



UNIVERSIDAD PERUANA LOS ANDES



**Asignatura:  
BASE DE DATOS II**

**Tema:  
MONITOREO Y RENDIMIENTO**

**Docente:  
FERNANDEZ BEJARANO RAUL**

**Estudiante:  
BREÑA QUISPE MEGAM**

HUANCAYO-2025

# 1. Análisis de rendimiento con SQL Profiler y Extended Events

## SQL Profiler — Concepto

- SQL Profiler es una herramienta gráfica de SQL Server que permite capturar y analizar en tiempo real las acciones que ocurren dentro del motor, como:
  - Consultas ejecutadas
  - Bloqueos
  - Errores
  - Inicios de sesión
  - Uso de CPU
  - Lecturas y escrituras
- Es muy útil para diagnosticar problemas de rendimiento, especialmente cuando una consulta está lenta o cuando hay mucha carga en el servidor.

Sin embargo, SQL Profiler está en proceso de ser reemplazado porque consume bastantes recursos cuando capture grandes volúmenes de datos.

## Extended Events — Concepto

Extended Events (XE) es el sistema moderno de monitoreo de SQL Server. Está pensado para reemplazar a SQL Profiler.

Ofrece:

- Bajo consumo de recursos
- Más eventos disponibles
- Mejor filtrado
- Posibilidad de guardar el resultado en archivos .xel
- Integración con SSMS

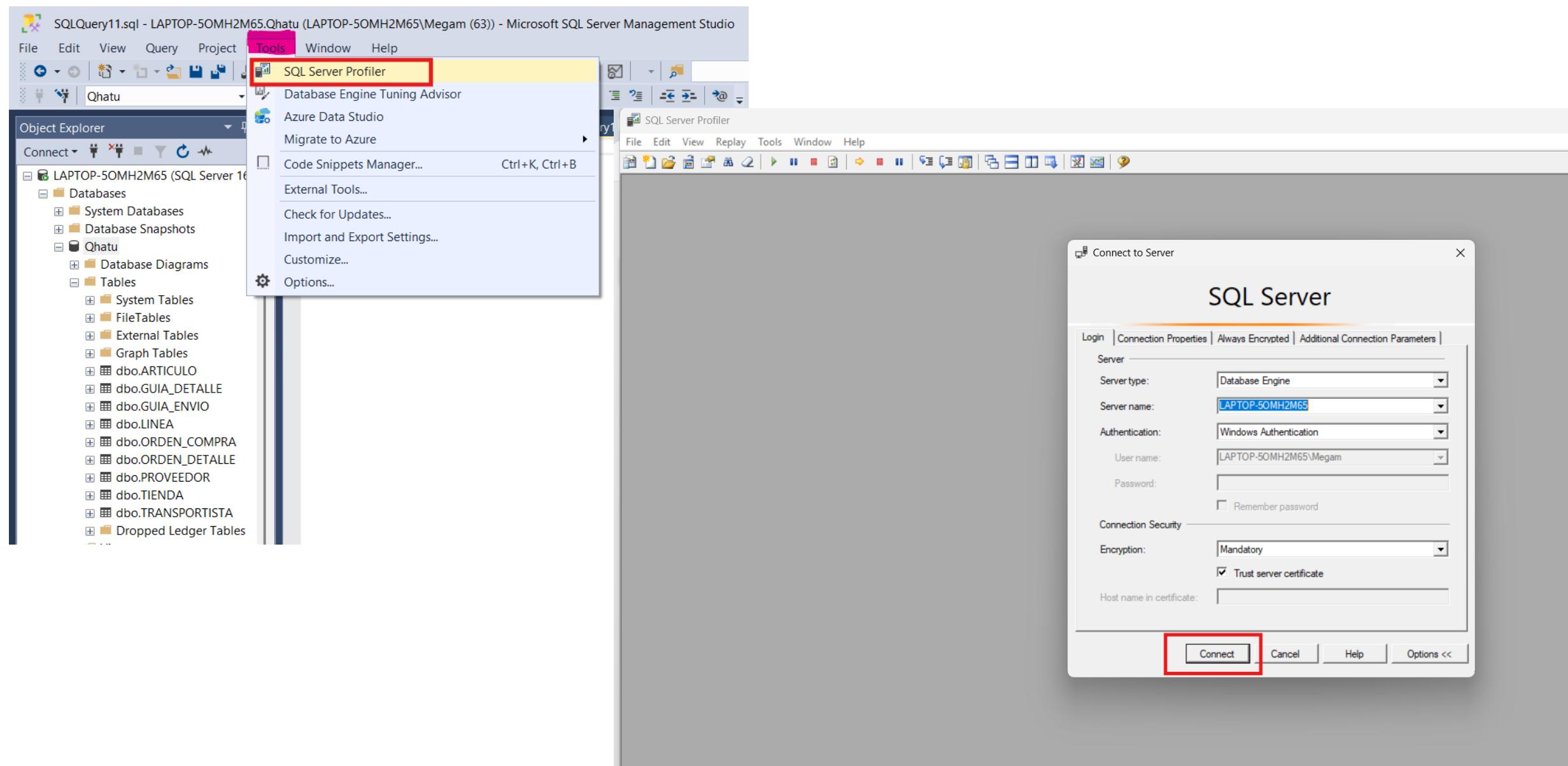
Es ideal para diagnosticar:

- Consultas lentas
- Bloqueos (deadlocks)
- Alto consumo de CPU
- Problemas intermitentes

# Diferencias principales:

SQL Profiler	Extended Events
Herramienta antigua	Nuevo estándar
Alto consumo de recursos	Ligero y eficiente
Menos eventos disponibles	Muchos más eventos
Interfaz básica	Interfaz moderna en SSMS
No recomendado en producción	Recomendado para producción

# Pasos para usar el SQL Profiler



# Código de ejemplo para probar (Extended Events)

- Crear un Extended Event para detectar consultas lentas

```
CREATE EVENT SESSION SlowQueries
```

```
ON SERVER
```

```
ADD EVENT sqlserver.rpc_completed(
```

```
    ACTION(sqlserver.sql_text)
```

```
    WHERE duration > 1000
```

```
),
```

```
ADD EVENT sqlserver.sql_batch_completed(
```

```
    ACTION(sqlserver.sql_text)
```

```
    WHERE duration > 1000
```

```
)
```

```
ADD TARGET package0.event_file (
```

```
    SET filename = 'C:\Program Files\Microsoft SQL  
Server\MSSQL16.MSSQLSERVER\MSSQL\Log\slow_queries.xel'
```

```
)
```

```
WITH (STARTUP_STATE = ON);
```

```
GO
```

```
ALTER EVENT SESSION SlowQueries ON SERVER STATE = START;
```

```
GO
```

```
-- PASO 2
CREATE EVENT SESSION SlowQueries
ON SERVER
ADD EVENT sqlserver.rpc_completed(
    ACTION(sqlserver.sql_text)
    WHERE duration > 1000
),
ADD EVENT sqlserver.sql_batch_completed(
    ACTION(sqlserver.sql_text)
    WHERE duration > 1000
)
ADD TARGET package0.event_file (
    SET filename = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Log\slow_queries.xel'
)
WITH (STARTUP_STATE = ON);
GO

ALTER EVENT SESSION SlowQueries ON SERVER STATE = START;
GO

WAITFOR DELAY '00:00:02';
SELECT 'Prueba consulta lenta';
```

Detener sesión:

**ALTER EVENT SESSION SlowQueries ON SERVER STATE = STOP;**

## 2. Estadísticas e índices (creación, fragmentación, mantenimiento).

### ¿Qué son las estadísticas?

Las estadísticas son objetos internos que usa el Motor de SQL Server para estimar cuántas filas tiene una tabla, cómo están distribuidos los valores y qué tan selectivas son las búsquedas.

Son fundamentales para que SQL Server elija el mejor plan de ejecución.

Las estadísticas contienen:

- Número de filas
- Distribución de valores (histograma)
- Densidad
- Promedio de repetición
- Columna o columnas involucradas

SQL Server usa estadísticas para decidir:

- Qué índice usar
- Cuántas lecturas hacer
- Si hacer un JOIN hash, merge o nested loop
- Si escanear una tabla o usar un índice

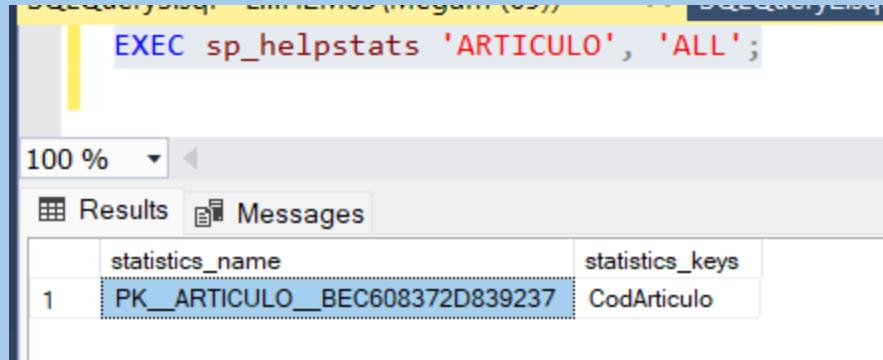
### ¿Cuándo se deben actualizar las estadísticas?

Cuando cambia la cantidad de datos:

- Muchas inserciones
- Muchas actualizaciones
- Muchas eliminaciones

Si las estadísticas están desactualizadas → mal rendimiento.

**Código útil sobre estadísticas**  
Ver estadísticas de una tabla  
**EXEC sp\_helpstats 'MiTabla', 'ALL';**

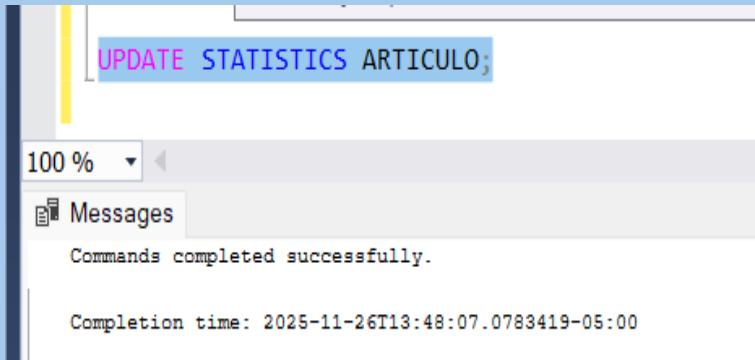


The screenshot shows a SQL Server Management Studio window with the following details:

- Query Editor: Shows the T-SQL command: `EXEC sp_helpstats 'ARTICULO', 'ALL';`
- Status Bar: Shows "100 %"
- Results Tab: Shows the output of the command:

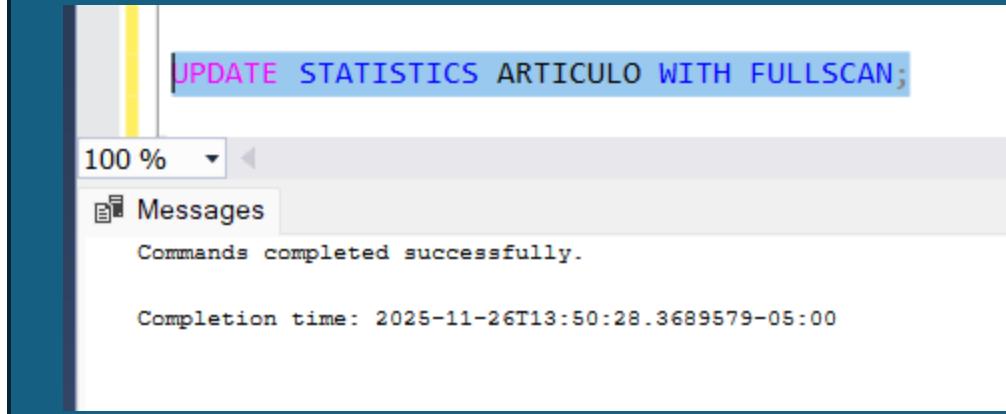
statistics_name	statistics_keys
PK_ARTICULO_BEC608372D839237	CodArticulo
- Messages Tab: Shows no messages.

Actualizar estadísticas manualmente  
**UPDATE STATISTICS MiTabla;**



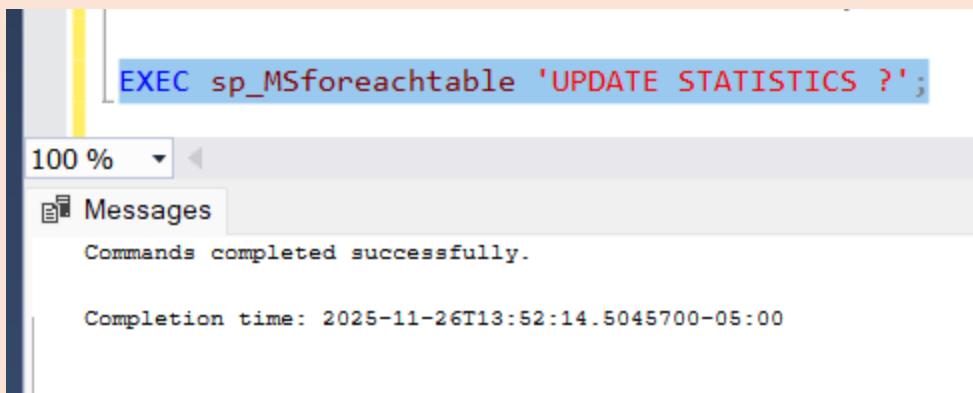
The screenshot shows a SQL query window with the following content:  
Text area: UPDATE STATISTICS ARTICULO;  
Progress bar: 100 %  
Messages area: Commands completed successfully.  
Completion time: 2025-11-26T13:48:07.0783419-05:00

Actualizar con FULLSCAN (más preciso)  
**UPDATE STATISTICS MiTabla WITH FULLSCAN;**



The screenshot shows a SQL query window with the following content:  
Text area: UPDATE STATISTICS ARTICULO WITH FULLSCAN;  
Progress bar: 100 %  
Messages area: Commands completed successfully.  
Completion time: 2025-11-26T13:50:28.3689579-05:00

Actualizar todas las estadísticas de una base de datos  
**EXEC sp\_MSforeachtable 'UPDATE STATISTICS ?';**



The screenshot shows a SQL query window with the following content:  
Text area: EXEC sp\_MSforeachtable 'UPDATE STATISTICS ?';  
Progress bar: 100 %  
Messages area: Commands completed successfully.  
Completion time: 2025-11-26T13:52:14.5045700-05:00

# ÍNDICES EN SQL SERVER

## ¿Qué es un índice?

Un índice es una estructura **tipo árbol (B-tree)** que acelera las búsquedas y ordenamientos en una tabla.

Funcionan igual que el índice de un libro:

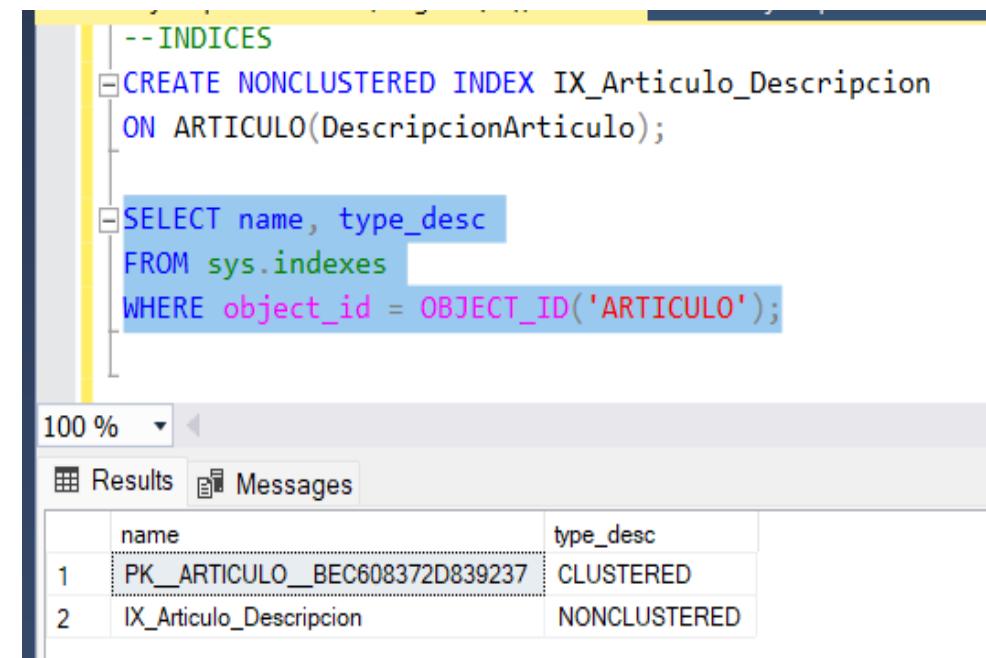
En vez de revisar página por página, vas directo al punto.

## Tipos principales de índices

### ◆ Índice Clustered (CLUSTERED)

- Ordena físicamente la tabla
- Solo puede existir uno por tabla
- Es más rápido cuando se busca por la clave primaria

Ejemplo:



The screenshot shows a SQL Server Management Studio window. The top pane displays T-SQL code for creating a non-clustered index and selecting data from the sys.indexes system catalog. The bottom pane shows the results of the query, which returns two rows: a clustered index named PK\_ARTICULO\_BEC608372D839237 and a non-clustered index named IX\_Articulo\_Descripcion.

```
-- INDICES
CREATE NONCLUSTERED INDEX IX_Articulo_Descripcion
ON ARTICULO(DescripcionArticulo);

SELECT name, type_desc
FROM sys.indexes
WHERE object_id = OBJECT_ID('ARTICULO');
```

name	type_desc
PK_ARTICULO_BEC608372D839237	CLUSTERED
IX_Articulo_Descripcion	NONCLUSTERED

# Índices Compuestos

Usan varias columnas:

```
CREATE NONCLUSTERED  
INDEX
```

```
IX_Ventas_Fecha_Cliente
```

```
ON Ventas(Fecha, ClienteID);
```

# Índices Únicos (UNIQUE)

Garantizan que no se dupliquen los valores:

```
CREATE UNIQUE INDEX  
IX_Usuarios_Correo ON  
Usuarios(Correo);
```

```
-- INDICES  
CREATE NONCLUSTERED INDEX IX_OrdenDetalle_Orden_Articulo  
ON ORDEN_DETALLE (NumOrden, CodArticulo);  
  
SELECT name, type_desc  
FROM sys.indexes  
WHERE object_id = OBJECT_ID('ORDEN_DETALLE');
```

Results

	name	type_desc
1	PK_ORDEN_DETALLE	CLUSTERED
2	IX_OrdenDetalle_Orden_Articulo	NONCLUSTERED

```
-- INDICES  
CREATE UNIQUE INDEX IX_Linea_Nombre  
ON LINEA(NomLinea);  
  
SELECT name, type_desc  
FROM sys.indexes  
WHERE object_id = OBJECT_ID('LINEA');
```

Results

	name	type_desc
1	PK_LINEA_C46A5B1A40FF4AF9	CLUSTERED
2	U_Linea_NomLinea	NONCLUSTERED
3	IX_Linea_Nombre	NONCLUSTERED

# FRAGMENTACIÓN EN ÍNDICES

## ¿Qué es la fragmentación?

Es el deterioro natural del índice cuando:

- Se insertan muchas filas
- Se actualizan valores del índice
- Se eliminan filas

La estructura del árbol se desordena y provoca:

- ✗ Más lecturas
- ✗ Pérdida de rendimiento
- ✗ Más uso de disco

## Tipos de fragmentación

### ◆ Fragmentación interna

Las páginas del índice tienen espacio vacío.

### ◆ Fragmentación externa

Las páginas ya no están en orden lógico → lecturas más lentas.

Cómo medir la fragmentación:

**SELECT**

```
index_id,  
avg_fragmentation_in_percent,  
page_count  
FROM sys.dm_db_index_physical_stats(DB_ID(),  
OBJECT_ID('MiTabla'), NULL, NULL, 'DETAILED');
```

The screenshot shows a SQL Server Management Studio window with the following details:

- Query Editor:** Contains the T-SQL code for checking index fragmentation.
- Status Bar:** Shows "100 %".
- Results Tab:** Contains a table with the following data:

	index_id	avg_fragmentation_in_percent	page_count
1	1	0	1
2	2	0	1
3	3	0	1

# MANTENIMIENTO DE ÍNDICES

¿Qué hacer cuando detectas fragmentación?

SQL Server recomienda:

Fragmentación	Acción recomendada
5% – 30%	REORGANIZE
> 30%	REBUILD

## **REORGANIZE (fragmentación leve)**

Es un proceso suave, no bloquea tanto la tabla:

`ALTER INDEX IX_Milndice ON MiTabla REORGANIZE;`

## **REBUILD (fragmentación alta)**

Reconstruye el índice desde cero.

Es más pesado, pero deja el índice como nuevo.

`ALTER INDEX IX_Milndice ON MiTabla REBUILD;`

Con compresión:

`ALTER INDEX IX_Milndice  
ON MiTabla  
REBUILD WITH (DATA_COMPRESSION = PAGE);`

**Reconstruir todos los índices de una tabla**  
**ALTER INDEX ALL ON MiTabla REBUILD;**

## **Script automático para mantenimiento**

```
DECLARE @SQL NVARCHAR(MAX) = '';
SELECT @SQL = @SQL +
'IF (SELECT avg_fragmentation_in_percent
    FROM sys.dm_db_index_physical_stats(DB_ID(), ' + CAST(OBJECT_ID AS VARCHAR)
+ ', NULL, NULL, "LIMITED") > 30
    ALTER INDEX ALL ON ' + QUOTENAME(name) + ' REBUILD;
'

FROM sys.objects
WHERE type = 'U';
EXEC (@SQL);
```

# 3. Administración de transacciones y bloqueos.

## Concepto de transacción

Una transacción es un conjunto de operaciones SQL que se ejecutan como una unidad lógica de trabajo.

- Todas las operaciones deben completarse correctamente para que se aplique la transacción.
- Si ocurre un error, ninguna operación se aplica y se revierte todo.

## Propiedades ACID

Toda transacción en SQL Server cumple las propiedades ACID:

Propiedad	Significado
<b>A</b> (Atomicidad)	Todas las operaciones de la transacción se completan o ninguna se aplica.
<b>C</b> (Consistencia)	La base de datos pasa de un estado válido a otro válido después de la transacción.
<b>I</b> (Aislamiento)	Las operaciones de una transacción no afectan a otras transacciones concurrentes hasta que se confirman.
<b>D</b> (Durabilidad)	Una vez que se confirma, los cambios son permanentes aunque falle el sistema.

# Sintaxis de transacciones en SQL Server

**BEGIN TRANSACTION;** -- Inicio de la transacción

-- Operaciones SQL

**UPDATE ARTICULO**

**SET StockActual = StockActual - 5**

**WHERE CodArticulo = 1;**

**INSERT INTO ORDEN\_DETALLE (NumOrden, CodArticulo,  
PrecioCompra, CantidadSolicitada)**

**VALUES (1001, 1, 15.50, 5);**

-- Confirmar cambios

**COMMIT TRANSACTION;**

-- Revertir cambios en caso de error

**-- ROLLBACK TRANSACTION;**

Explicación:

- **BEGIN TRANSACTION:** inicia la transacción.
- Bloque de operaciones: **INSERT, UPDATE, DELETE, etc.**
- **COMMIT:** aplica todos los cambios de manera permanente.
- **ROLLBACK:** deshace todos los cambios si algo falla.

# Niveles de aislamiento (Isolation Levels)

Controlan cómo interactúan las transacciones concurrentes:

Nivel de aislamiento	Qué permite	Ejemplo de efecto
<b>READ UNCOMMITTED</b>	Leer datos no confirmados	Puede leer registros que podrían revertirse (dirty reads)
<b>READ COMMITTED</b> (default)	Solo lee datos confirmados	Evita leer registros temporales
<b>REPEATABLE READ</b>	Bloquea filas leídas hasta finalizar la transacción	Evita que otras transacciones modifiquen esas filas
<b>SERIALIZABLE</b>	Bloquea rango de datos para evitar inserciones concurrentes	Garantiza ejecución secuencial
<b>SNAPSHOT</b>	Lectura consistente sin bloquear	Usa versión temporal de los datos

# Bloqueos (Locks) en SQL Server

¿Qué son los bloqueos?

Un lock protege los datos durante una transacción para garantizar la integridad.

- Evita que otras transacciones lean o modifiquen datos mientras están en uso.
- SQL Server administra los locks automáticamente.

## Tipos de bloqueos

Tipo de lock	Nivel	Qué bloquea
<b>Shared (S)</b>	Lectura	Permite leer pero no modificar
<b>Exclusive (X)</b>	Escritura	Bloquea lectura y escritura por otros
<b>Update (U)</b>	Evita deadlocks al actualizar	Se convierte en Exclusive si se aplica el cambio
<b>Intent (IS, IX)</b>	Señaliza intención de bloquear filas/páginas	Coordinación de locks a nivel de página o tabla

## Ejemplo práctico con bloqueos y transacciones

-- Transacción 1

```
BEGIN TRANSACTION;
```

```
UPDATE ARTICULO
```

```
SET StockActual = StockActual - 2
```

```
WHERE CodArticulo = 1;
```

-- Transacción 2 (ejecutada al mismo tiempo en otra sesión)

```
SELECT * FROM ARTICULO
```

```
WHERE CodArticulo = 1;
```

-- Dependiendo del nivel de aislamiento, la Transacción 2 puede esperar hasta que la primera haga COMMIT

```
COMMIT TRANSACTION;
```

## Buenas prácticas de transacciones y bloqueos

- Mantener transacciones cortas → evita bloqueos prolongados.
- Usar niveles de aislamiento apropiados según la necesidad de consistencia.
- Evitar operaciones masivas dentro de la transacción si es posible.
- Siempre manejar errores con TRY...CATCH y ROLLBACK.
- Monitorizar bloqueos y deadlocks usando SQL Profiler o Extended Events.

## Ejemplo con manejo de errores:

```
BEGIN TRY  
    BEGIN TRANSACTION;  
  
    UPDATE ARTICULO  
    SET StockActual = StockActual - 3  
    WHERE CodArticulo = 2;  
  
    COMMIT TRANSACTION;  
END TRY  
BEGIN CATCH  
    ROLLBACK TRANSACTION;  
    PRINT 'Ocurrió un error y se deshizo la transacción';  
END CATCH;
```

## Cómo ver bloqueos en SQL Server:

```
SELECT  
    r.session_id,  
    r.status,  
    r.command,  
    t.text AS SQLText,  
    l.resource_type,  
    l.resource_associated_entity_id,  
    l.request_mode  
FROM sys.dm_exec_requests r  
JOIN sys.dm_tran_locks l  
    ON r.session_id = l.request_session_id  
CROSS APPLY  
    sys.dm_exec_sql_text(r.sql_handle) t;
```

- Te muestra qué sesión está bloqueando qué recurso.
- Muy útil para optimizar transacciones concurrentes.

## 4. Análisis de planes de ejecución.

El plan de ejecución es la ruta que SQL Server utiliza para ejecutar una consulta.

Permite ver:

- Cómo accede SQL Server a los datos
- Si usa índices o escanea tablas completas
- Si hace joins eficientes o lentos
- Si hay problemas de rendimiento

Es la herramienta principal para optimizar consultas.

**Mostrar plan real ejecutado**

En SSMS activa:

**Include Actual Execution Plan (Ctrl + M)**

O ejecuta:

**SET STATISTICS PROFILE ON;  
GO**

# Operadores más importantes

## 1. Index Seek (BUENO)

Uso un índice para encontrar exactamente las filas.

- Rápido.
- Ideal.

Ejemplo que genera SEEK:

```
SELECT *  
FROM ARTICULO  
WHERE CodArticulo = 10;
```

## 2. Index Scan (REGULAR)

- Recorre todo el índice.
- Pasa cuando buscas por una columna que **no tiene índice**.

Ejemplo:

```
SELECT *  
FROM ARTICULO  
WHERE DescripcionArticulo LIKE  
'%leche%';
```

## 3. Table Scan (MALO)

- SQL Server recorre **toda la tabla**.
  - Lento en tablas grandes.  
Sucede cuando:
    - No hay índices adecuados.
      - Usas funciones en el WHERE.

## 4. Key Lookup

- SQL Server usa un índice NONCLUSTERED
- PERO necesita valores que no están en ese índice
- Entonces va al índice clustered a buscarlos

Ejemplo:

```
SELECT PrecioProveedor  
FROM ARTICULO  
WHERE DescripcionArticulo = 'Leche Gloria';
```

## 5. Hash Match

SQL Server crea una tabla hash para hacer joins.  
Bueno para grandes cantidades de datos.

## 6. Nested Loop

- SQL Server compara filas una por una.
- Excelente para joins pequeños.
- Lento si las tablas son grandes.

## Cómo ver el plan en formato texto

```
SET SHOWPLAN_TEXT ON;  
GO
```

```
SELECT *  
FROM ARTICULO  
WHERE StockActual > 10;
```

```
SET SHOWPLAN_TEXT OFF;  
GO
```

Concepto	Qué significa
Plan estimado	Antes de ejecutar
Plan real	Después de ejecutar
Seek	Rápido
Scan	Lento
Key Lookup	Puede optimizarse
Cost (%)	Dónde está el problema

## 5. Optimización de consultas T-SQL.

La optimización de consultas es el proceso de mejorar el rendimiento de instrucciones SQL para que se ejecuten más rápido y consuman menos recursos (CPU, memoria, E/S, bloqueos).

A continuación, los puntos más importantes:

### 5.1. Usar índices de manera eficiente

✓ Usa índices en columnas:

- usadas en *WHERE*
- *JOIN*
- *ORDER BY*
- *GROUP BY*

✗ Evita funciones sobre columnas indexadas

Ejemplo malo:

**SELECT \* FROM Ventas**

**WHERE YEAR(Fecha) = 2024;**

SQL Server NO usa el índice.

Ejemplo óptimo:

**SELECT \* FROM Ventas**

**WHERE Fecha >= '2024-01-01' AND Fecha < '2025-01-01';**

### 5.2. Evitar SELECT \*\*\*

Traer columnas innecesarias consume E/S, CPU y red.



Malo:

**SELECT \* FROM Cliente;**



Bueno:

**SELECT ClienteID, Nombre, Correo  
FROM Cliente;**

### 5.3. Usar JOINs correctos

✓ Evita subconsultas correlacionadas lentas:

👎 Malo:

```
SELECT Nombre,  
       (SELECT COUNT(*) FROM Ventas  
        WHERE Ventas.ClienteID =  
              Cliente.ClienteID)  
  FROM Cliente;
```

👍 Bueno:

```
SELECT c.Nombre, COUNT(v.VentaID)  
  FROM Cliente c  
 LEFT JOIN Ventas v ON c.ClienteID =  
                   v.ClienteID  
 GROUP BY c.Nombre;
```

### 5.4. Filtrar lo antes posible (puesta en escena)

SQL Server ejecuta más rápido cuando reduce filas al inicio.

👍 Bueno:

```
SELECT *  
  FROM Ventas
```

```
WHERE Fecha >= '2024-01-01';
```

👎 Malo (aplicar filtros después de JOIN):

```
SELECT *  
  FROM Ventas v  
 JOIN Cliente c ON v.ClienteID = c.ClienteID  
 WHERE v.Fecha >= '2024-01-01';
```

## 5.5. Evita OR cuando sea posible

Los OR rompen índices.

👎 Malo:

**WHERE Nombre = 'Ana' OR  
Apellido = 'Lopez';**

👍 Bueno (si quieres usar  
índices):

**WHERE Nombre = 'Ana'**

**UNION**

**SELECT ...**

**WHERE Apellido = 'Lopez';**

## 5.6. Evitar LIKE con comodín al inicio

✗ Esto NO usa índice:

**WHERE Nombre LIKE '%ana';**

✓ Esto SÍ usa índice:

**WHERE Nombre LIKE 'ana%';**

## 5.7. Evitar funciones y CAST en WHERE

✗ Malo:

**WHERE CAST(Fecha AS DATE) = '2024-06-01';**

✓ Bueno:

**WHERE Fecha >= '2024-06-01' AND Fecha <  
'2024-06-02';**

## 5.8. Usar variables con tipología adecuada

✗ Malo:

**DECLARE @dni VARCHAR(100);**

👍 Bueno:

**DECLARE @dni CHAR(8);**

Cuanto más grande la variable → más lenta la comparación.

# 6. Control de recursos con Resource Governor.

El Resource Governor es una característica de SQL Server que permite administrar y limitar el uso de recursos (CPU, memoria y solicitudes concurrentes) para distintos grupos de usuarios o cargas de trabajo. Sirve para evitar que una consulta pesada o un usuario consuma todos los recursos del servidor, afectando a los demás.

## 6.1. ¿Qué recursos controla?

Resource Governor puede limitar:

- ✓ CPU
  - Porcentaje mínimo / máximo asignado a cada grupo.
- ✓ Memoria
  - Cantidad máxima de memoria que puede consumir.
- ✓ Solicitudes simultáneas
  - Controla cuántas consultas puede ejecutar un grupo.

⚠ No controla:

- Disco (I/O) directamente
- Red
- Almacenamiento

## 6.2. Componentes principales

Resource Governor trabaja con 3 conceptos:

### 1 Workload Group (Grupo de Trabajo)

Es un "contenedor" donde SQL Server agrupa sesiones o consultas.

Define límites como:

- Máximo de CPU
- Mínimo de CPU
- Memoria permitida
- Cantidad de sesiones activas

Ejemplo de grupo:

```
CREATE WORKLOAD GROUP GrupoReportes
WITH
(
    MAX_DOP = 1,
    REQUEST_MAX_CPU_TIME_SEC = 5,
    IMPORTANCE = MEDIUM
);
```

### 2 Resource Pool (Pool de Recursos)

Es donde SQL Server asigna los recursos reales.

Un pool define:

- Min/Max CPU
- Min/Max memoria

Ejemplo:

```
CREATE RESOURCE POOL PoolReportes
WITH
(
    MIN_CPU_PERCENT = 10,
    MAX_CPU_PERCENT = 30,
    MIN_MEMORY_PERCENT = 10,
    MAX_MEMORY_PERCENT = 30
);
```

### 3 Classifier Function (Función Clasificadora)

Es una función que identifica **qué usuario, aplicación o base de datos** entra a qué grupo.

Ejemplo:

```
CREATE FUNCTION dbo.fnClasificarSesion()
```

```
RETURNS sysname
```

```
WITH SCHEMABINDING
```

```
AS
```

```
BEGIN
```

```
DECLARE @grupo SYSNAME;
```

```
IF (APP_NAME() = 'QhatuApp')
```

```
    SET @grupo = 'GrupoReportes';
```

```
ELSE
```

```
    SET @grupo = 'default';
```

```
RETURN @grupo;
```

```
END;
```

### 6.3. Cómo funciona el proceso

- Un usuario se conecta a SQL Server.
- La función clasificadora analiza:
  - nombre del login
  - nombre de la aplicación
  - base de datos
  - dirección IP
- SQL Server asigna esa sesión a un **grupo de trabajo**.
- El grupo usa recursos del **resource pool** configurado.
- Si el usuario intenta usar más CPU/memoria de lo permitido → SQL Server limita su ejecución.

## 6.4. Beneficios reales

- ✓ Evita que una consulta pesada bloquee el sistema

Si Qhatu suelta una consulta enorme, SQL Server la limita automáticamente.

- ✓ Protege procesos críticos

Ejemplo: evitar que reportes o consultas del usuario final afecten procesos de facturación o backups.

- ✓ Se puede controlar por aplicación, usuario o IP

Muy útil en servidores compartidos.

- ✓ Ajustes dinámicos

Se puede modificar sin reiniciar SQL Server.

## 6.5. Consultar estado del Resource Governor

Ver pools:

```
SELECT * FROM  
sys.resource_governor_resource_pools;
```

Ver grupos:

```
SELECT * FROM  
sys.resource_governor_workload_groups;
```

Ver configuración actual:

```
SELECT * FROM  
sys.dm_resource_governor_configuration;
```

A large, rectangular stained-glass window is set into a dark wooden frame. The window is divided into a grid of panes, each containing a different color or pattern. The colors range from deep reds and purples at the top to bright yellows and oranges near the bottom, creating a vibrant sunset effect. In the center of the window, there is a dark, silhouetted shape that looks like a tree or a cluster of rocks. The overall effect is one of a traditional church window, but with a modern, colorful twist.

**GRACIAS**