25

Consejos Élite de Administración Avanzada en SQL Server

Administrar SQL Server va mucho más allá de ejecutar consultas o hacer copias de seguridad.
El verdadero dominio implica entender el **motor interno**, anticiparse a los cuellos de botella, proteger los datos y mantener la disponibilidad constante.

Estos 25 consejos reúnen lo mejor de la práctica profesional:

Recuerda: cada entorno es distinto.
Prueba, mide, documenta y aplica con criterio.
Un DBA excelente no solo mantiene su servidor estable... **lo hace evolucionar**.

1️⃣ Optimize for Ad Hoc Workloads

📘 Contexto

Cada vez que SQL Server ejecuta una consulta, genera un **plan de ejecución** y lo almacena en caché para reutilizarlo. Sin embargo, en entornos donde se ejecutan miles de consultas distintas (por ejemplo, aplicaciones con ORM como Entity Framework o Hibernate), se generan **planes únicos que nunca se vuelven a usar**.
Esto **satura la memoria** con objetos inútiles, reduciendo el rendimiento global.

⚙️ Solución

SQL Server ofrece una opción para evitar almacenar un plan completo en la primera ejecución. Solo si la consulta se repite, se guarda el plan optimizado.

```
EXEC sp_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp_configure 'optimize for ad hoc workloads', 1;

RECONFIGURE;
```

🧩 Funcionamiento interno

Con esta opción activa, SQL Server guarda un **“stub plan”** (una versión ligera del plan) tras la primera ejecución.
Cuando la misma consulta se ejecuta de nuevo, el stub se reemplaza por el plan completo.
Esto libera espacio en el plan cache y reduce presión sobre la memoria de procedimiento.

🧠 Mejores prácticas

- Ideal para servidores con muchas consultas dinámicas o APIs.
- No recomendable para sistemas con pocas consultas repetitivas y parametrizadas.
- Supervisa el efecto usando:

```
SELECT COUNT(*) AS TotalPlans, SUM(size_in_bytes)/1024/1024 AS MB_Used

FROM sys.dm_exec_cached_plans;
```

2️⃣ Resource Governor

📘 Contexto

Cuando varios servicios comparten el mismo SQL Server (por ejemplo, ETL, reporting y transacciones), uno de ellos puede acaparar CPU y RAM.
El **Resource Governor** permite definir límites y prioridades de consumo para cada grupo de usuarios o conexiones.

⚙️ Ejemplo práctico

```
CREATE RESOURCE POOL PoolReportes WITH (MAX_CPU_PERCENT = 40,
MAX_MEMORY_PERCENT = 40);

CREATE WORKLOAD GROUP GrupoReportes USING PoolReportes;

ALTER RESOURCE GOVERNOR RECONFIGURE;
```

Luego, se asocia un inicio de sesión o aplicación al grupo mediante una función de clasificación:

```
CREATE FUNCTION dbo.fn_ClasificaConexion()

RETURNS sysname

AS

BEGIN

  IF (SUSER_NAME() = 'usuario_reportes')

     RETURN 'GrupoReportes';

  RETURN 'default';

END;

GO

ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION = dbo.fn_ClasificaConexion);

ALTER RESOURCE GOVERNOR RECONFIGURE;
```

🧩 Funcionamiento interno

El motor usa el **Scheduler interno** (SOS Scheduler) para asignar CPU y memoria a cada grupo.
Esto evita que procesos intensivos bloqueen el procesamiento de transacciones críticas.

🧠 Mejores prácticas

- Define grupos para ETL, Reporting, Testing, etc.
- No uses límites excesivamente restrictivos: pueden generar esperas.
- Monitorea con:

```
SELECT * FROM sys.dm_resource_governor_resource_pools;
```

3️⃣ Instant File Initialization

📘 Contexto

Cada vez que SQL Server crea o amplía un archivo MDF/NDF, Windows rellena el nuevo espacio con ceros por motivos de seguridad.
Esto puede hacer que **una restauración o creación de base de datos tarde minutos u horas**, especialmente en discos grandes.

⚙️ Activación

1. Abre secpol.msc.
2. Ve a: **Políticas locales → Asignación de derechos de usuario → Perform volume maintenance tasks**.
3. Agrega la cuenta del servicio SQL Server (por ejemplo: NT SERVICE\MSSQLSERVER).
4. Reinicia el servicio SQL Server.

🧩 Detalles técnicos

Al habilitar esta opción, SQL Server puede **asignar el espacio físico sin inicializarlo**, lo que acelera enormemente las operaciones de creación y crecimiento.
Solo aplica a archivos de datos (no a logs .ldf, que siempre deben inicializarse).

🧠 Mejores prácticas

- Altamente recomendable en entornos de producción.
- No hay riesgo si los discos están dedicados a SQL Server.
- Verifica con:

```
SELECT instant_file_initialization_enabled FROM sys.dm_server_services;
```

4️⃣ Monitorea la fragmentación de índices

📘 Contexto

La fragmentación ocurre cuando las páginas de un índice se almacenan en orden físico diferente al lógico.
Esto genera más lecturas y ralentiza el rendimiento de consultas, especialmente en grandes tablas.

⚙️ Detección

```
SELECT
   OBJECT_NAME(object_id) AS Tabla,
   index_id,
   avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'LIMITED')
WHERE avg_fragmentation_in_percent > 10
ORDER BY avg_fragmentation_in_percent DESC;
```

🧩 Interpretación

- 0–10%: sin acción necesaria.
- 10–30%: reorganiza (menos costoso).
- 30%: reconstruye (recrea completamente el índice).

```
ALTER INDEX NombreIndice ON MiTabla REORGANIZE;
ALTER INDEX NombreIndice ON MiTabla REBUILD;
```

🧠 Mejores prácticas

- Automatiza con un plan de mantenimiento inteligente.
- No reconstruyas todos los índices cada noche.
- Usa FILLFACTOR para reducir fragmentación futura.

```
ALTER INDEX ALL ON MiTabla REBUILD WITH (FILLFACTOR = 90);
```

5️⃣ Desactiva Auto Shrink

📘 Contexto

El parámetro AUTO_SHRINK reduce automáticamente el tamaño de los archivos de base de datos cuando hay espacio libre.
Aunque parece útil, en realidad causa **fragmentación y carga constante de I/O**.

⚙️ Solución

ALTER DATABASE MiBase SET AUTO_SHRINK OFF;

🧩 Funcionamiento interno

Cada ciclo de shrink mueve páginas de datos para liberar espacio físico. Esto genera fragmentación lógica, hace que los índices pierdan eficiencia y puede bloquear transacciones en curso.

🧠 Mejores prácticas

- Usa SHRINK solo tras eliminar grandes volúmenes de datos.
- Realiza mantenimiento posterior con DBCC DBREINDEX o ALTER INDEX REBUILD.
- Supervisa espacio con:

EXEC sp_spaceused;

6️⃣ Usa Extended Events en lugar de SQL Profiler

📘 Contexto

SQL Profiler fue durante años la herramienta estándar para capturar eventos, pero su principal problema es que **consume demasiados recursos** y puede afectar al rendimiento del propio servidor.
Desde SQL Server 2008, Microsoft introdujo **Extended Events (XE)**, una arquitectura ligera y moderna que permite capturar información de bajo nivel sin penalización significativa.

⚙️ Ejemplo práctico

Captura de bloqueos (deadlocks):

CREATE EVENT SESSION [MonitorBloqueos] ON SERVER

ADD EVENT sqlserver.lock_deadlock

ADD TARGET package0.event_file(SET filename='C:\Logs\Deadlocks.xel');

ALTER EVENT SESSION [MonitorBloqueos] ON SERVER STATE = START;

🧩 Funcionamiento interno

Extended Events trabaja sobre el **motor de diagnósticos** del kernel de SQL Server y permite seleccionar solo los eventos necesarios. La información se almacena en formato .xel, que puede analizarse con SQL Server Management Studio (SSMS) o Power BI.

🧠 Mejores prácticas

- Crea sesiones específicas (bloqueos, esperas, rendimiento de consultas).
- Evita capturar todos los eventos simultáneamente.
- Detén las sesiones tras el análisis para evitar crecimiento excesivo de archivos .xel.

7️⃣ Configura alertas para errores críticos (823, 824, 825)

📘 Contexto

Los errores 823, 824 y 825 indican **problemas físicos o lógicos** en el subsistema de disco.

- 823: error físico de lectura o escritura.
- 824: error lógico (corrupción).
- 825: advertencia de reintento en lectura (posible disco moribundo).
  Ignorarlos puede terminar en corrupción de datos irreversible.

⚙️ Ejemplo

```
EXEC msdb.dbo.sp_add_alert
  @name = 'Error 825 - Problema de E/S',
  @message_id = 825,
  @notification_message = 'Posible fallo de disco detectado';
```

Para notificar por correo, crea un operador y asócialo:

```
EXEC msdb.dbo.sp_add_operator
  @name = 'DBA_Alerta',
  @email_address = 'dba@empresa.com';


EXEC msdb.dbo.sp_add_notification
  @alert_name = 'Error 825 - Problema de E/S',
  @operator_name = 'DBA_Alerta',
  @notification_method = 1;
```

🧩 Funcionamiento interno

SQL Agent monitorea el registro de errores y dispara las alertas configuradas.
El 825 no detiene el proceso, pero **es una advertencia de hardware inminente**.

🧠 Mejores prácticas

- Configura alertas también para el error 9002 (log lleno) y 17883 (scheduler bloqueado).
- Asegúrate de tener Database Mail configurado para notificaciones automáticas.

8️⃣ Divide TempDB para mejorar la concurrencia

📘 Contexto

TempDB es el "área temporal global" del motor: se usa para consultas temporales, versiones de filas, operaciones de ordenación y objetos temporales.
El problema: si todos los hilos compiten por el mismo archivo físico, se producen **esperas en PFS/GAM/SGAM**.

⚙️ Solución

Agrega varios archivos del mismo tamaño:

ALTER DATABASE tempdb ADD FILE (NAME='tempdev2', FILENAME='D:\tempdb2.ndf', SIZE=512MB, FILEGROWTH=128MB);

ALTER DATABASE tempdb ADD FILE (NAME='tempdev3', FILENAME='D:\tempdb3.ndf', SIZE=512MB, FILEGROWTH=128MB);

🧩 Cómo funciona

SQL Server asigna páginas de forma circular entre los archivos de TempDB.
Más archivos = menor contención.
Regla de oro: un archivo por núcleo hasta 8. Más de eso raramente mejora rendimiento.

🧠 Mejores prácticas

- Usa discos rápidos (NVMe o SSD).
- Todos los archivos deben tener el mismo tamaño.
- No fragmentes TempDB en unidades distintas salvo por necesidad.

9️⃣ Ajusta MAXDOP y Cost Threshold for Parallelism

📘 Contexto

El paralelismo distribuye una consulta entre varios núcleos, pero no siempre mejora el rendimiento. Consultas pequeñas que se paralelizan pueden saturar CPU innecesariamente.

⚙️ Ajuste recomendado

```
EXEC sp_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp_configure 'max degree of parallelism', 4;  -- núcleos asignados

EXEC sp_configure 'cost threshold for parallelism', 50;  -- costo mínimo para paralelizar

RECONFIGURE;
```

🧩 Detalles técnicos

- MAXDOP: máximo de núcleos usados por una sola consulta.
- Cost Threshold: determina cuándo el optimizador decide paralelizar.
  El valor por defecto (5) está pensado para equipos antiguos, lo que hace que muchas consultas se paralelicen sin beneficio real.

🧠 Mejores prácticas

- Servidores OLTP: MAXDOP = 1 o 2.
- Servidores OLAP o BI: MAXDOP = 4–8.
- Evita el valor 0 (sin límite), que puede saturar la CPU.
  Monitorea efectos con:

```
SELECT * FROM sys.dm_exec_requests WHERE dop > 1;
```

🔟 Identifica las consultas más costosas

📘 Contexto

SQL Server almacena estadísticas de ejecución en la vista sys.dm_exec_query_stats. Analizarlas permite detectar **consultas lentas o con alto consumo de CPU** sin depender de herramientas externas.

⚙️ Consulta práctica

```
SELECT TOP 20
  qs.total_worker_time / qs.execution_count AS AvgCPU,
  qs.total_elapsed_time / qs.execution_count AS AvgTime,
  qs.execution_count,
  DB_NAME(st.dbid) AS DatabaseName,
  SUBSTRING(st.text, (qs.statement_start_offset/2)+1,
  ((CASE qs.statement_end_offset WHEN -1 THEN DATALENGTH(st.text)
  ELSE qs.statement_end_offset END - qs.statement_start_offset)/2)+1) AS QueryText
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) st
ORDER BY AvgTime DESC;
```

🧩 Interpretación

- **AvgCPU**: promedio de CPU por ejecución.
- **AvgTime**: duración media total.
- **execution_count**: número de veces que se ejecutó.
  Permite identificar consultas repetidas con bajo rendimiento y priorizar su optimización.

🧠 Mejores prácticas

- Usa DBCC FREEPROCCACHE con cuidado si modificas índices.
- Implementa Query Store para seguimiento histórico.
- Parametriza consultas y actualiza estadísticas para mejorar planes.

1️⃣ Habilita la compresión de backups por defecto

📘 Contexto

Por defecto, SQL Server guarda los backups sin compresión, ocupando mucho espacio y ralentizando operaciones de lectura/escritura.
Desde SQL Server 2008, puedes activar **la compresión por defecto**, reduciendo el tamaño de las copias hasta en un 70%.

⚙️ Comando

```
EXEC sp_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp_configure 'backup compression default', 1;

RECONFIGURE;
```

🧩 Funcionamiento

El motor aplica compresión en tiempo real durante el proceso de copia, sin requerir espacio temporal adicional.
Aunque consume algo más de CPU, en la mayoría de los entornos el beneficio en I/O y velocidad es notable.

🧠 Mejores prácticas

- Ideal en servidores donde el cuello de botella es el disco, no la CPU.
- Usa RESTORE VERIFYONLY para validar integridad post-copia.
- Monitoriza con:

```
SELECT backup_start_date, backup_finish_date, compressed_backup_size/1048576 AS MB_Comprimido

FROM msdb.dbo.backupset;
```

1️⃣2️⃣ Supervisa bloqueos en tiempo real

📘 Contexto

Los bloqueos son inevitables en sistemas concurrentes, pero pueden volverse críticos cuando una sesión impide el progreso de muchas otras.
SQL Server permite verlos en tiempo real mediante vistas dinámicas.

⚙️ Diagnóstico básico

```
SELECT
  blocking_session_id AS Bloqueador,
  session_id AS Bloqueado,
  wait_type, wait_time, wait_resource
FROM sys.dm_exec_requests
WHERE blocking_session_id <> 0;
```

🧩 Interpretación

- **Bloqueador:** sesión que posee el recurso.
- **Bloqueado:** sesión en espera.
- **wait_resource:** objeto específico (tabla, página, fila).

Para ver detalles de la consulta bloqueadora:

```
SELECT text
FROM sys.dm_exec_sql_text(sql_handle);
```

🧠 Mejores prácticas

- Nunca mates sesiones sin entender el contexto.
- Usa Extended Events (sqlserver.lock_deadlock_chain) para capturar bloqueos persistentes.
- Implementa índices o particiones si el origen son operaciones de tabla completa.

1️⃣3️⃣ Limpia planes de ejecución obsoletos

📘 Contexto

SQL Server almacena planes de ejecución para acelerar consultas repetidas. Sin embargo, tras grandes cambios (índices nuevos, estadísticas actualizadas, consultas parametrizadas), los planes antiguos pueden degradar el rendimiento.

⚙️ Comando

DBCC FREEPROCCACHE;

🧩 Efectos

Borra todos los planes almacenados, forzando al optimizador a regenerarlos la próxima vez.
Si solo quieres limpiar un plan específico:

DBCC FREEPROCCACHE (<sql_handle>);

🧠 Mejores prácticas

- Úsalo solo tras cambios estructurales.
- No lo ejecutes durante horario de producción.
- Verifica qué consultas están consumiendo más memoria:

SELECT TOP 20 usecounts, size_in_bytes, text

FROM sys.dm_exec_cached_plans

CROSS APPLY sys.dm_exec_sql_text(sql_handle)

ORDER BY size_in_bytes DESC;

1️⃣4️⃣ Habilita Query Store

📘 Contexto

Introducido en SQL Server 2016, el **Query Store** es una herramienta interna que guarda el historial de ejecución de consultas, sus planes y métricas de rendimiento.
Permite detectar cuándo un cambio de plan causa una caída de rendimiento.

⚙️ Activación

ALTER DATABASE MiBase SET QUERY_STORE = ON;

ALTER DATABASE MiBase SET QUERY_STORE (OPERATION_MODE = READ_WRITE);

🧩 Cómo funciona

Query Store almacena en disco información sobre cada consulta y plan generado, lo que permite comparar versiones de ejecución a lo largo del tiempo.

🧠 Mejores prácticas

- Actívalo solo en bases con carga significativa.
- Limpia datos antiguos con:

ALTER DATABASE MiBase SET QUERY_STORE CLEAR;

- Usa SSMS → Query Store → *Top Resource Consuming Queries* para diagnósticos visuales.

1️⃣5️⃣ Audita intentos de inicio de sesión fallidos

📘 Contexto

Los errores de autenticación pueden deberse tanto a configuraciones incorrectas como a ataques de fuerza bruta.
SQL Server los registra automáticamente en su log interno.

⚙️ Revisión rápida

```
EXEC xp_readerrorlog 0, 1, 'Login failed';
```

🧩 Explicación

Este procedimiento lee el registro de errores del servicio SQL. Cada entrada indica la hora, usuario y motivo del fallo.
Si quieres filtrarlo por usuario:

```
EXEC xp_readerrorlog 0, 1, 'Login failed for user', 'sa';
```

🧠 Mejores prácticas

- Revisa periódicamente los fallos para detectar patrones sospechosos.
- Configura alertas con SQL Agent para errores repetidos del mismo usuario.
- Si usas autenticación mixta, cambia contraseñas predeterminadas y desactiva cuentas inactivas.

1 6 Ejecuta DBCC CHECKDB periódicamente

📘 Contexto

Aunque SQL Server es estable, la corrupción de datos puede aparecer por fallos de disco, memoria, controladores o cortes de energía.
El comando **DBCC CHECKDB** valida la integridad lógica y física de una base de datos.
Ignorar este chequeo puede significar perder datos sin aviso previo.

⚙️ Comando

DBCC CHECKDB('MiBase') WITH NO_INFOMSGS, ALL_ERRORMSGS;

🧩 Funcionamiento

DBCC CHECKDB realiza internamente varios pasos:

1. Comprueba la consistencia de páginas, índices y metadatos.
2. Verifica relaciones lógicas entre tablas e índices.
3. Detecta corrupción a nivel de página o asignación.

🧠 Mejores prácticas

- Ejecuta semanalmente en horario de baja carga.
- Si la base es grande, hazlo en una réplica o copia restaurada.
- Automatiza alertas si detecta errores.
- Nunca uses REPAIR_ALLOW_DATA_LOSS sin respaldo previo.

1️⃣7️⃣ Desactiva xp_cmdshell (y actívala solo si es necesario)

📘 Contexto

xp_cmdshell permite ejecutar comandos del sistema operativo desde SQL Server. Es potente, pero también una puerta directa al sistema operativo.
Por seguridad, está deshabilitada por defecto.

⚙️ Activar y desactivar

-- Activar temporalmente

EXEC sp_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp_configure 'xp_cmdshell', 1;

RECONFIGURE;

-- Desactivar nuevamente

EXEC sp_configure 'xp_cmdshell', 0;

RECONFIGURE;

🧩 Riesgos y beneficios

- Permite automatizar tareas (copias, scripts de sistema).
- Pero expone el servidor a ejecución arbitraria si una cuenta comprometida accede al motor.

🧠 Mejores prácticas

- Crea una cuenta de proxy con permisos mínimos si debes usarla.
- Revisa auditorías (xp_logininfo) para controlar accesos.
- Desactívala inmediatamente tras su uso.

1 8 Realiza backups con checksum

📘 Contexto

Hacer backups regulares no garantiza que sean válidos.
El parámetro WITH CHECKSUM obliga a SQL Server a calcular sumas de verificación de cada página leída y almacenada, detectando corrupción durante la copia.

⚙️ Ejemplo

```
BACKUP DATABASE MiBase

TO DISK='C:\backups\MiBase.bak'

WITH CHECKSUM, STATS=10;
```

🧩 Funcionamiento

Durante el backup, el motor lee cada página de datos, genera un hash y lo compara con el valor almacenado en la cabecera de página.
Si se detecta inconsistencia, se detiene la operación.

🧠 Mejores prácticas

- Combina CHECKSUM con VERIFYONLY después del backup:

```
RESTORE VERIFYONLY FROM DISK='C:\backups\MiBase.bak' WITH CHECKSUM;
```

- Usa compresión y checksum juntos: son complementarios.
- Documenta y almacena backups fuera del entorno productivo.

1️⃣9️⃣ Aplica compresión de datos e índices

📘 Contexto

Las tablas grandes con datos históricos o consultas intensivas pueden ocupar mucho espacio y ralentizar lecturas.
SQL Server ofrece **compresión a nivel de fila o página** sin cambiar la lógica de la aplicación.

⚙️ Ejemplo

```
ALTER INDEX ALL ON Ventas REBUILD WITH (DATA_COMPRESSION = PAGE);
```

🧩 Funcionamiento

- **ROW**: comprime valores repetidos en columnas.
- **PAGE**: aplica algoritmos de compresión a páginas enteras.
  Ambos reducen el tamaño en disco y RAM, aunque PAGE exige más CPU al leer/descomprimir.

🧠 Mejores prácticas

- Ideal en tablas históricas, de solo lectura o poco actualizadas.
- Evita usarla en tablas de inserción continua.
- Usa esta vista para medir impacto:

```
SELECT object_name(object_id), data_compression_desc, row_count

FROM sys.partitions

WHERE data_compression <> 0;
```

2️⃣0️⃣ Implementa alta disponibilidad

📘 Contexto

En entornos críticos, las caídas no son opción. SQL Server ofrece varias tecnologías de **alta disponibilidad** según tus necesidades de recuperación.

⚙️ Opciones principales

1. **AlwaysOn Availability Groups**
   Réplicas sincronizadas que permiten lectura y failover automático.
2. **Failover Cluster Instances (FCI)**
   Redundancia total del servidor mediante Windows Cluster.
3. **Log Shipping**
   Copias asincrónicas del log de transacciones a otro servidor.

🧩 Funcionamiento

- AlwaysOn usa endpoints TCP para replicar cambios en tiempo real.
- Log Shipping ejecuta copias del log periódicamente y las restaura en el secundario.
- FCI depende de hardware compartido o SAN.

🧠 Mejores prácticas

- Evalúa tus RTO (tiempo máximo de caída) y RPO (pérdida de datos tolerable).
- Prueba escenarios de conmutación al menos cada trimestre.
- Documenta roles y secuencia de recuperación.

2️⃣1️⃣ Diseña planes de mantenimiento personalizados

📘 Contexto

El asistente de mantenimiento de SQL Server es rápido, pero demasiado genérico. Ejecuta tareas innecesarias y no optimiza según la fragmentación real o el uso.
Un plan de mantenimiento manual permite controlar qué índices, estadísticas y logs se gestionan y cuándo.

⚙️ Ejemplo básico de mantenimiento

```
-- Reorganizar índices fragmentados moderadamente

ALTER INDEX ALL ON MiTabla REORGANIZE;


-- Reconstruir índices muy fragmentados

ALTER INDEX ALL ON MiTabla REBUILD WITH (FILLFACTOR = 90);


-- Actualizar estadísticas

EXEC sp_updatestats;


-- Limpiar logs antiguos del agente

EXEC msdb.dbo.sp_purge_jobhistory @oldest_date = '2025-01-01';
```

🧩 Funcionamiento

Cada tarea debe ejecutarse en el orden correcto para evitar conflictos y reducir I/O.
Las estadísticas deben actualizarse después de reconstruir índices, ya que los REBUILD las actualizan automáticamente.

🧠 Mejores prácticas

- Programa los mantenimientos fuera de horas pico.
- Implementa scripts dinámicos que solo actúen sobre objetos con fragmentación real.
- Registra resultados en tablas de auditoría para medir mejoras.

2️⃣ Controla el crecimiento del log de transacciones

📘 Contexto

El archivo .ldf almacena todas las transacciones hasta que se realiza un **backup de log**.
Si no se copia con frecuencia, crece indefinidamente y puede llenar el disco.

⚙️ Backup del log

BACKUP LOG MiBase

TO DISK='C:\backups\MiBase_log.trn'

WITH INIT, NAME='Backup Log Automático';

🧩 Funcionamiento

Cada backup de log marca las porciones del archivo como reutilizables, permitiendo que SQL Server las sobrescriba.
Si la base está en **modo de recuperación FULL**, los backups del log son obligatorios.

🧠 Mejores prácticas

- Realiza backups de log cada 15–30 minutos en entornos de alta actividad.
- Revisa bases innecesariamente en FULL; cámbialas a SIMPLE si no requieren recuperación punto a punto.
- Monitorea crecimiento con:

DBCC SQLPERF(LOGSPACE);

2️⃣3️⃣ Habilita sys.dm_exec_query_plan_stats (SQL Server 2019+)

📘 Contexto

Antes de 2019, solo podías capturar planes de ejecución en tiempo real mediante Profiler o Extended Events.
Con la configuración **LAST_QUERY_PLAN_STATS**, SQL Server guarda automáticamente el último plan ejecutado para cada consulta.

⚙️ Activación

```
ALTER DATABASE SCOPED CONFIGURATION SET LAST_QUERY_PLAN_STATS = ON;
```

🧩 Funcionamiento

Esta opción crea instantáneas ligeras de los planes, accesibles desde sys.dm_exec_query_plan_stats.
Permite ver cómo cambia el rendimiento sin activar rastreos pesados.

🧠 Mejores prácticas

- Ideal para entornos de desarrollo y diagnóstico.
- Evita dejarlo siempre activo en producción si hay millones de consultas únicas.
- Analiza planes almacenados:

```
SELECT * FROM sys.dm_exec_query_plan_stats;
```

2️⃣4️⃣ Sustituye triggers innecesarios por lógica controlada

📘 Contexto

Los **triggers** son potentes, pero también ocultos y costosos.
Cada trigger añade lógica implícita a las operaciones DML (INSERT, UPDATE, DELETE), afectando el rendimiento y dificultando el mantenimiento.

⚙️ Alternativas recomendadas

- Usa **procedimientos almacenados** para encapsular reglas de negocio.
- Implementa **CHECK constraints** para validaciones simples.
- Si el trigger es inevitable, asegúrate de limitarlo:

```
IF TRIGGER_NESTLEVEL() > 1 RETURN;
```

🧩 Funcionamiento

Un trigger se ejecuta dentro de la transacción principal. Si algo falla dentro del trigger, la operación entera hace rollback.
Esto puede causar bloqueos y latencia inesperada.

🧠 Mejores prácticas

- Documenta claramente cada trigger activo.
- Evita lógica condicional compleja o llamadas a otras bases.
- Usa auditorías automáticas en lugar de triggers de logging.

2️⃣5️⃣ Usa sp_WhoIsActive para diagnóstico avanzado

📘 Contexto

Creada por Adam Machanic, esta herramienta es un estándar entre DBAs.
Proporciona en una sola vista información sobre sesiones activas, bloqueos, uso de CPU, I/O, memoria, esperas y consultas en ejecución.

⚙️ Ejemplo de uso

EXEC sp_WhoIsActive;

Puedes añadir parámetros opcionales:

EXEC sp_WhoIsActive @get_plans = 1, @get_locks = 1;

🧩 Funcionamiento

El procedimiento consulta vistas dinámicas como sys.dm_exec_requests, sys.dm_os_waiting_tasks y sys.dm_exec_query_stats, presentando la información consolidada y legible.

🧠 Mejores prácticas

- Ejecuta sp_WhoIsActive en servidores en producción cuando notes lentitud.
- Automatiza su ejecución cada minuto en caso de incidentes.
- Guarda resultados en una tabla temporal para análisis histórico:

EXEC sp_WhoIsActive @destination_table = 'DBA_Monitor';

📦 Disponible en: https://whoisactive.com

🧩 Conclusión general

Administrar SQL Server va mucho más allá de ejecutar consultas o hacer copias de seguridad. El verdadero dominio implica entender el **motor interno**, anticiparse a los cuellos de botella, proteger los datos y mantener la disponibilidad constante.

Estos 25 consejos reúnen lo mejor de la práctica profesional:

- **Optimización del rendimiento** (caché, índices, paralelismo).
- **Diagnóstico proactivo** (Extended Events, Query Store, sp_WhoIsActive).
- **Seguridad y continuidad** (CHECKDB, checksum, alta disponibilidad).
- **Automatización inteligente** (planes personalizados, control del log).

Recuerda: cada entorno es distinto.
Prueba, mide, documenta y aplica con criterio.
Un DBA excelente no solo mantiene su servidor estable… **lo hace evolucionar**.

📘 Sobre el autor

Óscar de la Cuesta
@oscardelacuesta — palentino.es

Apasionado de la tecnología, las bases de datos y la administración de sistemas, dedico mi trabajo, blog y presencia en redes a **compartir conocimiento práctico y accesible**, ayudando a otros profesionales a optimizar sus infraestructuras y mejorar el rendimiento de sus entornos de datos.

Con esta **Guía Élite de SQL Server**, busco ofrecer una visión completa —desde la optimización interna del motor hasta la seguridad, el diagnóstico y la alta disponibilidad—, para que cualquier administrador o técnico pueda dominar su entorno con rigor y confianza.

Autor: Óscar de la Cuesta Campillo
@oscardelacuesta
Palentino.es

.