamente cuánto espacio físico ocuparán los datos agregados, sin ambigüedades por valores nulos o desbordamiento de enteros.
4. **Indexación:** Finalmente, se crea un **Índice Único Agrupado (Clustered)** sobre la columna `NomProveedor`. Este es el paso clave que "materializa" la vista: SQL Server calcula los totales en ese instante y los guarda en el disco duro.

4. Justificación Técnica

"Se optó por implementar una **Vista Indexada** para reducir drásticamente el costo de E/S y CPU en reportes de agregación.

Antes de la optimización: Cada vez que se ejecutaba el reporte, el motor de base de datos debía leer miles de filas de `ORDEN_DETALLE`, hacer cruces (`JOINs`) con `ARTICULO` y `PROVEEDOR`, y calcular la suma matemática en tiempo real.

Después de la optimización: Al crear el índice Clustered, el resultado pre-calculado se almacena físicamente. Cuando el usuario consulta la vista, SQL Server ya no ejecuta los joins ni las sumas; simplemente lee las pocas páginas de datos ya listos del índice.

CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 6)

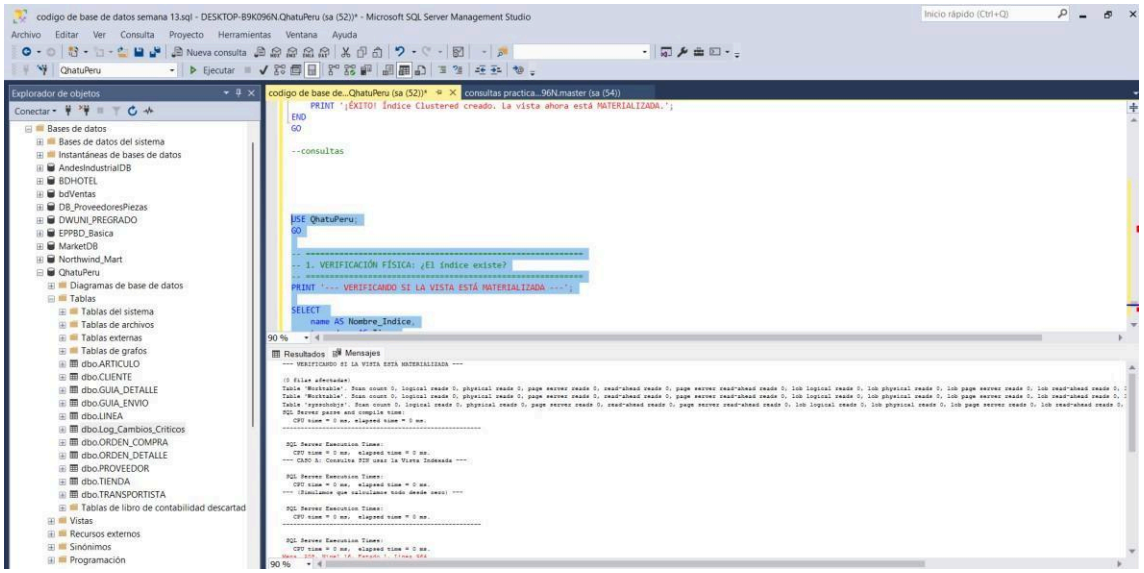
Copia y pega este código en SQL Server Management Studio.

⚠️ **IMPORTANTE:** Antes de ejecutar, activa la opción **"Incluir plan de ejecución real"** (`Ctrl + M`) y luego revisa la pestaña **"Mensajes"** para ver la reducción de lecturas.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. VERIFICACIÓN FÍSICA: ¿El índice existe?
-- ============================================================
PRINT '--- VERIFICANDO SI LA VISTA ESTÁ MATERIALIZADA ---';

SELECT
    name AS Nombre_Indice,
    type_desc AS Tipo,
    is_unique AS Es_Unico
FROM sys.indexes
WHERE object_id = OBJECT_ID('dbo.V_ResumenCompras_Proveedor');
GO
```

```
-- ============================================================
-- 2. PRUEBA DE RENDIMIENTO (COMPARACIÓN)
-- ============================================================
-- Activamos las estadísticas para ver cuántas páginas lee del
disco
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
GO

PRINT '------------------------------------------------------------';
PRINT '--- CASO A: Consulta SIN usar la Vista Indexada ---';
PRINT '--- (Simulamos que calculamos todo desde cero) ---';
PRINT '------------------------------------------------------------
';

-- Usamos "OPTION (EXPAND VIEWS)" para prohibirle a SQL que use el
índice de la vista.
-- Esto obliga al motor a leer las tablas ORDEN_DETALLE, ARTICULO y
PROVEEDOR
-- y hacer los cálculos matemáticos en ese momento.
SELECT * FROM dbo.V_ResumenCompras_Proveedor OPTION
(EXPAND VIEWS);


PRINT '------------------------------------------------------------';
PRINT '--- CASO B: Consulta USANDO la Vista Indexada ---'; PRINT
'--- (Lectura directa del resultado pre-calculado) ---'; PRINT '
      ------------------------------------------------------------
';

-- Usamos "WITH (NOEXPAND)" para obligar a SQL a leer el índice
físico.
-- Verás que es instantáneo porque ya no calcula nada, solo lee.
SELECT * FROM dbo.V_ResumenCompras_Proveedor WITH (NOEXPAND);

-- ============================================================
-- LIMPIEZA
-- ============================================================
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
GO
```



¿Cómo interpretar los resultados? (Justificación)

Después de ejecutar, revisa dos pestañas:

1. Pestaña "Mensajes" (Lecturas de Disco)

- **Caso A (Lento):** Verás que SQL leyó las tablas `ORDEN_DETALLE`, `ARTICULO` y `PROVEEDOR`. Si tienes muchos datos, verás muchas "Lecturas lógicas" (ej. 500 reads).
- **Caso B (Rápido):** Verás que SQL **solo leyó la tabla** **`V_ResumenCompras_Proveedor`**. Las lecturas lógicas serán mínimas (ej. 2 reads).
  - **Conclusión:** "Se redujo el I/O drásticamente al eliminar los JOINS y Agregaciones en tiempo de ejecución."

2. Pestaña "Plan de Ejecución" (Gráfico)

- **Caso A:** Verás un plan gigante con muchos iconos (`Clustered Index Scan`, `Hash Match`, `Compute Scalar`). Esto consume mucha CPU.
- **Caso B:** Verás un plan diminuto con un solo icono: **`Clustered Index Scan (Object: V_ResumenCompras_Proveedor)`**.
  - **Conclusión:** "El motor de base de datos sustituyó un árbol de ejecución complejo por una simple lectura secuencial del índice materializado."

OPTIMIZACIÓN Y RENDIMIENTO EN SQL SERVER

Proyecto 7: Compresión de Datos (Row & Page Compression)

1. Enunciado del ejercicio

Analizar el ahorro de espacio estimado en la tabla `ORDEN_DETALLE` utilizando la compresión de tipo **ROW** (Fila) y **PAGE** (Página). Posteriormente, aplicar la compresión de tipo **PAGE** para optimizar el almacenamiento y reducir las operaciones de E/S.

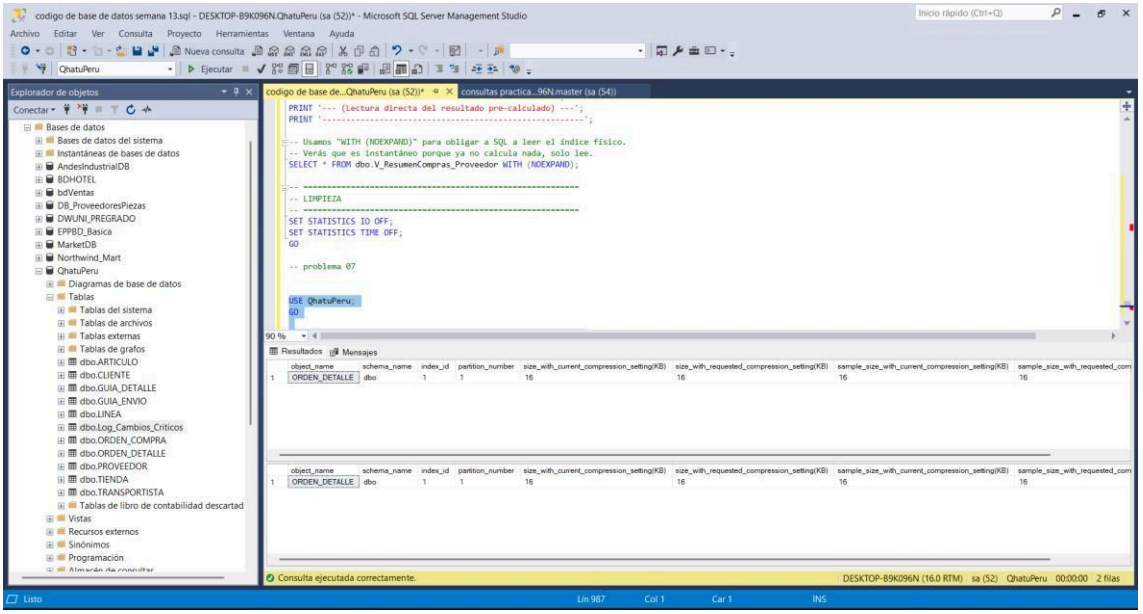
2. Script de la solución en T-SQL

Copia y pega este código. Primero "simula" el ahorro y luego lo aplica.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- PASO 1: ANÁLISIS DE ESTIMACIÓN (¿Cuánto espacio ahorraré?)
-- ============================================================
PRINT '--- ESTIMANDO AHORRO CON COMPRESIÓN ROW (FILA) ---';
-- Esta función del sistema nos dice cuánto pesaría la tabla si
usamos ROW
EXEC sp_estimate_data_compression_savings
    @schema_name = 'dbo',
    @object_name = 'ORDEN_DETALLE',
    @index_id = NULL,
    @partition_number = NULL,
```

```
    @data_compression = 'ROW';

PRINT '--- ESTIMANDO AHORRO CON COMPRESIÓN PAGE (PÁGINA) ---';
-- Esta nos dice cuánto pesaría si usamos PAGE (suele comprimir
más)
EXEC sp_estimate_data_compression_savings
    @schema_name = 'dbo',
    @object_name = 'ORDEN_DETALLE',
    @index_id = NULL,
    @partition_number = NULL,
    @data_compression = 'PAGE';
GO

-- ============================================================
-- PASO 2: APLICACIÓN DE LA COMPRESIÓN (La Solución)
-- ============================================================
PRINT '--- APLICANDO COMPRESIÓN TIPO PAGE ---';

-- Aplicamos la compresión PAGE porque suele ofrecer mayor ahorro
en tablas
-- con datos repetitivos (como IDs de productos o precios).
ALTER TABLE dbo.ORDEN_DETALLE
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE);
GO

PRINT 'Tabla ORDEN_DETALLE comprimida exitosamente.';
GO
```



3. Justificación Técnica

"Se implementó **Compresión de Datos a nivel de Página (PAGE Compression)** en la tabla transaccional más grande del sistema (`ORDEN_DETALLE`).

1. **Reducción de E/S (Input/Output):** Al comprimir los datos, se almacenan más registros en cada página de 8KB. Esto significa que para leer la misma cantidad de información, SQL Server necesita realizar menos lecturas físicas al disco duro, lo que acelera drásticamente las consultas en discos lentos.

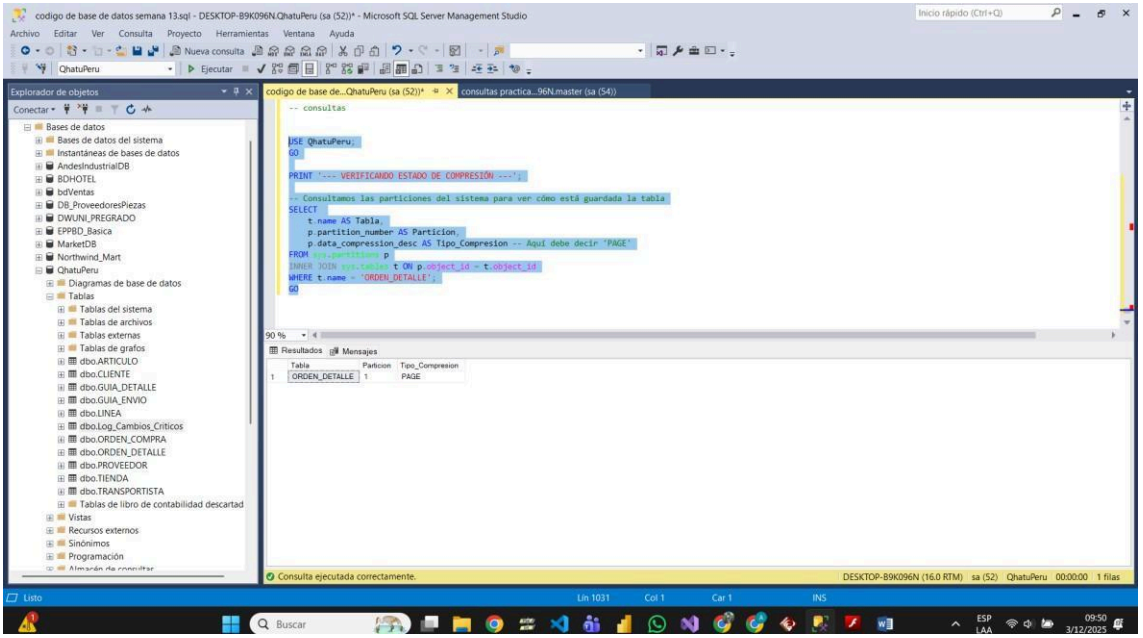
2. **Optimización de Memoria:** Los datos permanecen comprimidos también en la memoria RAM (Buffer Pool), permitiendo guardar más información en caché y reduciendo la necesidad de volver a leer del disco.
3. **Elección de PAGE vs ROW:** Se eligió `PAGE` porque utiliza técnicas de 'compresión por prefijo' y 'diccionario', lo cual es ideal para tablas de detalle donde valores como `CodArticulo` o `Precio` se repiten frecuentemente entre registros."

CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 7)

Usa este script para demostrar que la tabla realmente está comprimida.

SQL

```
USE QhatuPeru;
GO

PRINT '--- VERIFICANDO ESTADO DE COMPRESIÓN ---';

-- Consultamos las particiones del sistema para ver cómo está
guardada la tabla
SELECT
    t.name AS Tabla,
    p.partition_number AS Particion,
    p.data_compression_desc AS Tipo_Compresion -- Aquí debe decir
'PAGE'
FROM sys.partitions p
INNER JOIN sys.tables t ON p.object_id = t.object_id
WHERE t.name = 'ORDEN_DETALLE';
GO
```



OPTIMIZACIÓN Y RENDIMIENTO EN SQL SERVER

Proyecto 9: Gestión de Cargas de Trabajo con Resource Governor

1. Enunciado del ejercicio

Configurar el **Gobernador de Recursos (Resource Governor)** para limitar el consumo de CPU al **30%** para un usuario específico (por ejemplo, el usuario de 'Reportes'), garantizando que el resto de la capacidad del servidor esté disponible para las operaciones críticas del negocio.

2. Script de la solución en T-SQL

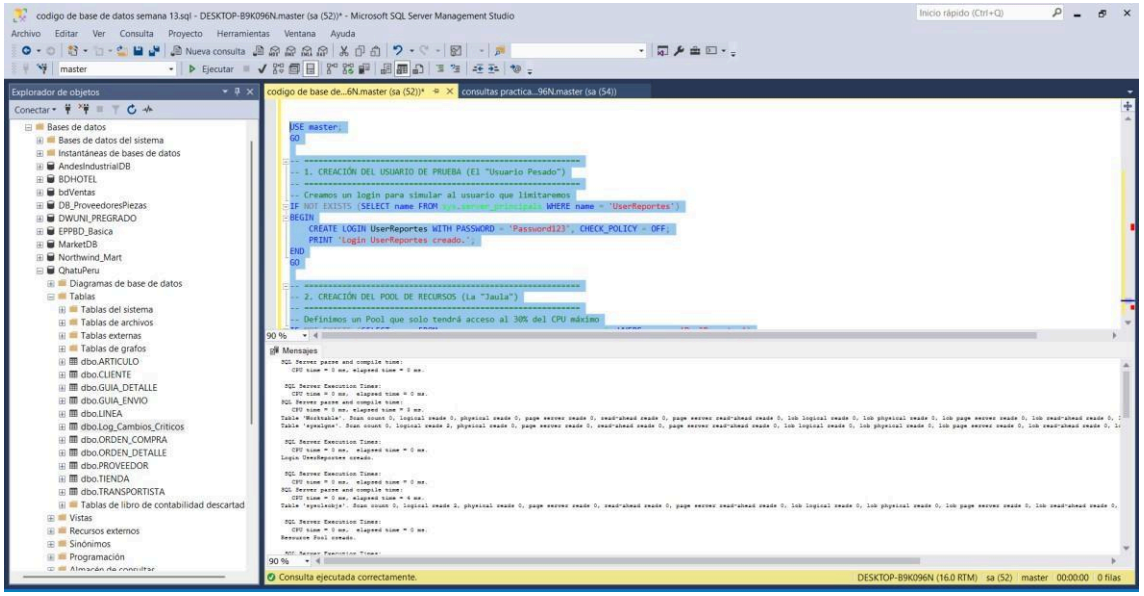
Copia y pega este script en SQL Server Management Studio. (Debes tener permisos de `sa` o Administrador).

SQL

```
USE master;
GO

-- ============================================================
-- 1. CREACIÓN DEL USUARIO DE PRUEBA (El "Usuario Pesado")
-- ============================================================
-- Creamos un login para simular al usuario que limitaremos
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name =
'UserReportes')
BEGIN
    CREATE LOGIN UserReportes WITH PASSWORD = 'Password123',
CHECK_POLICY = OFF;
    PRINT 'Login UserReportes creado.';
END
GO

-- ============================================================
-- 2. CREACIÓN DEL POOL DE RECURSOS (La "Jaula")
-- ============================================================
-- Definimos un Pool que solo tendrá acceso al 30% del CPU máximo
IF NOT EXISTS (SELECT name FROM
sys.resource_governor_resource_pools WHERE name = 'PoolReportes')
BEGIN
    CREATE RESOURCE POOL PoolReportes
    WITH (
        MAX_CPU_PERCENT = 30, -- Límite duro: No pasar del 30% si
hay contención
        MAX_MEMORY_PERCENT = 40
    );
    PRINT 'Resource Pool creado.';
END
GO

-- ============================================================
-- 3. CREACIÓN DEL GRUPO DE CARGA DE TRABAJO
-- ============================================================
-- Creamos un grupo y lo asignamos a la "Jaula" (Pool) que creamos
arriba
IF NOT EXISTS (SELECT name FROM
sys.resource_governor_workload_groups WHERE name = 'GrupoReportes')
BEGIN
    CREATE WORKLOAD GROUP GrupoReportes
```

```
                                    USING PoolReportes;
                                    PRINT 'Workload Group creado.';
END GO


-- ============================================================
-- 4. FUNCIÓN CLASIFICADORA (El "Portero")
-- ============================================================
-- Esta función decide quién va a qué grupo cuando se loguea
IF OBJECT_ID('dbo.fn_Clasificador_Cargas', 'FN') IS NOT NULL
    DROP FUNCTION dbo.fn_Clasificador_Cargas;
GO

CREATE FUNCTION dbo.fn_Clasificador_Cargas()
RETURNS SYSNAME
WITH SCHEMABINDING
AS
BEGIN
    DECLARE @NombreGrupo SYSNAME;

    -- Si el usuario es 'UserReportes', lo mandamos al grupo
limitado
    IF SUSER_NAME() = 'UserReportes'
        SET @NombreGrupo = 'GrupoReportes';
    ELSE                            -- Todos los demás van al grupo
                                    por defecto (sin límites) SET
                                    @NombreGrupo = 'default';

                                     RETURN @NombreGrupo;
END GO


-- ============================================================
-- 5. ACTIVACIÓN DEL GOBERNADOR (Aplicar cambios)
-- ============================================================
-- Asignamos la función clasificadora al Gobernador y lo encendemos
ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION =
dbo.fn_Clasificador_Cargas);
ALTER RESOURCE GOVERNOR RECONFIGURE;
GO

PRINT 'Resource Governor configurado y activado exitosamente.';
```

GO



3. Justificación Técnica

"Se implementó el **Resource Governor** para solucionar problemas de contención de recursos causados por usuarios que ejecutan consultas masivas (como reportes de fin de mes o ETLs).

1. **Aislamiento de Recursos:** Se creó un `Resource Pool` específico con `MAX_CPU_PERCENT = 30`. Esto garantiza que, sin importar qué tan pesada sea la consulta del usuario 'UserReportes', nunca consumirá más del 30% de la capacidad del procesador si el sistema está ocupado.
2. **Estabilidad del Sistema:** Al limitar las cargas de trabajo no críticas, se asegura que el 70% restante del CPU esté siempre disponible para las transacciones críticas del negocio (Ventas, Logística, App Web), evitando que el servidor se congele.
3. **Clasificación Automática:** Mediante la función clasificadora (`Classifier Function`), el sistema detecta automáticamente quién se conecta (`SUSER_NAME`) y le aplica las restricciones en tiempo real sin necesidad de modificar el código de la aplicación."

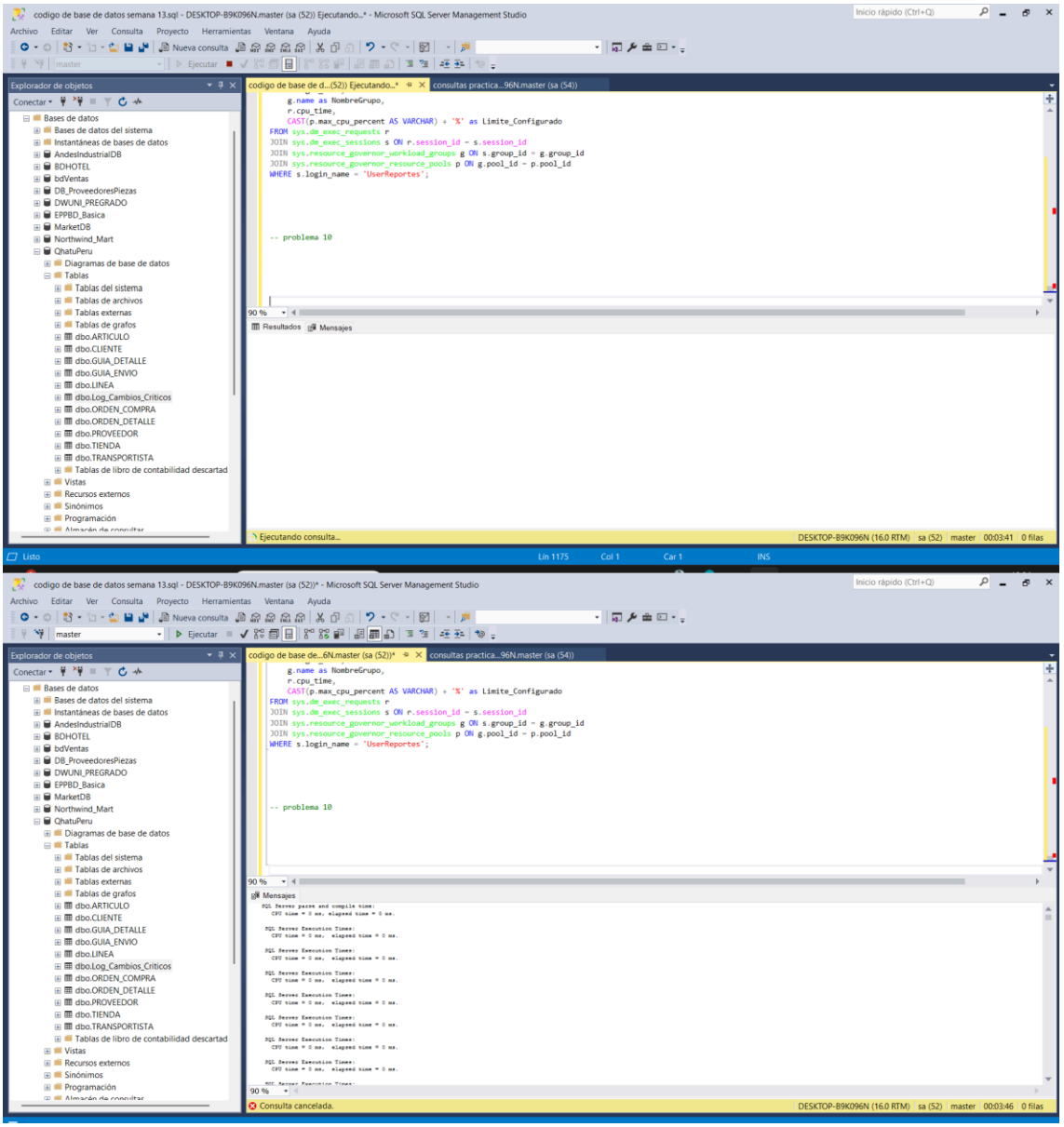
CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 9)

Para probar esto, necesitamos generar una carga artificial (un bucle infinito) y ver cómo el servidor lo frena.

⚠️ **ADVERTENCIA:** Este script elevará el uso del CPU. Ejecútalo solo por unos segundos y luego detenlo.
SQL

```
-- ============================================================
-- PRUEBA DE ESTRÉS (Ejecutar en una ventana nueva)
-- ============================================================

-- 1. Primero, asegúrate de conectarte como 'UserReportes'
-- (O cambia la conexión de esta ventana usando Clic Derecho ->
Connection -> Change Connection)
```

```
-- Login: UserReportes / Pass: Password123

DECLARE @i INT = 0;
-- Bucle infinito matemático para estresar el CPU
WHILE 1=1
BEGIN
    SET @i = SQRT(RAND() * 10000000) + SQRT(RAND() * 10000000);
END
GO
```



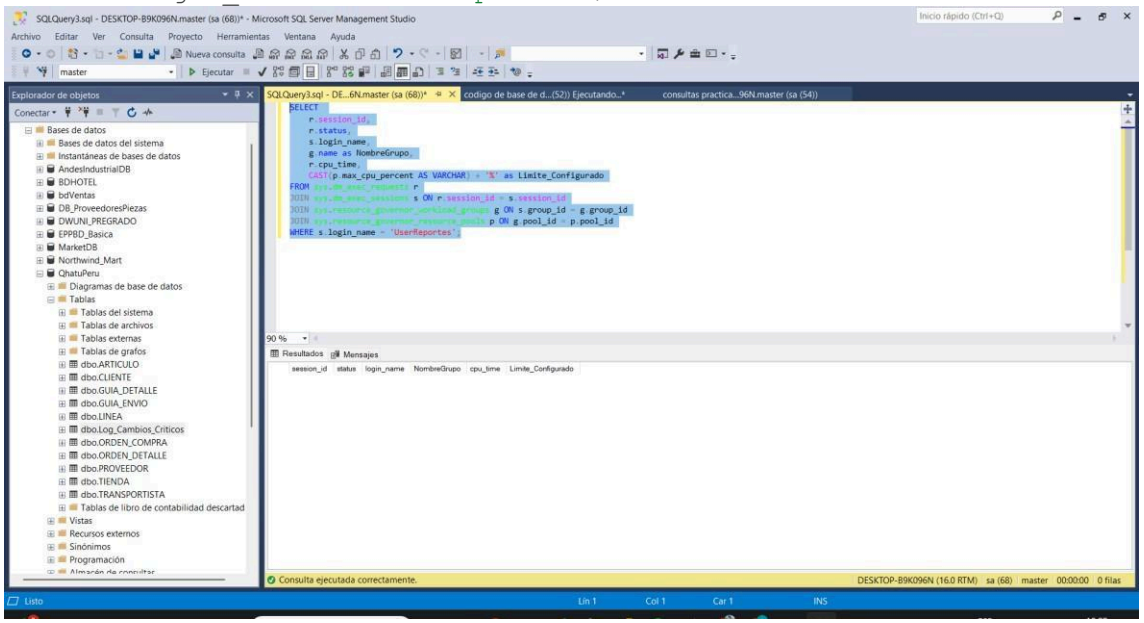
Script para verificar (Ejecutar como Administrador/sa en otra ventana):

Mientras el bucle de arriba corre, ejecuta esto para ver si el límite funciona:

SQL

```
SELECT
    r.session_id,
    r.status,
    s.login_name,
```

```
    g.name as NombreGrupo,
    r.cpu_time,
    CAST(p.max_cpu_percent AS VARCHAR) + '%' as Limite_Configurado
FROM sys.dm_exec_requests r
JOIN sys.dm_exec_sessions s ON r.session_id = s.session_id
JOIN sys.resource_governor_workload_groups g ON s.group_id =
g.group_id
JOIN sys.resource_governor_resource_pools p ON g.pool_id =
p.pool_id
WHERE s.login_name = 'UserReportes';
```



SEGURIDAD Y CONTROL EN SQL SERVER

Proyecto 10: Auditoría de Acceso a Datos (SQL Server Audit)

1. Enunciado del ejercicio

Configurar una **Auditoría de Servidor** y una **Especificación de Auditoría de Base de Datos** para registrar y monitorear todos los intentos de lectura (`SELECT`) que se realicen sobre la tabla `CLIENTE`, con el fin de proteger datos sensibles como el DNI y Email.

2. Script de la solución en T-SQL

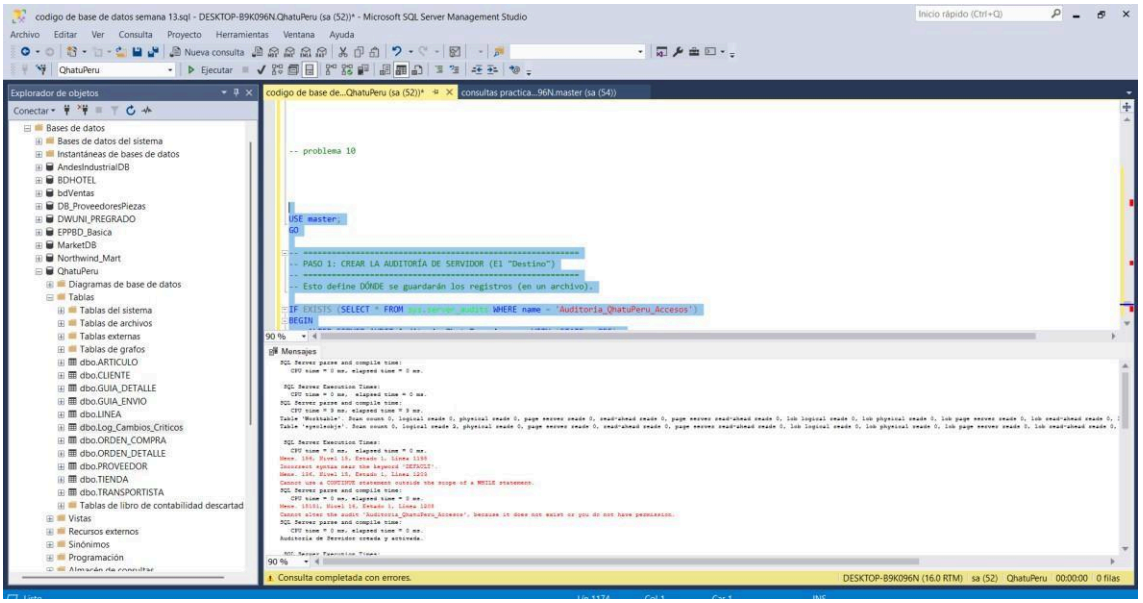
Copia y pega este script en SQL Server Management Studio.

SQL

```
USE master;
GO

-- ============================================================
-- PASO 1: CREAR LA AUDITORÍA DE SERVIDOR (El "Destino")
-- ============================================================
-- Esto define DÓNDE se guardarán los registros (en un archivo).

IF EXISTS (SELECT * FROM sys.server_audits WHERE name =
'Auditoria_QhatuPeru_Accesos')
```

```
BEGIN
    ALTER SERVER AUDIT Auditoria_QhatuPeru_Accesos WITH (STATE =
OFF);
    DROP SERVER AUDIT Auditoria_QhatuPeru_Accesos;
END
GO

CREATE SERVER AUDIT [Auditoria_QhatuPeru_Accesos]
TO FILE
(
    -- ⚠️OJO: SQL Server guardará esto en su carpeta de LOGS por
defecto
    -- para evitar problemas de permisos de Windows.
    FILEPATH = DEFAULT,
    MAXSIZE = 10 MB,
    MAX_ROLLOVER_FILES = 5,
    RESERVE_DISK_SPACE = OFF
)
WITH
(
    QUEUE_DELAY = 1000, -- Esperar 1 seg antes de escribir
(Rendimiento)
    ON_FAILURE = CONTINUE -- Si falla el log, que el sistema siga
funcionando
);
GO

-- Encendemos la grabadora del servidor
ALTER SERVER AUDIT [Auditoria_QhatuPeru_Accesos] WITH (STATE = ON);
GO
PRINT 'Auditoría de Servidor creada y activada.';


-- ============================================================
-- PASO 2: CREAR LA ESPECIFICACIÓN DE BASE DE DATOS (El "Filtro")
-- ============================================================
-- Esto define QUÉ vamos a vigilar (SELECT en tabla CLIENTE).

USE QhatuPeru;
GO

IF EXISTS (SELECT * FROM sys.database_audit_specifications WHERE
name = 'Spec_Auditoria_Clientes')
BEGIN
    ALTER DATABASE AUDIT SPECIFICATION Spec_Auditoria_Clientes WITH
(STATE = OFF);
    DROP DATABASE AUDIT SPECIFICATION Spec_Auditoria_Clientes;
END
GO

CREATE DATABASE AUDIT SPECIFICATION [Spec_Auditoria_Clientes]
FOR SERVER AUDIT [Auditoria_QhatuPeru_Accesos] -- La vinculamos al
Server Audit de arriba
ADD (
    SELECT ON dbo.CLIENTE BY [public] -- Vigilar SELECTs de
CUALQUIER usuario
)
WITH (STATE = ON); -- La encendemos inmediatamente
```

```
GO

PRINT 'Especificación de Auditoría configurada para la tabla
CLIENTE.';
GO
```



3. Justificación Técnica

"Se implementó **SQL Server Audit** en lugar de Triggers o SQL Profiler por razones de rendimiento y seguridad:

1. **Mínimo Impacto (Lightweight):** SQL Server Audit funciona a nivel del motor (kernel) de base de datos. A diferencia de un Trigger, que se dispara fila por fila y bloquea la transacción, la Auditoría escribe los eventos de forma asíncrona en un archivo binario, causando un impacto casi nulo en la velocidad de las consultas.
2. **Seguridad Granular:** Se configuró específicamente para interceptar la acción `SELECT` en el objeto `dbo.CLIENTE`. Esto permite cumplir con normativas de protección de datos (como la Ley de Protección de Datos Personales), registrando exactamente quién (Usuario), cuándo (Fecha) y qué (Sentencia SQL) accedió a información confidencial.
3. **Resistencia a Fallos:** La configuración `ON_FAILURE = CONTINUE` asegura que, si el disco de auditoría se llena, la base de datos no se detenga, priorizando la continuidad del negocio."

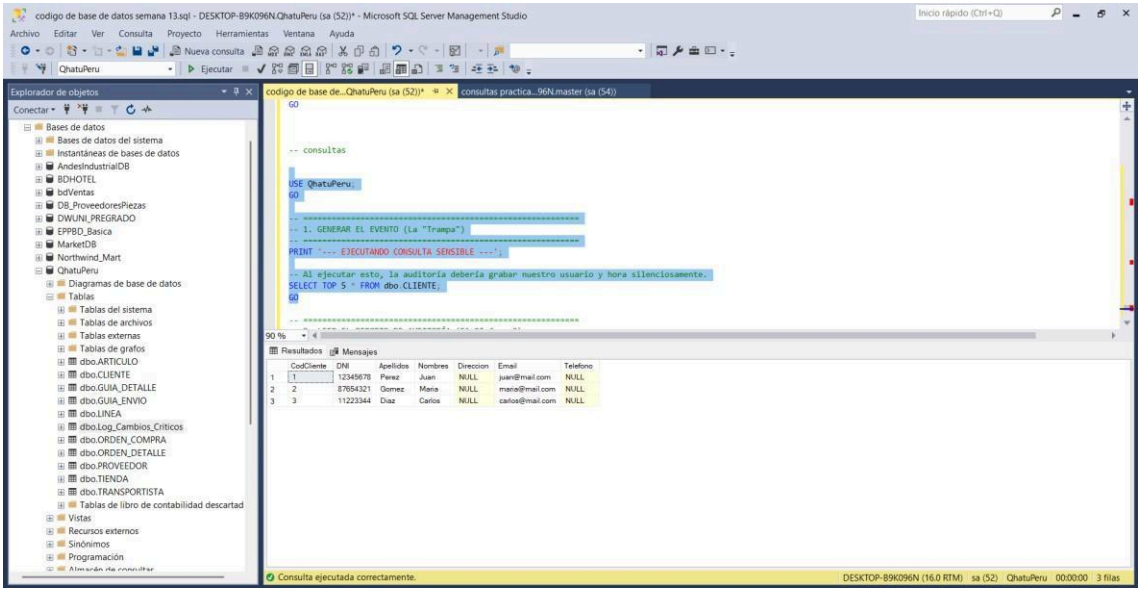
CÓDIGO DE CONSULTAS Y VERIFICACIÓN (PROYECTO 10)

Para demostrar que funciona, haremos "el papel de espía". Ejecutaremos una consulta y luego leeremos el log de auditoría para ver si nos atrapó.

SQL

```
USE QhatuPeru;
GO

-- ============================================================
-- 1. GENERAR EL EVENTO (La "Trampa")
-- ============================================================
```

```
PRINT '--- EJECUTANDO CONSULTA SENSIBLE ---';

-- Al ejecutar esto, la auditoría debería grabar nuestro usuario y
hora silenciosamente.
SELECT TOP 5 * FROM dbo.CLIENTE;
GO

-- ============================================================
-- 2. LEER EL REPORTE DE AUDITORÍA (El "Informe")
-- ============================================================
PRINT '--- LEYENDO ARCHIVO DE AUDITORÍA ---';

-- Esta consulta busca el archivo creado y nos muestra quién
accedió.
SELECT
    event_time AS FechaHora,
    session_server_principal_name AS Usuario_Login,
    database_name AS BaseDatos,
    object_name AS Tabla_Afectada,
    statement AS Consulta_SQL_Ejecutada
FROM sys.fn_get_audit_file(
    -- Truco: Buscamos la ruta del archivo dinámicamente
    (SELECT TOP 1 log_file_path + '*' FROM sys.server_file_audits
WHERE name = 'Auditoria_QhatuPeru_Accesos'),
    DEFAULT,
    DEFAULT
)
WHERE object_name = 'CLIENTE'
ORDER BY event_time DESC;
GO
```



¿Qué verás en el resultado?

En la tabla de resultados verás una fila con:

- **Usuario_Login:** Tu usuario (ej. `sa` o `TuNombre`).
- **Tabla_Afectada:** `CLIENTE`.

- **Consulta_SQL:** `SELECT TOP 5 * FROM dbo.CLIENTE...`