PRIMUS DATA PRESS

CUADERNOS

*Cuaderno N.º 009*

SERIE · TRUCOS DE T-SQL PARA TU DW

# Delayed Durability

*El commit que confirma sin esperar al disco: cuándo acelera de verdad una carga lenta, cuánto, y el precio que cobra.*

**Javier Loria**

*Inteligencia de Negocios*

Primus Data

Inteligencia de Negocios y Analítica Avanzada

# ¿Por qué mi carga tarda horas si la CPU está ociosa y el disco no se satura?

El proceso se arrastra toda la noche, y el culpable no es el cómputo sino una espera invisible: cada `COMMIT` se queda parado esperando a que el **log de transacciones** se grabe en disco. Con miles de transacciones chicas, esa espera — `WRITELOG` — se vuelve el techo.

La palanca poco conocida es **Delayed Durability**: el commit que confirma sin esperar al disco. Este cuaderno muestra cuándo ayuda de verdad, cuánto cambia —con **benchmark**, no con promesas— y el precio que cobra: una ventana de posible pérdida de datos.

> *Así lo hacemos en Primus Data — y así se lo enseñamos al personal técnico de nuestros clientes en nuestras mentorías.*

✦

LA SERIE

*Novena entrega de la serie Trucos de T-SQL para tu DW, nacida de la charla del mismo nombre en* **Charlemos de SQL Server 2026**. *A diferencia de los otros trucos, este no salió de una línea del EDW que auditamos: salió de un* **patrón** *—esas cargas de miles de transacciones diminutas que toda la noche golpean el mismo cuello de botella—. La* **Cooperativa MuuSQL** *reproduce esos casos sin exponer al cliente; la base vive en el repo de la serie.*

ANTES DE LAS NOTAS

# El mensajero que no espera

Pensá en un mensajero que te trae paquetes todo el día. La regla estricta es: no se va hasta que firmes el recibido *y* lo guardes en la bodega, con candado. Cada paquete, la misma ceremonia. Si llegan mil paquetes, el mensajero pasa el día parado en tu puerta esperando que termines de archivar.

La regla relajada es otra: firmás el recibido y el mensajero ya se va al siguiente. Los paquetes van a la bodega después, cuando se juntan varios. Vas muchísimo más rápido. El riesgo aparece solo si se te incendia la casa entre que firmaste y guardaste: esos paquetes —los que estaban en el pasillo— los perdés.

### LO MISMO, SIN METÁFORA

Eso es Delayed Durability. El **COMMIT**, por defecto, es el mensajero estricto: espera a que el log se grabe en disco antes de devolverte el control. La versión relajada confirma la transacción y sigue; el log se vacía después, en lote.

— **J. L.**, *mirando correr un loop de 25,000 commits.*

EL PROBLEMA

# El COMMIT espera al disco, una y otra vez

Toda transacción durable termina igual: antes de que SQL Server te diga «listo, confirmado», su registro tiene que estar grabado en el log de transacciones, en disco. Eso es un *log flush*, y es sincrónico: la sesión se queda esperando. Esa espera tiene nombre en las estadísticas: `WRITELOG`.

Con una carga normal no lo notás. El problema aparece con **muchas transacciones chicas, una atrás de otra**: la romana grabando un pesaje por camión, un proceso que escribe en un log fila por fila, un retry loop que confirma en cada vuelta. 25,000 transacciones diminutas son 25,000 log flushes, y la sesión pasa la mayor parte de su vida esperando al disco —no calculando, esperando.

EL SÍNTOMA

```
-- ¿En qué se va el tiempo? Mirá las esperas de la sesión
SELECT wait_type, wait_time_ms
FROM   sys.dm_exec_session_wait_stats
WHERE  session_id = @@SPID
ORDER BY wait_time_ms DESC;
-- WRITELOG arriba de todo = el COMMIT esperando al disco
```

REGLA DE CAMPO

*Si una carga lenta pasa su tiempo en `WRITELOG` y no en CPU, no tenés un problema de cómputo: tenés un problema de cuántas veces tocás el disco.*

EL TRUCO

# Confirmá ya, vaciá el log después

Delayed Durability le cambia la regla al COMMIT. En vez de esperar el flush, SQL Server marca la transacción como confirmada de inmediato y deja su registro en un buffer de log en memoria. El vaciado a disco ocurre después.

EL COMMIT, ANTES Y DESPUÉS

```
-- Durabilidad plena (el default del motor)
COMMIT TRAN;                                  -- se queda esperando el flush

-- Durabilidad diferida
COMMIT TRAN WITH (DELAYED_DURABILITY = ON);   -- confirma ya; vacía luego
```

El vaciado se dispara cuando pasa alguna de tres cosas firmes: el buffer se llena (~60 KB), corre `sp_flush_log`, o **cualquier transacción plenamente durable en la misma base** confirma —y de paso arrastra al disco todas las diferidas pendientes. Así, muchas transacciones comparten un mismo flush en lugar de pagar uno cada una.

> REGLA DE CAMPO
>
> *El flush diferido no es un reloj. Se dispara por tres cosas firmes —buffer lleno, `sp_flush_log`, o cualquier transacción durable que confirme en la misma base—. El motor también intenta vaciar por tiempo, pero sin intervalo garantizado: no contés con un «cada segundo».*

A NIVEL DE BASE

# FORCED manda, ALLOWED delega

Antes de que un COMMIT pueda diferirse, la base tiene que dejarlo. Hay tres estados, y conviene tenerlos claros:

LOS TRES ESTADOS

```
-- 0) DISABLED — el default. Nadie difiere, aunque el COMMIT lo pida.

-- 1) FORCED — toda transacción se difiere, diga lo que diga el COMMIT.
ALTER DATABASE MuuSQL_DW SET DELAYED_DURABILITY = FORCED;

-- 2) ALLOWED — la base delega: el COMMIT decide.
ALTER DATABASE MuuSQL_DW SET DELAYED_DURABILITY = ALLOWED;
```

En `FORCED`, toda transacción se difiere aunque el COMMIT no diga nada. En `DISABLED` —el default— nadie difiere, aunque el COMMIT lo pida. En `ALLOWED`, el COMMIT tiene la última palabra: el que lleva `WITH (DELAYED_DURABILITY = ON)` se difiere; el resto queda con durabilidad plena.

> REGLA DE CAMPO
>
> `FORCED` y `DISABLED` *mandan sobre el COMMIT;* `ALLOWED` *delega. Usá* `FORCED` *para una ventana de carga entera;* `ALLOWED` *para diferir solo las transacciones que elegís y dejar el resto plenas.*

EL BENCHMARK

# Los flushes caen; el tiempo, solo en proporción

En la demo, las mismas 25,000 filas, el mismo loop; lo único distinto es la durabilidad. Los log flushes caen de unos 25,000 —uno por fila— a unos pocos cientos, uno por buffer lleno. Y el `WRITELOG` casi desaparece, porque el COMMIT dejó de esperar al disco.

MISMAS 25,000 FILAS, DOS DURABILIDADES

**Plena:** ~25,000 log flushes · `WRITELOG` domina la espera · la carga se arrastra.

**Diferida:** unos pocos cientos de flushes · `WRITELOG` casi cero · la misma carga, una fracción del tiempo.

El tiempo total baja solo en proporción a cuánto pesaba esa espera. Ahí está la trampa: si `WRITELOG` ya era chico, no hay casi nada que ganar —por eso la demo mide en vez de prometer.

> REGLA DE CAMPO
>
> *La mejora de Delayed Durability es proporcional a cuánto pesaba `WRITELOG`. Medilo antes; si ya era chico, relajar durabilidad no te va a salvar.*

LA PRIMERA TRAMPA

# No es «asíncrono gratis»

Es fácil tomárselo a la ligera por el nombre. Delayed Durability **no** relaja la atomicidad, ni la consistencia, ni el aislamiento. Relaja exactamente una letra de ACID: la **D**, la durabilidad.

UNA SOLA LETRA

```
A   atomicidad      intacta
C   consistencia    intacta
I   aislamiento     intacto
D   durabilidad     ← la única que se relaja
```

Hay una ventana. Si el motor cae de forma no limpia —corte de energía, un kill al proceso, falla de hardware— entre el COMMIT y el siguiente flush, esas transacciones que la aplicación creyó guardadas se pierden en el recovery. La base no queda corrupta: queda consistente, pero más vieja. Perdés trabajo confirmado, no integridad.

> REGLA DE CAMPO
>
> *Eso reduce la decisión a una pregunta: ¿el dato es reproducible? Pagos, asientos contables, una boleta que no podés volver a leer del origen: nunca. Staging, telemetría, una carga que re-corrés desde el archivo del día: con confianza.*

LA CONDICIÓN REAL

# «Reproducible» es media verdad

La condición de verdad, en un DW o un data mart, no es solo que el dato sea reproducible: es que la **carga sea idempotente** —que la puedas volver a ejecutar y dé el mismo resultado, aun cuando la primera vez haya quedado a medias.

Las dimensiones y catálogos que cargás incrementalmente por hash —el truco del `LoadHash` del **primer cuaderno** de la serie— son el caso fácil: volvés a correr, el hash dice qué cambió y lo que ya estaba no se reescribe. Las tablas de hechos piden más cuidado: al levantar de una caída, tienen que saber si reinician desde cero o cómo retoman la carga trunca.

SI TU PROCESO YA ES IDEMPOTENTE

...la ventana de pérdida de Delayed Durability deja de doler: lo que no se grabó, la próxima corrida lo vuelve a poner. Por eso vive bien donde tus cargas ya son re-ejecutables —y se vuelve peligroso donde no lo son.

REGLA DE CAMPO

*Delayed Durability pide cargas idempotentes: re-ejecutables al mismo resultado, aun tras una caída parcial. Si la pérdida de la última transacción duele y no podés re-correr, no lo toqués.*

LA SEGUNDA TRAMPA

# No acelera un commit único: agrupá primero

Delayed Durability ataca el costo de *muchos* flushes. Si tu carga ya es un solo `INSERT` set-based, hay un commit y un flush: no hay nada que diferir, y activarlo no cambia nada.

Por eso, antes de relajar la durabilidad, hacé la pregunta incómoda: **¿por qué hay 25,000 commits?** Muchas veces la respuesta es un fila-por-fila que debió ser set-based, o un proceso que confirma por iteración pudiendo agrupar N filas por transacción. En la demo, esas mismas 25,000 filas en un único `INSERT ... SELECT` con durabilidad plena le ganan a todo lo demás —un commit, un flush, cero espera repetida— y sin renunciar a nada.

APUNTE OPERATIVO

```
-- Al cerrar una ventana de carga diferida: forzá el vaciado a mano
EXEC sys.sp_flush_log;
-- acota cuánto trabajo confirmado queda expuesto a una caída
```

REGLA DE CAMPO

*Antes de cambiar durabilidad, mirá si podés cambiar el patrón de commits. Agrupar gana sin riesgo; diferir gana con riesgo. Delayed Durability es para cuando* ***no podés*** *re-arquitectar: un ORM fila por fila, un proceso de terceros que no vas a reescribir.*

LOS LÍMITES

# Tres lugares donde no es opción

Es una palanca solo de SQL Server (2014+) y Azure SQL Database. Y a un DW le tocan de cerca tres lugares donde no existe o no debe usarse:

- ✕ **Fabric Warehouse.** No existe: el almacenamiento es distribuido (OneLake, Parquet/Delta por debajo), no hay log de transacciones clásico ni palanca de durabilidad que ajustar. En la nube administrada, la durabilidad no es tuya.
- ✕ **Availability Groups y mirroring.** Las diferidas no garantizan durabilidad ni en el primario ni en los secundarios, y el control vuelve al cliente antes del acuse de la réplica síncrona. El log backup y el log shipping solo se llevan lo ya endurecido.
- ✕ **Replicación transaccional y CDC.** No soportado: desde SQL Server 2022 CU2 / 2019 CU20 el motor rechaza habilitar uno sobre el otro (errores 22891/22892). Si tu fact está bajo CDC, descartalo de entrada.

> REGLA DE CAMPO
>
> *Lo set-based viaja entre motores; una palanca de durabilidad del motor, no. En Fabric, bajo réplica síncrona o bajo CDC, ni lo consideres —sin importar si el dato es reproducible.*

PARA LLEVAR

# Cinco cosas sobre Delayed Durability

— **El COMMIT por defecto espera al disco.** Con muchas transacciones chicas, esa espera ( `WRITELOG` ) es el techo de rendimiento —no la CPU.

— **Difiere el flush, no relaja ACID salvo la D.** El COMMIT confirma ya; el log se vacía en lote. Atomicidad y consistencia quedan intactas.

— **El precio es una ventana de pérdida.** Ante una caída no limpia perdés lo aún no vaciado. Usalo solo donde la carga es **idempotente**: staging, telemetría, dimensiones por hash.

— **Primero preguntá por qué hay tantos commits.** Agrupar en set-based suele ganar más, y sin riesgo. Diferir es para cuando no podés re-arquitectar.

— **Hay donde no es opción:** Fabric (no existe), Availability Groups/mirroring (no garantiza durabilidad en réplicas) y tablas bajo replicación o CDC (el motor lo rechaza).

*— El script completo —con benchmark A/B, medición de `WRITELOG` y conteo de log flushes— está en el repo de la serie. Corre en SQL Server 2014+ con un F5 (la medición por sesión necesita 2016+); corrélo una sola vez, en una conexión y con la base en reposo.*

PARA SEGUIR LEYENDO

# Estas notas siguen abiertas.

Si algún patrón te resonó, o te pareció equivocado, me interesa saberlo. La conversación técnica de fondo vive en el blog, y casi siempre mejora con quien la discute.

primusdata.net/blog · primusdata.net/recursos

---

*Cuadernos Primus N.º 009, de la serie Trucos de T-SQL para tu DW. Texto compuesto en Philosopher; títulos y código en Expletus Sans y monoespaciado. Patrones auditados en un EDW real. Las opiniones son del autor; los errores, también.*

**Javier Loria**
*para Primus Data Press*