SQL Específico de SQL Server

Antonio Espín Herranz

SQL (no estándar)

- SQL Server permite UPDATE con join:

```
UPDATE t
SET t.Campo = o.OtroCampo
FROM Tabla t
JOIN OtraTabla o ON t.Id = o.Id;
```

Ejemplos

```
UPDATE p
SET p.Precio = t.PrecioNuevo
FROM Productos p
JOIN Tarifas t ON p.IdTarifa = t.IdTarifa;

UPDATE c
SET c.Activo = 0
FROM Clientes c
LEFT JOIN Compras co ON c.IdCliente = co.IdCliente
WHERE co.IdCliente IS NULL;

UPDATE f
SET f.Total = SUM(d.Cantidad * d.Precio)
FROM Facturas f
JOIN Detalles d ON f.IdFactura = d.IdFactura
JOIN Clientes c ON f.IdCliente = c.IdCliente
WHERE c.Categoria = 'VIP';
```

SQL (no estándar)

- SQL permite DELETE con join:

```
DELETE t
FROM Tabla t
JOIN OtraTabla o ON t.Id = o.Id;
```

Ejemplos

```
DELETE p
FROM Pedidos p
LEFT JOIN Clientes c ON p.IdCliente = c.IdCliente
WHERE c.IdCliente IS NULL;

DELETE d
FROM Detalles d
JOIN Facturas f ON d.IdFactura = f.IdFactura
JOIN Clientes c ON f.IdCliente = c.IdCliente
WHERE c.Categoria = 'Baja';
```

Tener en cuenta

- Solo puedes borrar o actualizar UNA tabla (la que pones después de UPDATE o DELETE)
- Pero puedes usar todas las tablas que quieras en el FROM
- La sintaxis es idéntica a un SELECT con JOIN

```
UPDATE a
SET a.Stock = a.Stock - d.Cantidad
FROM Almacen a
JOIN Detalles d ON a.IdProducto = d.IdProducto
JOIN Facturas f ON d.IdFactura = f.IdFactura
JOIN Clientes c ON f.IdCliente = c.IdCliente
WHERE c.Categoria = 'Mayorista';
```

Tener en cuenta

- Es mas eficiente **JOIN** que una **subconsulta** (más lenta con varias tablas).

```
UPDATE Productos
SET Precio = (
   SELECT PrecioNuevo
   FROM Tarifas
   WHERE Tarifas.IdTarifa = Productos.IdTarifa
)
WHERE IdCategoria IN (
   SELECT IdCategoria
   FROM Categorias
   WHERE Activa = 1
);
```

```
UPDATE p
SET p.Precio = t.PrecioNuevo
FROM Productos p
JOIN Tarifas t ON p.IdTarifa = t.IdTarifa
JOIN Categorias c ON p.IdCategoria = c.IdCategoria
WHERE c.Activa = 1;
```

Tener en cuenta

- ¿Cuándo usar subconsultas?
  - Cuando necesitas un único valor escalar
  - Cuando la lógica es más clara así
  - Cuando no quieres que el JOIN duplique filas

- ¿Cuándo NO usar subconsultas?
  - Cuando hay más de una tabla relacionada
  - Cuando necesitas filtrar por varias condiciones
  - Cuando la subconsulta devuelve muchas filas
  - Cuando la lógica es equivalente a un JOIN

SQL (no estándar)

- MERGE con extensiones propias:

```
MERGE INTO Destino AS d
USING Origen AS o
ON d.Id = o.Id
WHEN MATCHED THEN UPDATE SET …
WHEN NOT MATCHED THEN INSERT (…);
```

Ejemplo

Para sincronización masiva → Sí, es más eficiente

Para operaciones simples → JOIN es más rápido y más claro

Para lógica compleja → MERGE puede ser más difícil de leer

- Por ejemplo: con una tabla nueva de productos:

```
MERGE INTO Productos p
USING ProductosNuevos n
ON p.idProducto = n.idProducto

WHEN MATCHED THEN
    UPDATE SET
        p.nombre  = n.nombre
        p.precio = n.precio

WHEN NOT MATCHED BY TARGET THEN
    INSERT (IdProducto, nombre, precio)
    VALUES (n.idProducto, n.nombre, n.precio)

WHEN NOT MATCHES BY SOURCE THEN
    UPDATE SET p.Activo=0;
```

SQL (no estándar)

- Permite TOP en SELECT, UPDATE Y DELETE:
  - SELECT TOP 10 * FROM Clientes;
    - Limitar resultados
  - Se puede usar con porcentaje:
    - SELECT TOP 5 PERCENT * FROM VENTAS ORDER BY IMPORTE DESC;
- DELETE TOP (100) FROM Logs;
  - **¿Para qué sirve?**
  - Borrar datos por lotes
  - Evitar bloqueos masivos
  - Limpiar logs antiguos sin parar el sistema
  - Controlar el impacto en índices y transacciones

SQL (no estándar)

- TOP en UPDATE
  - ¿Para qué sirve?
  - Actualizar datos por lotes (batching)
  - Evitar bloqueos grandes
  - Procesar millones de filas en partes
  - Controlar el impacto en producción

- Ejemplo realista: actualizar en lotes de 1000
  - UPDATE TOP (1000) Pedidos
  - SET Estado = 'Procesado'
  - WHERE Estado = 'Pendiente';

Utilidad TOP en UPDATE / DELETE

- Porque SQL Server:
  - Evita bloqueos largos
  - Reduce el tamaño de transacciones
  - Minimiza el impacto en índices
  - Permite procesar millones de filas en **batches controlados**
  - Evita timeouts en producción
  - En sistemas grandes, es **imprescindible**.

Sentencia	¿Para qué sirve TOP?	Ejemplo
SELECT	Limitar resultados	TOP 10
UPDATE	Actualizar por lotes	UPDATE TOP (1000)
DELETE	Borrar por lotes	DELETE TOP (5000)

SQL (no estándar)

- Permite OUTPUT en INSERT, UPDATE Y DELETE:
  - Sirve para saber **qué filas se insertaron** y obtener valores generados automáticamente (IDENTITY, DEFAULT,

  - **Inserta y devuelve el id generado:**

```
INSERT INTO Clientes (Nombre, Ciudad)
OUTPUT inserted.IdCliente
VALUES ('Antonio', 'Madrid');
```

  - **Actualizar y devuelve id insertado y el saldo:**

```
UPDATE Clientes
SET Saldo = Saldo + 100
OUTPUT inserted.Id, inserted.Saldo;
```

SQL (no estándar)

- Ejemplo: registrar en una tabla auditoria:

```
INSERT INTO Clientes (Nombre, Ciudad)
OUTPUT inserted.IdCliente, inserted.Nombre, GETDATE()
INTO AuditoriaClientes (IdCliente, Nombre, Fecha)
VALUES ('Ana', 'Sevilla’);
```

- Registrar cambios en los precios:

```
UPDATE Productos
SET Precio = Precio * 1.10
OUTPUT deleted.Precio AS PrecioAnterior,
    inserted.Precio AS PrecioNuevo,
    inserted.IdProducto
INTO AuditoriaPrecios (PrecioAnterior, PrecioNuevo, IdProducto);
```

deleted = valores antes del UPDATE

inserted = valores después del UPDATE

Ejemplo: join con output

- UPDATE p
- SET p.Stock = p.Stock - d.Cantidad
- OUTPUT inserted.IdProducto, deleted.Stock AS StockAntes, inserted.Stock AS StockDespues
- FROM Productos p
- JOIN Detalles d ON p.IdProducto = d.IdProducto
- WHERE d.Procesado = 0;

Ejemplo: output con merge

```
MERGE INTO Productos AS p
USING ProductosNuevos AS n
ON p.IdProducto = n.IdProducto

WHEN MATCHED THEN
   UPDATE SET p.Precio = n.Precio

WHEN NOT MATCHED THEN
   INSERT (IdProducto, Nombre, Precio)
   VALUES (n.IdProducto, n.Nombre, n.Precio)

OUTPUT $action AS Accion,
     inserted.*,
     deleted.*;
```

Para auditoria simple
Mas eficiente que los Triggers

SQL (no estándar)

- SQL Server permite **funciones escalares y TVF** dentro de SELECT:

```
SELECT dbo.MiFuncion(Valor)
FROM Tabla;
```

- **TVF**: Table-Valued Function es una función que devuelve una tabla:

```
CREATE FUNCTION dbo.PedidosPorCliente (@IdCliente INT)
RETURNS TABLE
AS
RETURN (
   SELECT IdPedido, Fecha, Total
   FROM Pedidos
   WHERE IdCliente = @IdCliente
);
```

SQL (no estándar)

```
CREATE FUNCTION dbo.CalcularIVA (@Precio DECIMAL(10,2),
@PorcentajeIVA DECIMAL(5,2))
RETURNS DECIMAL(10,2)
AS
BEGIN
   RETURN @Precio * (@PorcentajeIVA / 100.0);
END;
```

- SELECT dbo.CalcularIVA(100, 21) AS IVA;
- **Con muchos datos, cálculos complejos puede ralentizar!**

SQL (no estándar)

Id	PadreId	Nombre
1	NULL	Ropa
2	1	Hombre
3	1	Mujer
4	2	Camisas
5	2	Pantalones
6	4	Manga larga

- SQL Server permite **CTE recursivas**

```
WITH cte AS (
   SELECT Id, PadreId, nombre
   FROM Categorias
   WHERE PadreId IS NULL        -- 1) Categorías raíz

   UNION ALL

   SELECT c.Id, c.PadreId, nombre
   FROM Categorias c
   JOIN cte ON c.PadreId = cte.Id -- 2) Hijos de cada categoría encontrada
)
SELECT * FROM cte;
```

1 (Ropa)
2 (Hombre)
4 (Camisas)
6 (Manga larga)
5 (Pantalones)
3 (Mujer)

SQL (no estándar)

- Permite TRY/CATCH en SQL:

```
BEGIN TRY
   SELECT 1/0;
END TRY
BEGIN CATCH
   PRINT ERROR_MESSAGE();
END CATCH;
```

Ejemplo

```
BEGIN TRY
   SELECT CAST('ABC' AS INT);  -- Error de conversión
END TRY
BEGIN CATCH
   SELECT
      ERROR_NUMBER()   AS NumeroError,
      ERROR_MESSAGE()  AS Mensaje,
      ERROR_LINE()     AS Linea,
      ERROR_PROCEDURE() AS Procedimiento;
END CATCH;
```