PRIMUS DATA PRESS

CUADERNOS

Cuaderno N.º 008

SERIE · TRUCOS DE T-SQL PARA TU DW

El truco del MERGE + OUTPUT

SCD Tipo 2 atómico: cerrar la versión vieja y abrir la nueva en una sola sentencia, sin pisar la historia.

Javier Loria

Inteligencia de Negocios

Primus Data

Inteligencia de Negocios y Analítica Avanzada

POR QUÉ IMPORTA

Si actualizás una dimensión sobrescribiendo el dato viejo, los reportes del año pasado dejan de cuadrar.

Cuando una dimensión de tu **data warehouse** cambia —un cliente se muda, un productor sube de categoría, un producto cambia de precio— y actualizás el dato *encima*, borrás lo que era cierto antes. El costo aparece después: los reportes muestran el precio de hoy sobre las ventas de ayer.

Mantener ese **historial de cambios** sin pisarlo es el **SCD Tipo 2**. Este cuaderno lo resuelve con una sola sentencia — `MERGE` con `OUTPUT $action` —: cierra la versión vieja, abre la nueva y deja, de regalo, la **auditoría** de qué cambió, todo atómico.

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

✦

LA SERIE

Octava entrega de la serie Trucos de T-SQL para tu DW, nacida de la charla del mismo nombre en **Charlemos de SQL Server 2026**. *Los patrones salen de auditar un EDW real de retail/agroindustria con más de diez años en producción; la* **Cooperativa MuuSQL** *reproduce esos casos sin exponer al cliente. La base de demo vive en el* repo de la serie.

ANTES DE LAS NOTAS

La historia clínica no se sobrescribe

Cuando cambiás de dirección, el doctor no borra la anterior y escribe la nueva encima: agrega una entrada, con su fecha, y la vieja queda como estaba. Al que lea el expediente en cinco años le importa no solo dónde vivís hoy, sino dónde vivías cuando te recetaron aquello.

Una dimensión SCD2 es exactamente eso: un expediente. Y al que lo lea en cinco años le importa el dato de entonces, no solo el de hoy.

DÓNDE QUEDÓ LA CARD N.º 001

El cuaderno **N.º 001** de esta serie se quedó a medio camino: detectaba qué productores habían cambiado, pero los actualizaba *encima*, borrando la versión anterior. Eso es SCD Tipo 1: rápido, y amnésico.

Acá cerramos el círculo con el Tipo 2.

— **J. L.**, *entre una sesión de estudio y un cliente.*

EL PROBLEMA

Cerrar y abrir sin desincronizar

En la Cooperativa MuuSQL la dimensión de productores guarda historia con dos fechas: `DimStart` y `DimEnd`. `DimEnd IS NULL` marca la versión vigente.

Un productor como `P0006` ya vivió tres vidas: dos cerradas y una vigente. Cuando llega un cambio no queremos pisar la vigente —hay que **cerrarla** (ponerle `DimEnd`) y **abrir una nueva**. Dos operaciones que pasan juntas o no pasan.

EL REFLEJO: DOS PASOS

```
-- paso 1: cerrar la vigente que cambió
UPDATE p SET p.DimEnd = @Hoy FROM dim.Productores p
JOIN #Stg s ON s.CodigoProductor = p.CodigoProductor
WHERE p.DimEnd IS NULL AND p.LoadHash <> s.LoadHash;
-- paso 2: abrir la nueva (¿y los nuevos?)
INSERT INTO dim.Productores ( ... ) SELECT  ...  FROM #Stg;
```

Funciona, pero lee el origen dos veces, y si el paso 2 falla sin transacción quedan versiones cerradas sin reemplazo: el expediente con una página arrancada.

> REGLA DE CAMPO
>
> *Cerrar la versión vieja y abrir la nueva son una sola operación. Si se pueden ejecutar por separado, tarde o temprano se ejecutan por separado.*

EL TRUCO

MERGE cierra, OUTPUT captura

MERGE combina INSERT, UPDATE y DELETE en una sentencia; su OUTPUT devuelve $action —qué le pasó a cada fila— y lo captura. Esas dos piezas arman el SCD2:

MERGE + OUTPUT

```
MERGE dim.Productores AS T
USING #Stg AS S
  ON  T.CodigoProductor = S.CodigoProductor
  AND T.DimEnd IS NULL                    -- solo la vigente
WHEN MATCHED AND T.LoadHash <> S.LoadHash THEN
  UPDATE SET T.DimEnd = @Hoy              -- cerrá la anterior
WHEN NOT MATCHED BY TARGET THEN
  INSERT ( ... ) VALUES ( ... )           -- productor nuevo
OUTPUT $action, S.* INTO #Cambios;
```

En una pasada cierra las vigentes que cambiaron (el LoadHash distinto las detecta) e inserta las nuevas, y deja en #Cambios qué pasó. Falta abrir la versión nueva de las cerradas:

ABRIR LA NUEVA

```
INSERT INTO dim.Productores (CodigoProductor, ... , DimStart, DimEnd)
SELECT CodigoProductor, ... , @Hoy, NULL
FROM   #Cambios
WHERE  Accion = 'UPDATE';
```

REGLA DE CAMPO

*La atomicidad la da el BEGIN TRAN, no el MERGE. Lo que el MERGE aporta es la **pasada única** y la bitácora en $action, de regalo.*

POR QUÉ UNA SOLA SENTENCIA

Componible — y sin re-leer el target

El resultado del `MERGE` se puede componer: `#Cambios` ya trae los valores nuevos, así que el `INSERT` no re-une al staging. El patrón hasta se colapsa en una sentencia, con el `INSERT` consumiendo el `OUTPUT` directo:

TODO EN UNA

```
INSERT INTO dim.Productores ( ... )
SELECT C.CodigoProductor,  ... , @Hoy, NULL
FROM ( MERGE  ... OUTPUT $action AS Accion, S.* ) AS C
WHERE C.Accion = 'UPDATE';
```

Luce mejor en una charla, pero tiene letra chica —la tabla destino no puede tener triggers, ni joins ni `GROUP BY` en esa proyección—; en ETL real suele ganar la tabla intermedia, que es además la bitácora. Y hay una ventaja más fina: dos sentencias leen el target dos veces, y en `READ COMMITTED` entre esas lecturas se cuela un *non-repeatable read* o un *phantom*. Una sola sentencia no tiene ese hueco.

> REGLA DE CAMPO
>
> *Dos sentencias dejan un hueco entre lecturas; en* `READ COMMITTED` *ahí entran non-repeatable y phantom reads. Una sola no, y te ahorra subir el aislamiento. Asumí una carga a la vez; con cargas paralelas,* `HOLDLOCK` *sobre el target.*

DE REGALO

$action es la bitácora de la carga

La misma tabla `#Cambios` que alimentó el `INSERT` te dice qué pasó, sin trabajo extra:

BITÁCORA

```
SELECT Accion, Filas = COUNT(*)
FROM   #Cambios
GROUP BY Accion;
```

En nuestro lote: `'INSERT'` = 30 (los nuevos), `'UPDATE'` = 60 (los que cambiaron y reabrimos). Los 60 idénticos y los 20 que venían sucios pero, normalizados, eran iguales, **no aparecen**: el hash empató, ninguna cláusula `WHEN` disparó, el `OUTPUT` no emitió fila. Esa ausencia es la respuesta correcta —«nada que hacer»— y queda registrada por omisión.

> REGLA DE CAMPO
>
> `$action` *es auditoría gratis. Cuántas altas, cuántas versiones nuevas, qué llaves de negocio tocó la carga: todo sale del mismo* `OUTPUT`*.*

UNA NOTA AL MARGEN

Dos cargas el mismo día: duración cero

El grano de la versión es el día, no el evento: `@Hoy` es un `DATE`. Si la misma carga corre dos veces en un día y ambas traen cambios para el mismo productor, la segunda cierra la versión que la primera abrió: queda una fila con `DimStart = DimEnd`.

No es un error, y para point-in-time son inofensivas: el rango `@fecha ≥ DimStart AND @fecha < DimEnd`, con `DimStart = DimEnd`, es vacío —nunca salen como vigentes.

CUÁNDO DEJA DE SER RUIDO

El problema aparece si entre una carga y la otra **cargaste hechos**. Si una tabla de hechos se cargó contra la versión que después quedó en duración cero, esos hechos apuntan —por su llave subrogada— a una versión que el calendario considera que nunca existió.

REGLA DE CAMPO

Con grano de día, dos cargas con cambios el mismo día dejan versiones de duración cero. Inofensivas para point-in-time; trampa si cargás hechos entre corridas. Serializá las cargas, o subí `DimStart` *a* `DATETIME2`.

LA PRIMERA TRAMPA

Una BK duplicada bota la carga

`MERGE` tiene una regla estricta: cada fila del target se toca **una sola vez**. Si el origen trae el mismo productor dos veces, el motor se niega —no compara valores, cuenta cuántas veces la tocaría— y aborta toda la sentencia:

ABORTA

```
MERGE dim.Productores AS T
USING (VALUES ('P0011', 0x01), ('P0011', 0x02)) AS S (CodigoProductor,
LoadHash)
  ON T.CodigoProductor = S.CodigoProductor AND T.DimEnd IS NULL
WHEN MATCHED AND T.LoadHash <> S.LoadHash THEN UPDATE SET T.DimEnd = @Hoy;
```

Msg 8672: The MERGE statement attempted to UPDATE or DELETE the same row more than once. No es de laboratorio: reprocesos, un join aguas arriba, o una BK que al normalizar (`' P0011 '` → `'P0011'`) colapsa contra su gemela. Por eso el staging se dedupe **antes** del `MERGE`:

DEDUPE

```
... ROW_NUMBER() OVER (
      PARTITION BY UPPER(TRIM(CodigoProductor))
      ORDER BY (SELECT NULL)) AS rn
... WHERE rn = 1 AND CodigoProductor IS NOT NULL;
```

REGLA DE CAMPO

Dedupe la BK —y verificá que no sea nula— antes de todo `MERGE`. Una BK nula cae en `NOT MATCHED BY TARGET` y entra como fantasma que se re-inserta cada corrida.

LA SEGUNDA TRAMPA

BY SOURCE sobre un delta cierra toda la dimensión

`MERGE` tiene una tercera cláusula, `WHEN NOT MATCHED BY SOURCE`, que actúa sobre las filas del **target** que no aparecen en el origen. Es tentadora para «marcar las bajas». Y es correcta, pero **solo si el origen es el snapshot completo**.

Nuestro lote no lo es. Es un delta: 170 productores de los miles que tiene la dimensión. Contá cuántos cerrarías por error:

EL CONTEO

```
SELECT CerraríaPorError = COUNT(*)
FROM   dim.Productores AS T
WHERE  T.DimEnd IS NULL
  AND  NOT EXISTS (SELECT 1 FROM #Stg AS S
                   WHERE S.CodigoProductor = T.CodigoProductor);
```

Casi toda la dimensión. Un `BY SOURCE THEN UPDATE SET DimEnd = @Hoy` daría de baja a todos los que simplemente no venían en este lote —sin un solo error, con la carga reportando «éxito». A la mañana siguiente, la mitad de la cooperativa «no existe».

> REGLA DE CAMPO
>
> `BY SOURCE` *solo con snapshot completo. Con un delta, las bajas se manejan con una marca explícita que venga en el origen (una columna* `Deleted`*), nunca por ausencia.*

LA MISMA IDEA, OTRA PLATAFORMA

¿Y en Fabric?

Medio sí, medio no. `MERGE` **sí corre** en Fabric Warehouse. Lo que no viaja es justo el corazón de este truco: a junio de 2026, la cláusula `OUTPUT` **no está soportada** sobre `MERGE` en Fabric —lo probamos en un Warehouse real y la sentencia falla apenas aparece el `OUTPUT`.

Y sin `OUTPUT` no hay `$action` que capturar, así que el `MERGE` que alimenta el `INSERT` de la versión nueva no se arma igual. (Es el tipo de límite que Microsoft suele cerrar con el tiempo; si leés esto más adelante, vale la pena volver a probarlo.)

> REGLA DE CAMPO
>
> *¿La alternativa en Fabric? Los dos pasos sueltos —el `UPDATE` que cierra y el `INSERT` que abre— pero envueltos en un `BEGIN TRAN` que los vuelve atómicos. Menos elegante, y es justo el patrón que sí cruza.*

PARA RECORDAR

Lo set-based viaja entre motores; lo que depende de una extensión del lenguaje —como `OUTPUT` sobre `MERGE`— no siempre. Cuando dudes, asumí que la idea cruza y que la sintaxis fina hay que verificarla.

PARA LLEVAR

Siete cosas sobre MERGE + OUTPUT

— **SCD2 = cerrar la versión vigente y abrir una nueva, sin pisar la historia.** La card N.º 001 detecta el cambio; esta lo versiona.

— `MERGE` + `OUTPUT $action`: en una pasada cierra las que cambiaron e inserta las nuevas; el `OUTPUT` alimenta el `INSERT` que abre la nueva. La atomicidad la da el `BEGIN TRAN`.

— **Una sola sentencia no re-lee el target**, así que no expone el *non-repeatable read / phantom* de un `UPDATE` y un `INSERT` separados. Con cargas paralelas, `HOLDLOCK`.

— `$action` **es bitácora gratis:** altas, versiones nuevas, llaves tocadas, todo del mismo `OUTPUT`.

— **Dedupe la BK —y filtrá la nula— antes del** `MERGE`. Una fuente duplicada lo bota con error 8672; una BK nula siembra fantasmas.

— **Grano de día → dos cargas con cambios el mismo día dejan versiones de duración cero.** Inofensivas para point-in-time, trampa si cargás hechos entre corridas.

— `WHEN NOT MATCHED BY SOURCE` **solo con snapshot completo.** Con un delta cierra toda la dimensión calladito; las bajas, por marca explícita.

— El script completo, con la base sintética de la Cooperativa MuuSQL, está en el repo de la serie. Corre en SQL Server 2008+ con un F5; todo va en `BEGIN TRAN ... ROLLBACK`.

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 008, 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