PRIMUS DATA PRESS

CUADERNOS

Cuaderno N.º 007

SERIE · TRUCOS DE T-SQL PARA TU DW

El truco del CROSS APPLY

Parsear un código empacado sin UDF escalar — legible, rápido y a prueba de datos sucios.

Javier Loria

Inteligencia de Negocios

Primus Data

Inteligencia de Negocios y Analítica Avanzada

Separar un campo por un delimitador en SQL sin repetir CHARINDEX ni esconderlo en una UDF.

«¿Cómo **separo un campo por un delimitador en SQL**?», «¿cómo **extraigo parte de un texto en SQL Server**?» — la pregunta aparece apenas un sistema viejo guarda varios datos en un solo campo: un código de producto que mezcla línea, modelo y color; una cuenta contable con segmentos pegados con un guión.

La respuesta de manual —repetir `SUBSTRING` y `CHARINDEX`, o esconderlos en una función— deja un código ilegible y lento, y un proceso que se cae el día que llega un dato mal formado. Este cuaderno lo parsea legible, rápido y a prueba de datos sucios, con `CROSS APPLY` en vez de UDF escalar.

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

✦

LA SERIE

Séptima 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 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

Dos datos en una sola etiqueta

Pensá en la placa de un carro: `SJB-742`. Dos datos viven en una sola etiqueta —la provincia y el consecutivo— pegados con un guión. Nadie guarda la provincia en un campo aparte; la lee del pedazo de adelante cuando la necesita.

El problema empieza cuando tenés que leerla un millón de veces y querés que cada lectura sea limpia, sin un nido de funciones repetidas.

EL CASO DE ESTAS NOTAS

En la **Cooperativa MuuSQL** pasa igual con el código de finca —`'R07-F042-T2'` — solo que ahí no hay un corte, hay **dos**, y el dato del medio es el que cuesta. Y querés liquidar litros por tanque: el **Tanque** no vive en ninguna columna, solo existe adentro del código.

Léelas como conversación en voz alta, no como postura cerrada.

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

EL PROBLEMA

Un campo, tres datos

`dim.Fincas` guarda el código empacado: `'R07-F042-T2'` es la Ruta 07, la Finca 042 y el Tanque 2.

DIM.FINCAS

```
SELECT TOP (6) FincaID, CodigoFinca, NombreFinca, RutaID
FROM dim.Fincas
ORDER BY FincaID;
```

La Ruta ya vive en su propia columna (`RutaID`), así que ahí el parseo sobra. Pero el **Tanque** no está en ningún otro lado: solo existe adentro del código. Si querés liquidar litros por tanque, no hay atajo —hay que extraerlo.

> REGLA DE CAMPO
>
> *Antes de parsear un código, fijate qué pedazo NO tiene columna propia. Ese es el único que justifica el esfuerzo; el resto ya lo tenés.*

EL REFLEJO

Repetir CHARINDEX, o esconderlo en una UDF

La Ruta es fácil —lo de antes del primer guión. El que duele es el del medio, la Finca: necesitás dónde **empieza** y dónde **termina**, y sin un nombre para esas posiciones terminás recalculándolas.

ANTI-PATRÓN

```
SELECT CodigoFinca,
  Finca = SUBSTRING(CodigoFinca,
            CHARINDEX('-', CodigoFinca) + 1,
            CHARINDEX('-', CodigoFinca, CHARINDEX('-', CodigoFinca) + 1)
                - CHARINDEX('-', CodigoFinca) - 1)
FROM dim.Fincas;
```

Corre y da el resultado correcto. También es ilegible: `CHARINDEX('-', CodigoFinca)` aparece cinco veces. El día que el delimitador cambie de `-` a `_`, te toca una cacería de reemplazos sin red.

> REGLA DE CAMPO
>
> *La UDF escalar cuesta lo que no se ve: corre fila por fila y, sin inlining (que llegó en 2019), mata el paralelismo. El piso de esta serie es 2017 — legibilidad a cambio de un plan peor.*

EL TRUCO

CROSS APPLY encadenado

`CROSS APPLY (SELECT ...)` te deja nombrar una expresión y referenciarla en el resto de la consulta —como una variable por fila— sin el costo de la UDF. Y un segundo `APPLY` ve al primero:

T-SQL

```
SELECT f.CodigoFinca,
  Ruta   = SUBSTRING(f.CodigoFinca, 1,          d1.Pos - 1),
  Finca  = SUBSTRING(f.CodigoFinca, d1.Pos + 1, d2.Pos - d1.Pos - 1),
  Tanque = SUBSTRING(f.CodigoFinca, d2.Pos + 1, 999)
FROM dim.Fincas AS f
CROSS APPLY (SELECT Pos = CHARINDEX('-', f.CodigoFinca))            AS d1
CROSS APPLY (SELECT Pos = CHARINDEX('-', f.CodigoFinca, d1.Pos + 1)) AS d2;
```

`d1.Pos` es el primer guión; `d2.Pos` arranca a buscar el segundo **desde** `d1.Pos + 1`. Cada posición se calcula una vez y se reusa. Es set-based: el optimizador lo ve como expresión, no como la caja negra de la UDF. *(El `999` del Tanque significa «de aquí al final»; cualquier número mayor al largo del campo sirve, o usá `LEN(...)`.)*

> REGLA DE CAMPO
>
> *Si vas a repetir una expresión dos o más veces en el mismo `SELECT`, sacala a un `CROSS APPLY` y nombrala. Una vez calculada, una vez escrita.*

LO QUE SE GANA

El tanque se vuelve dimensión

El parseo deja de ser cosmético: el Tanque, que no era columna de nadie, se vuelve una dimensión agrupable.

T-SQL

```
SELECT Tanque = SUBSTRING(f.CodigoFinca, d2.Pos + 1, 999),
       Litros = SUM(e.Litros)
FROM fact.Entregas   AS e
JOIN dim.Fincas      AS f ON f.FincaID = e.FincaID
CROSS APPLY (SELECT Pos = CHARINDEX('-', f.CodigoFinca))             AS d1
CROSS APPLY (SELECT Pos = CHARINDEX('-', f.CodigoFinca, d1.Pos + 1)) AS d2
GROUP BY SUBSTRING(f.CodigoFinca, d2.Pos + 1, 999);
```

En el `GROUP BY` te toca repetir la expresión —T-SQL no deja agrupar por el alias del `SELECT`— pero `d2.Pos` sigue viniendo del `APPLY`, no de un `CHARINDEX` rehecho a mano.

MISMO TRUCO, OTRO DOMINIO

En otro EDW, la dimensión de planilla traía los conceptos como `'1001-Salario Base'` —código y descripción pegados— y los partían con un `CROSS APPLY (SELECT CHARINDEX('-', ...) AS DashI)`, reusando `DashI` para el `LEFT` y el `SUBSTRING`.

LA PRIMERA TRAMPA

El código sin guión tumba la consulta

`CHARINDEX` no falla cuando no encuentra el delimitador: devuelve `0`. Ese `0` se cuela en los cálculos de longitud y los vuelve negativos. Basta un código sucio:

CRASH

```
SELECT CodigoFinca,
  Ruta = LEFT(CodigoFinca, CHARINDEX('-', CodigoFinca) - 1)
FROM (VALUES ('R07-F042-T2'), ('SINFORMATO')) AS s(CodigoFinca);
```

`'SINFORMATO'` no tiene guión, `CHARINDEX` da `0`, y `LEFT(.., -1)` responde *«Invalid length parameter passed to the LEFT function»* y tumba **la sentencia entera** —no la fila mala. Una liquidación de tres millones de filas se cae por una. El blindaje vuelve ese `0` en algo seguro antes de restarle:

FIX

```
Ruta = LEFT(s.CodigoFinca,
        COALESCE(NULLIF(d1.Pos, 0) - 1, LEN(s.CodigoFinca)))
```

Y ojo con el segundo guión: si falta (`'R07-F042'`), el Tanque devuelve el código entero **sin error** —un tanque fantasma que suma litros donde no debe. El crash al menos avisa; este no.

> REGLA DE CAMPO
>
> `CHARINDEX` *que no encuentra devuelve* `0`*, no error. Blindá con* `NULLIF(pos, 0)` *cada posición —el* `0` *silencioso duele más que el que truena.*

LA SEGUNDA TRAMPA

STRING_SPLIT no es parsing posicional

«¿Y por qué no `STRING_SPLIT` y agarro el segundo pedazo?» Porque `STRING_SPLIT` devuelve un **conjunto**, no una lista ordenada:

SET, NO LISTA

```
SELECT value FROM STRING_SPLIT('R07-F042-T2', '-');
```

Salen tres filas — `R07`, `F042`, `T2` — pero el orden no está garantizado y no hay columna que diga cuál es cuál. No existe «el segundo token». Y el delimitador debe ser de **un solo carácter**: si es `'::'` o `'⟶'`, no aplica; `CHARINDEX` no tiene ese límite. SQL Server 2022 agregó el ordinal:

2022+

```
SELECT value, ordinal
FROM STRING_SPLIT('R07-F042-T2', '-', 1) ORDER BY ordinal;
```

Con `ordinal = 2` ya tenés la Finca. Pero en 2019 o antes —o si querés las tres partes como columnas, no como filas— el `CROSS APPLY` sigue ganando: posicional, en una pasada, todo en columnas.

LA MISMA IDEA, OTRA PLATAFORMA

¿Y en Fabric?

El truco viaja completo: `CROSS APPLY`, `CHARINDEX` y `SUBSTRING` corren idéntico en Warehouse y en el SQL endpoint. `STRING_SPLIT` también existe; el tercer argumento ordinal depende de la versión, igual que en SQL Server local.

La regla de fondo —parsear posicional con `APPLY` en vez de UDF escalar— no cambia de plataforma. El patrón no es de un motor: es de cómo pensás el parseo.

> REGLA DE CAMPO
>
> `APPLY` *posicional viaja entre motores; la UDF escalar se lleva sus costos a donde la pongas. Lo portable es la forma de pensar, no la sintaxis.*

PARA RECORDAR

Cuando dudes entre SQL Server local y Fabric, asumí que lo set-based viaja y lo fila-por-fila no. El `CROSS APPLY` encadenado es lo primero; la UDF escalar, lo segundo.

PARA LLEVAR

Cinco cosas sobre el CROSS APPLY

— `CROSS APPLY (SELECT expr)` **le pone nombre a un cálculo intermedio sin UDF escalar.** Lo calculás una vez, lo reusás en todo el `SELECT`, y el optimizador lo entiende.

— `APPLY` **encadenado para multi-delimitador:** el segundo `APPLY` ve al primero. Encontrá el guión 1, luego el 2 desde ahí.

— **La UDF escalar cuesta lo que no se ve:** corre fila por fila y, sin inlining, mata el paralelismo.

— `CHARINDEX` **devuelve** `0`**, no error, cuando no encuentra.** Ese `0` vuelve negativas las longitudes y tumba la sentencia entera. Blindá **cada** posición con `NULLIF` / `COALESCE`, no solo la primera.

— `STRING_SPLIT` **es para sets, no para campos posicionales.** El ordinal (2022+) ayuda; antes, o para columnas, `CROSS APPLY` gana.

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 (la sección del ordinal de `STRING_SPLIT` pide 2022).

— *sigue en la serie* **Trucos de T-SQL para tu DW**.

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 007, 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