PRIMUS DATA PRESS

CUADERNOS

Cuaderno N.º 001

SERIE · TRUCOS DE T-SQL PARA TU DW

El truco del LoadHash

Detectar cambios sin comparar treinta columnas — y las trampas que nadie cuenta, robadas a un data warehouse real.

Javier Loria

Inteligencia de Negocios

Primus Data

Inteligencia de Negocios y Analítica Avanzada

¿Cómo detecto qué filas cambiaron sin comparar columna por columna?

La pregunta aparece en toda **carga incremental** de un data warehouse. Comparar treinta columnas a mano es lento de escribir, lento de correr y fácil de equivocar: una columna que se olvida y el cambio pasa invisible hasta que, meses después, un número no cuadra.

La solución barata y conocida es un **hash por fila** con `HASHBYTES`: una huella que resume toda la fila, para comparar **una columna en vez de treinta**.

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

✦

LA SERIE

Este cuaderno abre 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 y agroindustria con más de diez años en producción; la* **Cooperativa MuuSQL** *es un escenario sintético que reproduce esos casos sin exponer al cliente. La base de demo y el script viven en el repo de la serie.*

ANTES DE LAS NOTAS

La huella digital de una fila

Pensá en cómo se identifica a una persona. Podés comparar atributo por atributo — nombre, dirección, documento, fecha de nacimiento — y rezar que ninguno venga escrito distinto. O podés hacer lo que hace cualquier sistema serio de identificación: comparar la huella digital. Una comparación, una respuesta.

Tu ETL no conoce ese truco. Cada noche, a las dos de la mañana, con treinta millones de filas de treinta columnas, compara nombre contra nombre, dirección contra dirección, columna por columna. Y le pedís que no se equivoque nunca.

> EL CASO DE ESTAS NOTAS
>
> Los ejemplos corren sobre la **Cooperativa MuuSQL**, un escenario sintético que reproduce casos reales de un EDW de retail y agroindustria sin exponer al cliente. La dimensión `dim.Productores` que verás en el código es de ahí.
>
> Léelas como conversación en voz alta, no como postura cerrada.

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

EL PROBLEMA

La comparación que nadie quiere mantener

En un data warehouse, las dimensiones tienen veinte, treinta o quinientas columnas. Cuando llega un lote nuevo del origen hay que responder algo simple: ¿qué filas cambiaron de verdad? La respuesta ingenua se ve así.

T-SQL

```
WHERE p.Nombre     <> s.Nombre
   OR p.Categoria <> s.Categoria
   OR p.Ruta      <> s.Ruta
   OR (p.Certificacion IS NULL     AND s.Certificacion IS NOT NULL)
   OR (p.Certificacion IS NOT NULL AND s.Certificacion IS NULL)
   OR p.Certificacion <> s.Certificacion
   -- ... 25 columnas más, cada una con su pareja de IS NULL
```

Tres problemas, y los tres muerden.

— **Es verboso.** Con treinta columnas y manejo de NULLs, son hasta noventa líneas de condiciones.

— **Es frágil.** Cuando el negocio agrega una columna, alguien tiene que acordarse de tocar esta comparación. Spoiler de campo: nadie se acuerda.

— **Es lento** cuando las columnas son grandes.

EL TRUCO

Una huella digital por fila

En lugar de comparar treinta columnas, calculás un hash de todas las columnas interesantes y lo guardás en una columna `LoadHash`. Es la huella digital de la fila. Cuando llegan datos nuevos, comparás huellas: treinta y dos bytes contra treinta y dos bytes. Una comparación. Única.

T-SQL

```
-- La huella nace en staging, una sola vez:
INSERT INTO stg.Productores (CodigoProductor, Nombre, ... , LoadHash)
SELECT CodigoProductor, Nombre, ... ,
       HASHBYTES('SHA2_256', CONCAT_WS('|', CodigoProductor, Nombre,
... ))
FROM Origen;

-- Al actualizar la dimensión, la huella decide (y viaja con la fila):
UPDATE p
SET    p.Nombre = s.Nombre, ... , p.LoadHash = s.LoadHash
FROM   dim.Productores p
INNER JOIN stg.Productores s ON p.CodigoProductor = s.CodigoProductor
WHERE p.LoadHash <> s.LoadHash;   -- reemplaza las 90 de arriba
```

¿Agregaron una columna? La sumás al `CONCAT_WS` y listo. El plan de ejecución compara binarios de treinta y dos bytes, no strings de quién sabe cuánto.

EL TRUCO · LETRA CHICA

Tres detalles que deciden el éxito

La huella se computa una sola vez, al cargar staging, y de ahí viaja[1]. En el UPDATE, `s.LoadHash` ya viene calculado; no lo recalculés en cada comparación.

Declará `LoadHash NOT NULL`. Si admite NULL, el `<>` devuelve UNKNOWN y esa fila queda invisible para el UPDATE — para siempre y sin ruido[2].

Sobre el algoritmo: `SHA2_256`, no `SHA1` ni `MD5` — esos dos están deprecados desde SQL Server 2016. ¿Y `SHA2_512`? Más lento y excesivo: sesenta y cuatro bytes para decidir si una fila cambió es matar moscas a cañonazos. Guardalo para validar integridad de documentos.

1 · Calcular una vez y arrastrar el resultado: el patrón más barato que conozco para no pagar CPU dos veces.

2 · Un bug que no hace ruido es el peor de todos: nadie lo busca porque nadie sabe que existe.

> HASTA ACÁ, CUALQUIER BLOG
>
> *Lo que sigue —las cuatro trampas— es lo que solo se aprende en producción.*

LAS CUATRO TRAMPAS

Trampa 1. El hash sin separador

```tsql
SELECT HASHBYTES('SHA2_256', CONCAT('AB', 'C')),   -- hashea 'ABC'
       HASHBYTES('SHA2_256', CONCAT('A', 'BC'));   -- hashea 'ABC'
también
```

Idénticos. Si concatenás sin separador, `'AB'+'C'` y `'A'+'BC'` dan la misma huella. Dos filas distintas que tu ETL jura iguales: el update nunca pasa y el dato queda viejo. Por eso `CONCAT_WS('|', …)` — el separador vuelve `'AB|C'` y `'A|BC'` huellas distintas, que es lo que son.

Trampa 2. El pipe que no separa nada

Esta corría en producción, en un EDW real.

```tsql
HASHBYTES('SHA2_256', CONCAT(A.NombreItem, '|'))
```

> REGLA DE CAMPO
>
> *Si ves un `CONCAT` que termina en pipe, o sobra el pipe o faltan columnas. Las dos son bug.*

LAS CUATRO TRAMPAS

Trampa 3 · CONCAT_WS se salta los NULL

```t-sql
SELECT CONCAT_WS('|', 'A', NULL, 'B'),   -- 'A|B'
       CONCAT_WS('|', 'A', 'B', NULL);   -- 'A|B' también
```

`CONCAT_WS` ignora los NULL en vez de dejar el hueco. Si la posición del NULL importa en tu negocio, dos filas distintas colisionan. La salida: un centinela — `ISNULL(columna, '~')` — para que el NULL deje rastro en la huella.

Y una advertencia que combina esta trampa con la del separador: un `|` en medio del contenido no molesta. Pero si un valor puede *empezar o terminar* con `|` y hay nulos o blancos alrededor, el contenido se confunde con el separador y dos filas vuelven a colisionar. Ahí consideré cambiar de separador — `~` o `^` son viables, y hay quien usa doble `||`.

> REGLA DE CAMPO
>
> *Separador y centinela de NULLs, siempre caracteres distintos.*

LAS CUATRO TRAMPAS

Trampa 4 · la collation perdona, el hash no

La más traicionera, porque el síntoma es el inverso de las otras tres: cambios falsos en vez de cambios perdidos.

```t-sql
SELECT CASE WHEN 'josé rodríguez' = 'José Rodríguez'
            THEN 'iguales' END,                  -- 'iguales'
(collation CI)
       HASHBYTES('SHA2_256', 'josé rodríguez'),     -- huella A
       HASHBYTES('SHA2_256', 'José Rodríguez');     -- huella B,
distinta
```

Para SQL Server con collation case-insensitive, esos nombres son la misma persona. Para `HASHBYTES` son bytes — y los de la minúscula no son los de la mayúscula. Lo mismo con espacios: `' P0017 '` y `'P0017'`. El origen te manda los datos con el casing cambiado y tu ETL "detecta" cientos de cambios que no existen: updates de filas idénticas, versiones SCD2 fantasma, historia inflada.

LA SOLUCIÓN COMPLETA

Normalizá antes de hashear

En los dos lados, desde el día uno. Este snippet incorpora la solución de todas las trampas anteriores a la vez: separador, centinela y normalización.

T-SQL

```
HASHBYTES('SHA2_256', CONCAT_WS('|',
    UPPER(TRIM(CodigoProductor)),
    UPPER(TRIM(Nombre)),
    UPPER(TRIM(Categoria)),
    ISNULL(Certificacion, '~'),
    RutaID))
```

El `ISNULL` va en cada columna que pueda venir NULL — en esta dimensión solo `Certificacion` lo es; en la tuya, revisá una por una.

> REGLA DE CAMPO
>
> *¿Hashes viejos sin normalizar? Recalculá la columna completa una vez, en ventana de mantenimiento. Duele menos que vivir con falsos positivos para siempre.*

LA RED DE SEGURIDAD

El índice UNIQUE filtrado

El LoadHash decide qué actualizar. Pero en una dimensión SCD2 hay una invariante mayor: solo puede existir una versión vigente por llave de negocio. Y los ETL tienen bugs.

T-SQL

```
CREATE UNIQUE NONCLUSTERED INDEX UQ_Productores_Vigentes
    ON dim.Productores (CodigoProductor)
    INCLUDE (ProductorID, LoadHash)
    WHERE DimEnd IS NULL;   -- solo vigila las vigentes
```

El filtro `WHERE DimEnd IS NULL` es la gracia: las versiones cerradas se acumulan sin límite, pero si un bug intenta insertar una segunda versión vigente del mismo productor, el INSERT falla ahí con error 2601. El bug muere en la carga, no tres meses después en un reporte que finanzas encontró antes que vos.

> REGLA DE CAMPO
>
> *Entre un bug silencioso y un error ruidoso, preferí siempre el error ruidoso.*

PARA CERRAR

Bonus · la recarga que no deja la tabla a medias

T-SQL

```
BEGIN TRAN;
    TRUNCATE TABLE etl.MiTabla;            -- transaccional

    INSERT INTO etl.MiTabla WITH (TABLOCK)  -- minimal logging
    SELECT ... FROM Origen;
COMMIT TRAN;
```

`TRUNCATE` siempre participa en transacciones (si el INSERT falla, se revierte y la tabla nunca queda vacía a media carga), aunque solo corre en tablas sin FOREIGN KEYs que las referencien. Y `WITH (TABLOCK)` habilita minimal logging cuando el modelo lo permite — heap sin índices, el caso típico de staging. Con 186 000 filas, la diferencia en log es de dos órdenes de magnitud.

Si estás en Spark, mirá `xxhash64`

El patrón viaja a cualquier plataforma; la función no tiene por qué ser la misma. `xxhash64()` es mucho más rápido que un hash criptográfico y devuelve un `long` de 64 bits. Para detectar cambios no necesitás criptografía, solo que no colisione en tu volumen. Si la huella cruza de motor, quedate con un solo algoritmo en ambos lados: un `BIGINT` de Spark no se compara con un `VARBINARY` de T-SQL.

PARA LLEVAR

Cuatro cosas que se mantienen solas

— **Una huella de treinta y dos bytes** reemplaza hasta noventa líneas de comparaciones. Y se mantiene sola.

— `CONCAT_WS('|')` **siempre**. Sin separador hay colisiones; con pipe sobrante hay bug.

— **Normalizá antes de hashear**, en ambos lados. La collation perdona, el hash no.

— **El índice UNIQUE filtrado** convierte bugs silenciosos en errores ruidosos. Preferí siempre el ruidoso.

EL SCRIPT COMPLETO

La demo —con la base sintética de la Cooperativa MuuSQL incluida— está en el repo público de la serie. Corre en cualquier SQL Server 2017+ con un F5.

— continúa en el Cuaderno N.º 002.

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 001, 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