



SQL Server Tips and Tricks for Access Developers

Access Europe

June 3rd 2026

Kevin Bell



What we are going to cover

- 5 or 6 SQL Server tips that you may, or may not know about
- Originally presented at Armen's Access Day in 2025
 - Also presented at the UK AUG, DevCon and at Evento Internacional de Datos in Valencia Spain
- I have used them all in the last 2-3 years to solve a real-world problems at a clients on real systems
 - I'm not necessarily an expert with any or all of them
- Generally, these are tips to make SQL Server run faster, or be more readable, or be more manageable or all three 😊
 - (None are solely related to Access)
 - Other relational databases may have similar functionality

Who is Kevin?

- “Developing” with Access since version 1.0
- Started using SQL Server with a beta version of Access 2.0/SQL 4.2
- Ran a small consulting firm in Colorado for 15+ years
- Joined the Microsoft Access Team as a test engineer in 2008
- Returned to consulting
 - SQL Server, .Net/#C, maybe 10% Access front-ends
- Always leaning – building a financial planning application
 - Tauri App, cross platform (Rust, React, Java, Next.JS, TypeScript, SQLite)
- Enjoy cycling and traveling the world looking for the perfect pint of ale

Temp Tables (SQL Server 7.0 1998)

- Temp Tables
 - `CREATE TABLE #Employee`
 (`EmployeeID INT NOT NULL PRIMARY KEY,`
 `FullName NVARCHAR(32) NOT NULL`);
- Table Variables (SQL Server 2000)
 - `DECLARE @Employee AS TABLE`
 (`EmployeeID INT NOT NULL PRIMARY KEY,`
 `FullName NVARCHAR(32) NOT NULL`);
 - (Saved in the tempdb unless memory-optimized)

Temp Tables

- Demo

Temp Tables – Case Study

- Client had a nightly Sarbanes-Oxley (SOX) audit report that documented the prior days edits, compared record changes
- >95% of the time there were little to no changes = few seconds
- <5% when there were changes runtime 5 mins to 3+ hours
- The client didn't care about the runtime, only that it ran successfully
- Engineering group migrated the report from SSIS to Azure Data Factory... which apparently has a 3-hour time limit
 - I ran the full process locally and it took 4h15min

Temp Tables – The Issue

- Audit report would query the prior days updates the find the $n-1$, $n+1$ record to show the previous / next state
- The audit table had grown from 2M records to more than 65M records
 - With just a few audit, the performance was acceptable
 - More than ~1,000 would timeout

Temp Tables – The Solution

- Take the list of changed records, the put all the related audit records into a temp table
 - Real world failure had ~1,100 changes which was ~24,000 total audit records
- Replace the existing table with the temp table
 - Given this was an audit report and significant change would have to go through a complete QA cycle
- Results?
 - Less than 2 seconds

Common Table Expressions { CTE } [SQL 2005]

- `WITH myCTE
AS (
 SELECT *
 FROM tableName
 WHERE someCriteria
)
SELECT *
FROM myCTE;`

Common Table Expressions

- `WITH myCTE
AS (
 SELECT *
 FROM tableName
 WHERE someCriteria
)`
`,myOtherCTE AS (
 SELECT *
 FROM myCTE
)`
`SELECT *
FROM myOtherCTE;`

CTE Recursive

- ```
WITH DateSequence AS (
 SELECT CAST('2026-04-01' AS DATE) AS DateValue -- starting date
 UNION ALL
 SELECT DATEADD(DAY, 1, DateValue)
 FROM DateSequence
 WHERE DATEADD(DAY, 1, DateValue) <= '2026-03-31' -- ending date
)
SELECT DateValue
FROM DateSequence
OPTION (MAXRECURSION 1000); -- default recursion is 100 levels
```

# Common Table Expressions

- Demo

# Common Table Expressions AI Bonus!

- CTE are often the better solution than Temp Tables
- Temp Tables are easier to debug
- Use AI to convert your CTE to Temp Tables
  - Ask it to add commented out `-- SELECT * FROM #tbl;` statements
- Then after editing/debugging use AI to convert the Temp Tables back to a CTE
- There may be a significant performance difference

# Table Variable vs Temp Table vs CTE

|             | Table Variable            | Temp Table               | CTE                   |
|-------------|---------------------------|--------------------------|-----------------------|
| Scope       | Batch or Stored Proc      | Session/Global           | Single Statement      |
| Indexing    | Limited                   | Yes                      | No                    |
| Constraints | Limited                   | Yes                      | No                    |
| DDL         | No                        | Yes                      | No                    |
| Performance | "Good" for small datasets | Good for larger datasets | Optimized for queries |

Table Variables can be defined with a User-Defined Table Type, can be used in UDF and generally do not trigger recompiles.

# Window Functions (SQL 2005, 2012, 2016)

- `WINDOW_FUNCTION(something) OVER (PARTITION BY field1, field2 ORDER BY field, field2) AS alias`



# Window Functions

- Aggregate Functions

- avg()
- count()
- max()
- min()
- sum()

- Ranking Functions

- row\_number()
- rank()
- dense\_rank()

- Analytic Functions

- lead()
- lag()
- ntile()
- first\_value()
- last\_value()
- nth\_value()

- Distribution Functions

- percent\_rank()
  - cume\_dist()
- 

# Windowing Functions

- Demo

# Window Functions – Case Study

- Client was looking to make a major business rule change that would require 750,000 items to be recalculated (~14M records)
- There was an existing process that used a cursor
  - This was fine as when records needed to be recalculated one off due to an event change
- Estimated the current cursor could process 20 items per second
  - More than 10 hours

# Window Functions – Solution

- Used ROW\_NUMBER and LAG/LEAD window functions to process the bulk of those 750k items
- Run the updated cursor logic on the ~20k exceptions
- Results: 10-12\* minutes to bulk process, plus 10-15 minutes for the exceptions
  - \* on a development server

# **sp\_recompile *object* [SQL Server 6.5 1996]**

- Drops the existing execution plan for an object and forces a recompile... in mid run
  - sp\_recompile N'*objectName*'
- Client had a quarterly process that once a year was significantly longer
  - More data to process lead to ~6.5 hours execution time
  - When the multiple steps were run individually, it was performant

# **sp\_recompile *object***

- Step 0 (Compiles and creates execution plan)
- Step 1
- Step 2
- Step 3
- Step 4 (0 records in a staging table)
- Step 5 (now 16M records in the staging table)
- Step 6
- Step n (records deleted from staging table) Runtime: 6+ hours

# **sp\_recompile *object***

- Step 0 (Compiles and creates execution plan)
- Step 1
- Step 2
- Step 3
- Step 4 (0 records in a staging table)
  - Step 4A sp\_recompile N'*stagingTable*'
- Step 5 ( now 16M records in the staging table)
- Step 6
- Step n (records deleted from staging table) Runtime : 27 minutes

# Table Value Function {TVF} [SQL 2000]

- Is a user-defined function that returns a table
  - I like to think of it as a parametrized view
- Inline Syntax
  - ```
CREATE OR ALTER FUNCTION myTVF(@ID INT)
RETURNS TABLE AS
RETURN (
    SELECT ID,
        ...
    FROM myTable
    WHERE ID=@ID
);
```



```

strSql = "SELECT      ent.LegalName, el.FundAccountDebtorExposureLimit / FX.FxRate AS ExpLimit, lim.OutstandingQ
"FROM
"      (SELECT      AccountDebtor, AccountDebtorID, InvoiceCurrencyID, SUM(AmountAdvanced) AS
"FROM
"      dbo.tblTradePurchases AS wptp " & _
"WHERE
"      (TradeDate IS NULL) " & _
"GROUP BY AccountDebtor, AccountDebtorID, InvoiceCurrencyID AS (p.p RIGHT OUTER JOIN " & _
"      dbo.tblEntities AS ent INNER JOIN " & _
"      dbo.tblExposureLimits AS el ON ent.EntityID = el.EntityID LEFT OUTER JOIN " & _
"      (SELECT      SUM(COALESCE (inv.FaceValue, 0) - COALESCE (par.DeferredPurchasePrice, 0) + COALESCE (rem
"      AS OutstandingBalance, inv.PlatformID, tblTradePurchases.PurchaseProgramID, inv.InvoiceCurrencyID, inv.Aco
"FROM
"      dbo.tblInvoices AS inv LEFT OUTER JOIN " & _
"      dbo.tblTradePurchases ON inv.TradePurchID = dbo.tblTradePurchases.TradePurchID LEFT OUTER JOIN " & _
"      (SELECT      InvoiceReferenceNumberID, SUM(AmountFulfilled) AS AmountFulfilled, SUM(DiscountFeeAdjustment
"      SUM(AmountOfFundRaisingCapital) AS AmountOfFundRaisingCapital, SUM(AmountOfFundRaisingQ) AS AmountOfF
"FROM dbo.tblFulfillances " & _
"WHERE
"      (ResittanceDate <= ' " & Me.txtdate & "') " & _
"GROUP BY InvoiceReferenceNumberID AS rem ON inv.InvoiceReferenceNumberID = rem.InvoiceReferenceNumberID
"      (SELECT      TransactionFee, DiscountFee, DeferredPurchasePrice, PlatformFee, AmountPaid, InvoiceRefere
"FROM dbo.tblPurchases " & _
"WHERE
"      (PurchaseDate <= ' " & Me.txtdate & "') AS par ON inv.InvoiceReferenceNumberID = par.Invoice
"GROUP BY inv.PlatformID, tblTradePurchases.PurchaseProgramID, inv.InvoiceCurrencyID, inv.AccountDebtorID, I
"HAVING
"      (dbo.tblTradePurchases.TradeDate = ' " & Me.txtdate & "') AS lim ON el.CurrencyID = lim.Cu
"      (SELECT      CurrencyID, FxRate, FxRate " & _
"FROM
"      dbo.tblRates) AS fx ON el.CurrencyID = fx.CurrencyID " & _
"WHERE
"      (el.FxRate = ' " & Me.cboFund & "') AND (el.FundAccountDebtorExposureLimit / FX.FxRate > 0);/"

```

- `strSql = "SELECT * FROM myTVF('" & me.txtDate & "'," & me.cboFund & "');"`

Table Value Function

- Demo

Shameless Plug & Thank You

- I have a set of **free** add-ins available to download for Access
 - [AccessUI Ribbon and Tree Builder](#)
 - [AccessUI Data Source Manager](#)
 - [AccessUI VBATaskDialog](#) - Updated!
- Contact me at kevin@accessui.com