PRIMUS DATA PRESS

CUADERNOS

Cuaderno N.º 012

SERIE · TRUCOS DE T-SQL PARA TU DW

Meta-programación

El SQL que escribe el SQL desde el catálogo: leer INFORMATION_SCHEMA y dejar que la máquina escriba lo que ibas a copiar cincuenta veces —y la línea que la separa de un agujero de inyección.

Javier Loria

Inteligencia de Negocios

Primus Data

Inteligencia de Negocios y Analítica Avanzada

META-PROGRAMACIÓN

POR QUÉ IMPORTA

¿Cómo hago lo mismo en todas las tablas sin escribirlo cincuenta veces?

La pregunta aparece apenas un data warehouse pasa de un puñado de tablas. El reflejo es copiar y pegar el mismo patrón —el hash de cambios, el `JOIN` de carga, un conteo de auditoría — tabla por tabla, y rezar para acordarse de actualizar las cincuenta cuando el esquema cambia.

No pasa: alguien agrega una columna, se olvida de un script, y el bug vive callado meses. La salida es **generar SQL desde el catálogo** —leer `INFORMATION_SCHEMA` y dejar que el SQL escriba el SQL—, lo que la industria llama *metadata-driven SQL*. El generado lee el esquema real, así que nunca se desincroniza.

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

✦

LA SERIE · CIERRE

Cuaderno que **cierra** *la serie Trucos de T-SQL para tu DW, de la charla del mismo nombre en* **Charlemos de SQL Server 2026**. *Es el más avanzado de la serie, y el que mejor resume su tesis: el código de un EDW de diez años no solo tiene buenos patrones — a veces tiene la máquina que los escribe. La* **Cooperativa MuuSQL** *reproduce un EDW real sin exponer al cliente; la base vive en el repo de la serie.*

ANTES DE LAS NOTAS

El plano cargado en la máquina, no medido a mano

Pensá en un plano de ingeniería y una máquina de corte CNC. El operario novato mide cada pieza y la corta a mano, una por una. Si el plano cambia, tiene que acordarse de volver a medirlas todas, y alguna se le va a pasar.

El que sabe carga el plano en la máquina: la CNC lee las medidas y corta las piezas solas, todas idénticas al plano, siempre. Si el plano cambia, vuelve a correr la máquina. Una sola fuente, cero piezas medidas a ojo.

LO MISMO, SIN METÁFORA

En SQL Server el plano ya existe y se llama catálogo. `INFORMATION_SCHEMA` te dice cada tabla, columna y llave de la base. Y como es data —filas que podés consultar— sirve para *generar* el SQL que ibas a escribir a mano. Meta-programar es cargar el plano en la máquina.

— **J. L.**, *y así cerramos el círculo: en el primer cuaderno vimos un buen* `LoadHash`*; en este escribimos la máquina que lo escribe.*

EL PROBLEMA

Escribirlo a mano, cincuenta veces

El `LoadHash` del primer cuaderno — `CONCAT_WS` con separador, `UPPER` / `TRIM` en los strings, la llave y las columnas de control afuera— está perfecto para una dimensión. Ahora multiplicalo por cincuenta: cincuenta expresiones casi iguales, cada una con su lista de columnas escrita a mano.

× 50 DIMENSIONES

```
-- dim.Productores ... y otras cuarenta y nueve, casi iguales:
HASHBYTES('SHA2_256', CONCAT_WS('|',
   UPPER(LTRIM(RTRIM(Nombre))), Region, Categoria /* ...y la nueva que alguien
va a olvidar */))
```

El problema no es escribirlas: es mantenerlas. El día que el negocio agrega una columna a una dimensión, alguien tiene que acordarse de sumarla al `CONCAT_WS` de esa tabla. Spoiler de campo, el mismo del primer cuaderno: nadie se acuerda. El hash queda desincronizado del esquema real, deja de ver esa columna, y el cambio pasa invisible durante meses.

> REGLA DE CAMPO
>
> *Todo SQL escrito a mano que repite la forma del catálogo se desincroniza del catálogo. Es cuestión de tiempo, no de disciplina.*

EL TRUCO

Leé el catálogo, generá el SQL

El esquema es data. Cada columna de cada tabla es una fila en `INFORMATION_SCHEMA.COLUMNS`. Si es data, la podés filtrar, ordenar y concatenar — y `STRING_AGG` arma la expresión completa:

GENERAR EL LOADHASH

```
SELECT @Hash = 'HASHBYTES(''SHA2_256'', CONCAT_WS(''|'',' +
  STRING_AGG(CONVERT(NVARCHAR(MAX),
    CASE WHEN c.DATA_TYPE LIKE '%char%'
      THEN 'UPPER(LTRIM(RTRIM(' + QUOTENAME(c.COLUMN_NAME) + ')))'
      ELSE QUOTENAME(c.COLUMN_NAME) END), ', ')
    WITHIN GROUP (ORDER BY c.ORDINAL_POSITION)) + '))'
FROM INFORMATION_SCHEMA.COLUMNS AS c
WHERE c.TABLE_SCHEMA = 'dim' AND c.TABLE_NAME = 'Productores'
  AND c.COLUMN_NAME NOT IN ('ProductorID','LoadHash','DimStart','DimEnd');
```

La regla del primer cuaderno, ahora codificada: al hash entran las columnas de negocio; quedan afuera la llave, las `IDENTITY` y las de control. Los strings se normalizan. Sale exactamente el `LoadHash` que habrías escrito a mano —pero nadie lo escribió—. Cambiás el nombre de la tabla y el generador la lee tal como está hoy. Nunca se desincroniza, porque *es* el catálogo.

EN EL EDW QUE AUDITAMOS

Un `ScriptBuilders` generaba el hash de cada dimensión leyendo `INFORMATION_SCHEMA`, y un `GeneratorJoinCondition` armaba el `JOIN ON` desde la llave real. Diez años de tablas, cero hashes escritos a mano.

EL SEGUNDO GENERADOR

El JOIN desde la llave real

El mismo principio resuelve el `ON` de un `JOIN` o un `MERGE`. Leés las columnas de la llave primaria y las encadenás:

GENERAR EL JOIN ON

```
SELECT @OnClause = STRING_AGG(CONVERT(NVARCHAR(MAX),
    't.' + QUOTENAME(k.COLUMN_NAME) + ' = s.' + QUOTENAME(k.COLUMN_NAME)),
    ' AND ') WITHIN GROUP (ORDER BY k.ORDINAL_POSITION)
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE  AS k
JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS t
  ON t.CONSTRAINT_NAME = k.CONSTRAINT_NAME
WHERE t.CONSTRAINT_TYPE = 'PRIMARY KEY' AND k.TABLE_NAME = 'Entregas';
```

Sobre una fact con llave de tres columnas sale `t.[FechaID] = s.[FechaID] AND t.[ProductorID] = s.[ProductorID] AND t.[Turno] = s.[Turno]`: la llave completa, en orden, sin olvidar una. Si mañana cambia, el generador la sigue; el `JOIN` a mano se queda viejo hasta que el `MERGE` empieza a duplicar filas. Lo generado es texto y se ejecuta con `sp_executesql` — que parametriza **valores**, no identificadores: por eso la defensa contra inyección de este patrón es `QUOTENAME`, no la parametrización.

> REGLA DE CAMPO
>
> *Si te encontrás escribiendo el mismo patrón para la tabla número tres, parate. Probablemente lo que tenés que escribir es el generador, no la tercera copia.*

LA PRIMERA TRAMPA

QUOTENAME, siempre

El catálogo es data confiable, pero los nombres no siempre son amables. Una columna con un espacio, un caracter raro o una palabra reservada — `Order`, `User`, `Group` — rompe el SQL generado si lo concatenás crudo. `QUOTENAME` envuelve cada identificador en corchetes y escapa los corchetes internos:

IDENTIFICADOR SANO

```
'SELECT * FROM ' + QUOTENAME('dim') + '.' + QUOTENAME('Order')
--  →  SELECT * FROM [dim].[Order]  : sano
```

Y hay un segundo motivo, más serio. El día que el «nombre de tabla» deje de venir del catálogo y venga de un parámetro externo, `QUOTENAME` es la línea que separa la meta-programación de un agujero de inyección SQL. Es barato y no se negocia.

> REGLA DE CAMPO
>
> *Todo identificador que entra a SQL generado va con `QUOTENAME`. No es estética: es correctitud y es seguridad.*

LA SEGUNDA TRAMPA

El orden importa, y el MAX también

`STRING_AGG` sin `WITHIN GROUP (ORDER BY ...)` concatena en un orden **arbitrario**. Para una lista de conteos da igual; para un `LoadHash` es fatal: las mismas columnas en otro orden producen otro hash, y la carga de mañana «detecta» cambios que no existen —la pesadilla del primer cuaderno— o, peor, se le escapan los que sí. Elegí un orden estable (`ORDINAL_POSITION`) y respetalo.

Y un detalle que muerde, aunque de forma honesta: si los elementos no son de tipo `MAX`, `STRING_AGG` no trunca callado — aborta con el error **9829**. Es el reverso del `CONCAT_WS` del primer cuaderno, que con el mismo desborde se quedaba corto en silencio: aquel callaba, este grita. La defensa es la misma: envolver cada elemento en `CONVERT(NVARCHAR(MAX), ...)`.

Una nota para quien todavía no tiene `STRING_AGG` (anterior a SQL Server 2017): el EDW lo armaba con CTEs recursivos —un nivel por columna— y `OPTION (MAXRECURSION 0)`. Más verboso, mismo resultado. `FOR XML PATH` es la otra vía clásica.

> REGLA DE CAMPO
>
> `WITHIN GROUP (ORDER BY ...)` *estable y* `CONVERT(NVARCHAR(MAX), ...)`*. Sin el orden, el hash miente; sin el* `MAX`*,* `STRING_AGG` *aborta con el error 9829.*

EL LÍMITE DEL PLANO

Lo que el catálogo no te dice

`INFORMATION_SCHEMA` es el catálogo portable y el punto de partida, pero es ciego a tres cosas que un generador de producción no puede ignorar — las tres envenenan un `LoadHash` si entran:

- **Columnas `IDENTITY`** (subrogadas): no son negocio, no van al hash.
- **Columnas calculadas**: derivadas de otras, y a veces no deterministas.
- **Ocultas de una temporal table** (`ValidFrom` / `ValidTo`): el motor las reescribe en cada update.

Para verlas hay que bajar a `sys.columns` — `is_computed` , `generated_always_type` —; por eso la demo usa `COLUMNPROPERTY( … , 'IsIdentity')` , que en `INFORMATION_SCHEMA` no existe. Y una verdad incómoda: el generador es tan correcto como el patrón que codifica. La máquina automatiza el patrón, no lo corrige — las defensas frágiles del primer cuaderno (el centinela del `NULL` , el `CONVERT` con estilo fijo) tiene que llevarlas el generador.

> REGLA DE CAMPO
>
> `INFORMATION_SCHEMA` *te da los nombres;* `sys.columns` *te dice cuáles excluir. Y una tabla sin llave primaria declarada hace que el generador del* `JOIN` *salga* `NULL` *en silencio: validá antes de ejecutar.*

¿Y EN FABRIC?

Sí, viaja entero

`INFORMATION_SCHEMA`, `STRING_AGG`, `QUOTENAME` y `sp_executesql` corren igual en **Fabric Warehouse**: el generador funciona tal cual, sin tocar una línea.

La salvedad es de **alcance, no de sintaxis** — el catálogo de Fabric expone menos objetos (no hay índices ni particiones que generar, y algunas DMVs de SQL Server no existen), pero el patrón de fondo —leer el catálogo, armar el SQL con `STRING_AGG`, ejecutarlo— es idéntico.

Lindo cierre de serie: el cuaderno anterior, partition switching, no viajaba a Fabric; este sí, de punta a punta. La serie cierra con un truco que funciona en los dos mundos.

> REGLA DE CAMPO
>
> *El generador es portable:* `INFORMATION_SCHEMA`, `STRING_AGG` *y* `sp_executesql` *viajan a Fabric Warehouse tal cual. Solo cambia el alcance del catálogo, no la sintaxis.*

PARA LLEVAR

Cinco cosas sobre meta-programación

— **El esquema es data.** `INFORMATION_SCHEMA` y `sys.*` son tablas consultables; `STRING_AGG` arma el SQL que ibas a escribir a mano para N tablas.

— **El generado no se desincroniza.** Un generador del `LoadHash` o del `JOIN ON` lee el catálogo real: si la tabla cambia, el SQL la sigue. El de mano se queda viejo en silencio.

— `QUOTENAME` **siempre.** En todo identificador que entra a SQL generado: por correctitud y por seguridad contra inyección.

— **El orden importa.** `WITHIN GROUP (ORDER BY ...)` estable, o el hash cambia entre corridas. Y `NVARCHAR(MAX)`, o `STRING_AGG` aborta al pasar los 8000 bytes.

— **No siempre vale.** El generado se depura como texto y, cuando falla, falla para todas las tablas a la vez. Para pocas tablas estables, las copias cuestan menos; gana cuando son muchas y cambian.

— El script completo —que genera el `LoadHash`, el predicado de `JOIN` y un script de auditoría sobre toda la Cooperativa MuuSQL— está en el repo de la serie. Corre en cualquier SQL Server 2017+ con un F5.

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 012, último 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