PRIMUS DATA PRESS

CUADERNOS

*Cuaderno N.º 010*

SERIE · TRUCOS DE T-SQL PARA TU DW

# Temporal Tables

*El point-in-time que el motor mantiene por vos: cómo se veía cualquier dato en cualquier fecha, en una línea —y las trampas que vuelven la respuesta sutilmente equivocada.*

**Javier Loria**

*Inteligencia de Negocios*

Primus Data
Inteligencia de Negocios y Analítica Avanzada

# ¿Cómo estaba este cliente — o este precio— el 1 de marzo?

Es una pregunta que tarde o temprano llega de auditoría, de un reclamo o de un reporte que tiene que cuadrar contra una foto vieja. Si tu tabla solo guarda el estado actual, la respuesta es «no sé»: el dato de ayer se sobrescribió.

Construir ese **historial de cambios** a mano —columnas de vigencia, lógica de cierre, un `WHERE` con rango— funciona, pero es código que alguien mantiene y que falla callado. Las **temporal tables (system-versioned)** lo hacen por vos: «¿cómo se veía este dato en tal fecha?» se responde en una línea.

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

✦

LA SERIE

*Décima entrega de la serie Trucos de T-SQL para tu DW, de la charla del mismo nombre en* **Charlemos de SQL Server 2026**. *Es la otra cara del cuaderno del SCD2 manual: el mismo problema de «¿cómo se veía esto en tal fecha?», resuelto por el motor en vez de por tu ETL. 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 Registro que recuerda por vos

Pensá en el Registro de la Propiedad. Si querés saber quién era dueño de un lote el 1 de marzo de 2024, no le pedís al dueño actual que recuerde; le preguntás al Registro, y el Registro te contesta desde su historia sellada. Cada traspaso quedó anotado con su fecha, y nadie tuvo que llevar un cuaderno aparte.

El sistema guarda la versión vieja sin que nadie se lo pida, cada vez que algo cambia. Tu dimensión SCD2 es ese mismo Registro, pero llevado a mano: cada cambio, alguien tiene que acordarse de cerrar la fila vieja con su `DimEnd`, abrir la nueva con su `DimStart`, y escribir el `WHERE` con el rango correcto cada vez que alguien pregunta por el pasado.

> LO MISMO, SIN METÁFORA
>
> Las temporal tables son el Registro: el motor lleva la historia — guarda cada versión vieja, sellada con su fecha— y vos solo preguntás. Sin cuaderno aparte, sin lógica de cierre de versiones que mantener.

— **J. L.**, *preguntándole al motor cómo se veía una fila el 1 de marzo.*

EL PROBLEMA

# A mano se escribe bien… casi siempre

Responder «¿cómo estaba este productor el 1 de marzo de 2024?» sobre un SCD2 manual se ve así: un `WHERE` contra el rango de vigencia. Funciona. Pero esconde tres acuerdos que tenés que respetar en todos lados, todo el tiempo.

EL POINT-IN-TIME A MANO

```
-- ¿Cómo estaba esta fila en @Cuando? Sobre un SCD2:
WHERE @Cuando ≥ p.DimStart
  AND (@Cuando < p.DimEnd OR p.DimEnd IS NULL)
```

Que vos mantenés `DimStart` / `DimEnd` en cada carga. Que el extremo es semi-abierto ( `<` , no `≤` ). Que la versión abierta lleva `DimEnd NULL` . Si alguien en otro reporte escribe `≤` en vez de `<` , o se olvida del `OR DimEnd IS NULL` , el resultado no truena: devuelve la versión equivocada con cara de estar bien. Y eso lo descubrís cuando un número histórico no cuadra, meses después.

REGLA DE CAMPO

*Un point-in-time a mano no falla con un error: falla devolviendo la versión equivocada con cara de correcta. Y cada reporte que pregunta por el pasado tiene que escribir el mismo `WHERE` , igual de bien.*

EL TRUCO

# El point-in-time en una línea

Una temporal table es una tabla normal —con su **llave primaria**, que es requisito— más dos columnas de período (`ValidFrom` / `ValidTo`) y una tabla de historia enganchada por `SYSTEM_VERSIONING`. La actual guarda solo lo vigente; la de historia, todo lo que fue. El motor mantiene las dos.

LA PREGUNTA DEL MILLÓN

```
SELECT Categoria, RutaID
FROM   dim.ProductoresTemporal
       FOR SYSTEM_TIME AS OF '2024-03-01'
WHERE  CodigoProductor = 'P0006';
```

Sin `DimEnd`, sin NULLs, sin rango. El motor busca en actual + historia la fila cuyo período contiene ese instante. Un detalle de sintaxis que agarra rápido: `AS OF` solo acepta un **literal o una variable**, nunca una expresión ni una función. `AS OF DATEADD(SECOND, -1, @Borde)` no compila; calculá el instante en una variable y pasá la variable.

> REGLA DE CAMPO
>
> `FOR SYSTEM_TIME AS OF` *responde «¿cómo se veía esto en tal fecha?» en una línea, contra actual + historia. El argumento es un literal o una variable —nunca una expresión—: calculá el instante aparte.*

CINCO FORMAS

# AS OF y cuatro maneras más de mirar el tiempo

`AS OF` es la más usada, pero no la única. Hay cinco subcláusulas de `FOR SYSTEM_TIME`, y la diferencia entre dos de ellas es solo un borde:

LAS CINCO FORMAS

```
FOR SYSTEM_TIME
  AS OF @t                 -- la foto exacta en @t
  FROM @a TO @b            -- activas en [@a, @b)  excl.
  BETWEEN @a AND @b       -- activas en [@a, @b]  incl.
  CONTAINED IN (@a, @b)    -- vigencia entera en [@a, @b]
  ALL                      -- todo: actual + historia
```

`FROM/TO` y `BETWEEN/AND` se diferencian solo en el borde superior: el primero lo **excluye**, el segundo lo **incluye**. Una versión que arranca justo en el límite aparece en una y se cae de la otra.

> REGLA DE CAMPO
>
> `FROM..TO` *excluye el borde superior,* `BETWEEN..AND` *lo incluye. Una versión que arranca en el límite aparece en una y se cae de la otra.*

LO MÁS SUBESTIMADO

# El versionado lo hace el motor, no tu ETL

Acá está el verdadero ahorro. En el SCD2 manual, aplicar un cambio es el ritual del `MERGE`: cerrar la versión vigente, abrir la nueva, todo atómico. En una temporal table, eso lo hace el motor en cada `UPDATE`:

UN UPDATE CUALQUIERA

```
UPDATE dim.ProductoresTemporal
SET    Categoria = 'A'
WHERE  CodigoProductor = 'P0006';
-- el motor cierra la versión vigente y abre la nueva
```

Ese `UPDATE` —uno cualquiera— le estampa el `ValidTo` a la versión vigente, abre una nueva y manda la vieja a historia. Tu código no escribe una sola línea de versionado. El `DELETE` tampoco borra de verdad: archiva la fila, que sigue consultable con `AS OF`. Un apunte que muerde en las recargas: el motor versiona por **fila tocada**, no por valor cambiado —un `UPDATE` que no altera ningún dato igual genera versión nueva e infla la historia.

> REGLA DE CAMPO
>
> *Si tu ETL tiene código dedicado a «cerrar la versión vieja y abrir la nueva», una temporal table te borra ese código de versionado entero. (La lógica de transformación y de llaves sigue siendo tuya.)*

LA PRIMERA TRAMPA

# Los períodos viven en UTC

Esta sorprende a todo el mundo la primera vez. `ValidFrom` / `ValidTo` no usan la hora local del servidor: usan `SYSUTCDATETIME()`. Y el argumento de `AS OF` se interpreta **siempre como UTC**.

CONVERTÍ ANTES DE PREGUNTAR

```
-- AS OF interpreta su argumento como UTC, no como hora local:
... FOR SYSTEM_TIME AS OF @InstanteLocal ...   -- ±6 h

-- La salida: convertir a UTC (respeta el horario de verano)
SET @EnUTC = @InstanteLocal
    AT TIME ZONE 'Central America Standard Time'
    AT TIME ZONE 'UTC';
```

En Costa Rica (UTC−6), si guardás «cambió a las 18:00» y consultás `AS OF '18:00'`, podés recibir la versión vieja: el período real arrancó a las 00:00 UTC del día siguiente. No hay error; hay seis horas de corrimiento silencioso. El «+6h» a mano solo vale donde no hay horario de verano; con DST se equivoca dos veces al año —por eso `AT TIME ZONE`.

> REGLA DE CAMPO
>
> *El argumento de `AS OF` es UTC. Si le pasás hora local cruda, preguntás por un instante corrido seis horas.*

LA SEGUNDA TRAMPA

# Semiabierto, y la historia hay que pedirla

El `ValidTo` de una versión es exactamente el `ValidFrom` de la siguiente, y ese instante pertenece a la **nueva**. El período es `[desde, hasta)` —semiabierto—, la misma regla que las vigencias con `LEAD` del cuaderno del DW. `AS OF` justo en el borde devuelve la que arranca ahí, nunca la que cerró en ese instante.

El motor lo respeta por su cuenta; el riesgo aparece cuando comparás contra esas fechas en tu propio `WHERE`. Y la trampa que cierra el cuaderno es la más fácil de pasar por alto: **un `SELECT` normal no ve la historia.** Sin `FOR SYSTEM_TIME`, devuelve solo lo vigente.

### EL REPORTE QUE MIENTE SIN AVISAR

Un reporte que se anuncia «histórico» y se olvida del `FOR SYSTEM_TIME` no falla: devuelve el presente disfrazado de pasado. No hay error —hay una foto del hoy con etiqueta de ayer.

> REGLA DE CAMPO
>
> *Sin `FOR SYSTEM_TIME`, solo ves lo vigente. Un reporte histórico sin esa cláusula miente sin avisar.*

LO QUE EL MOTOR NO HACE

# La historia no se purga ni se indexa sola

El motor te regala la historia, pero no su administración. Tres cosas para tener en el radar:

- **La historia crece sin techo.** Cada cambio acumula una fila para siempre, salvo que le pongas `HISTORY_RETENTION_PERIOD`. En una dimensión que cambia seguido, esa tabla engorda sola: definí retención si el volumen importa.
- **La tabla de historia quiere su propia indexación.** El point-in-time es fácil de escribir, no siempre barato de correr: un `AS OF` sobre una historia enorme paga el recorrido. Para historia analítica grande, un índice **columnstore** le cambia la vida.
- **Cambiar columnas no exige apagar el versionado.** Un `ALTER` de columna se propaga solo a la historia. La letra menuda: `ONLINE = ON` no surte efecto en una temporal, así que ese `ALTER` **bloquea la tabla** mientras corre —planificá la ventana.

> REGLA DE CAMPO
>
> *La historia no se purga ni se indexa sola.* `HISTORY_RETENTION_PERIOD` *para que no crezca sin fin, y un índice pensado para cómo la vas a consultar.*

¿Y EN FABRIC?

# Depende de cuál Fabric

**Fabric SQL Database** —la base operacional— soporta temporal tables igual que SQL Server: `SYSTEM_VERSIONING` y `FOR SYSTEM_TIME` viajan completos. **Fabric Warehouse** y el **SQL analytics endpoint** del Lakehouse no las tienen, pero sí su propia respuesta al point-in-time: **Time Travel**.

TIME TRAVEL · WAREHOUSE

```
SELECT * FROM dbo.Productores
OPTION (FOR TIMESTAMP AS OF '2024-03-01T00:00:00');
```

No versiona fila por fila —consulta el almacén entero tal como estaba en un instante—, pero responde la misma pregunta: afecta toda la consulta (joins incluidos) y es de solo lectura. El Warehouse retiene hasta 30 días por defecto (1 a 120); en el Lakehouse la ventana la manda el `VACUUM` de Delta. Y comparte la trampa de este cuaderno: el instante se interpreta **en UTC**.

> REGLA DE CAMPO
>
> *El truco vive en SQL Server (2016+), Azure SQL DB y Managed Instance. En Fabric Warehouse/Lakehouse no hay temporal tables, pero Time Travel responde lo mismo —y Microsoft lo mueve seguido: reverificá.*

PARA LLEVAR

# Cuatro cosas sobre temporal tables

— **El motor mantiene la historia sola.** Cada `UPDATE` cierra la versión vigente y abre una nueva, sin código de versionado en tu ETL.

— **Point-in-time en una línea:** `FOR SYSTEM_TIME AS OF`. Y cuatro subcláusulas más para rangos — `FROM..TO` excluye el borde superior, `BETWEEN..AND` lo incluye.

— **Los períodos viven en UTC** y usan el tiempo de inicio de transacción. El argumento de `AS OF` es UTC: convertí tu hora local antes de preguntar.

— **Sin `FOR SYSTEM_TIME` solo ves lo vigente.** La historia hay que pedirla; el reporte que la olvida muestra el presente y lo hace pasar por historia.

*— El script completo —con la temporal table y su historia retro-fechada en la base de la Cooperativa MuuSQL— está en el repo de la serie. Corre en cualquier SQL Server 2016+ 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 010, 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*