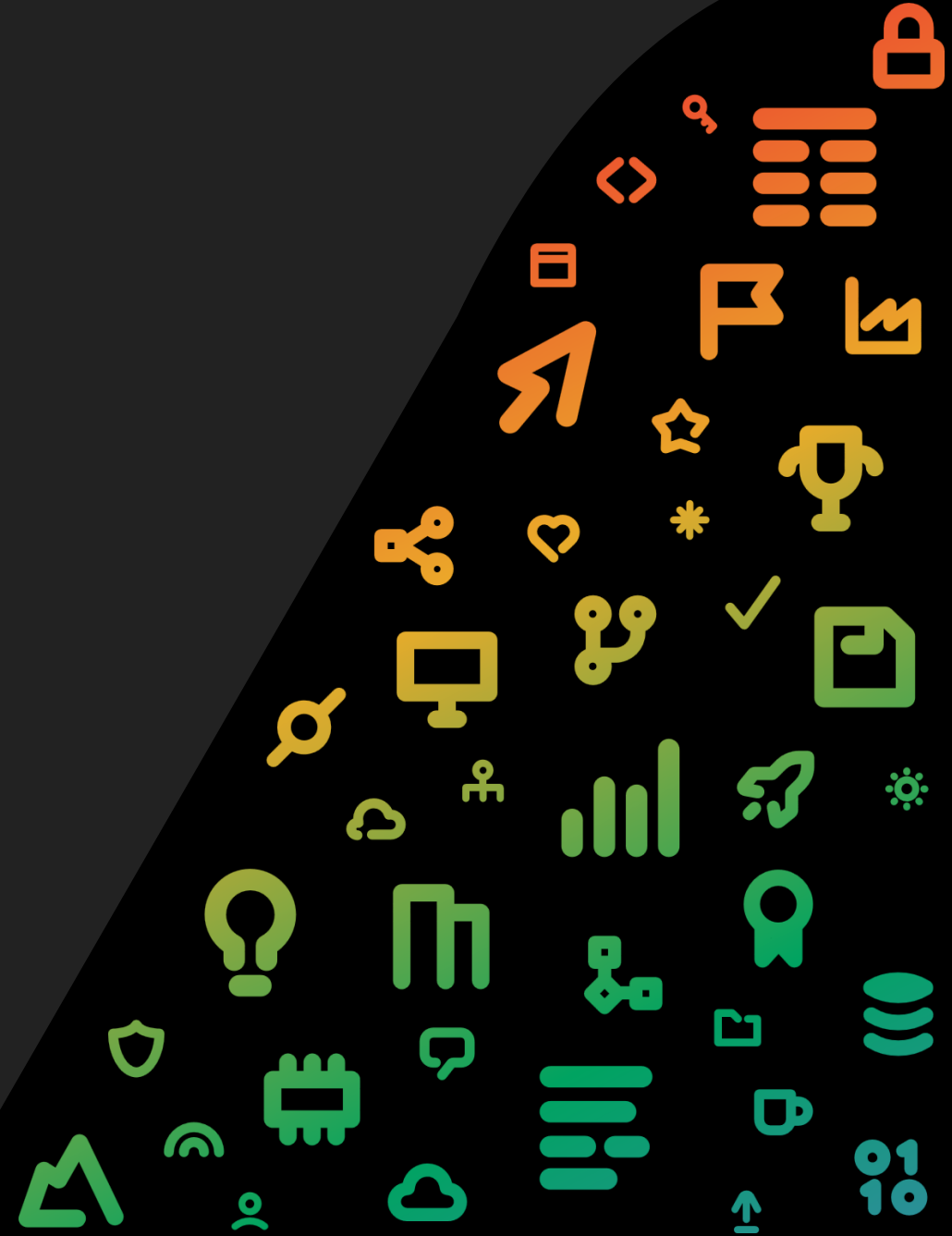




SQL Titbits for the Inexperienced

Erland Sommarskog

Data Platform MVP



Erland Sommarskog



Independent consultant based in Stockholm

SQL Server MVP since 2001

 esquel@sommarskog.se

 Slides and scripts: <http://www.sommarskog.se/present>



These are the Titbits

- GO is not an SQL statement.
- NULL values and the NOT IN trap.
- The Virtue of Using Aliases.
- The CASE Expression.
- ISNULL and COALESCE.
- UNION and UNION ALL.
- CTEs and Derived Tables.
- Temp Tables and Table Variables.

GO Is Not an SQL Statement

[01_go.sql](#)

- GO is never seen by SQL Server, it is an instruction to the query tool to split up the script in batches.
- The tool sends the text up to the first GO to SQL Server and waits for response before sending the next batch.
- Query tools understand GO, but client APIs does not. In your own program, you need to parse out GO yourself.
- CREATE PROCEDURE / FUNCTION / VIEW / TRIGGER must be in a batch of their own.
 - Thus, GO marks the end of a stored procedure!

NULL values

[02 NULLs.sql](#)

- NULL represents an *unknown* value.
- SQL has three truth values: TRUE, FALSE and UNKNOWN.
- Comparisons with NULL yield UNKNOWN.
- WHERE, ON etc only include a row if the condition is TRUE.
- Use IS NULL and IS NOT NULL to check for NULL.
- TRUE **OR** UNKNOWN => TRUE.
- TRUE **AND** UNKNOWN => UNKNOWN.
- **NOT** UNKNOWN => UNKNOWN.

The NOT IN Trap

[03_NOT_IN.sql](#)

```
SELECT * FROM dbo.tbl  
WHERE col1 NOT IN (SELECT col2 FROM dbo.tbl2);
```

- Return no rows if col2 is NULL for *any* row in tbl2.
 - Because that unknown NULL value may or may not be equal to col1.
- Add WHERE IS NOT NULL to subquery to avoid the trap:

```
SELECT * FROM dbo.tbl  
WHERE col1 NOT IN (SELECT col2 FROM dbo.tbl2  
                    WHERE col2 IS NOT NULL);
```

The Virtue of Column Aliases

[04_ColPrefixes.sql](#)

- When a query refers to more than one table, define aliases for all tables, and prefix all columns with the aliases.
- Avoids surprises when you use the wrong column name in subqueries.
- Prevents errors with ambiguous column name when a column is added later to a table.
- Makes the code easier to understand for outsiders.
- Table names instead of aliases? Makes it difficult to see the forest for the trees.

The CASE Expression

[05_CASE.sql](#)

- There is no CASE statement in SQL. There is a CASE *expression*.
- The result of a CASE is the THEN expression of the first WHEN condition that evaluates to TRUE.
- If no WHEN evaluates to TRUE, the result of CASE is the value for ELSE, or NULL if ELSE is absent.
- THEN/ELSE must be followed by an expression that evaluates to a *single* value – even if it is a subquery.
- And, no, you cannot call stored procedures in CASE.

CASE and Data Types

[06_Datatypes.sql](#)

- SQL is a *statically* typed language.
 - But with more implicit conversions than in other languages.
- A CASE expression always returns the one and same data type, even if you mix types.
- SQL Server applies a strict type-precedence list. Type(s) with lower precedence is converted to the type with the highest.
- This applies to all expressions, not only CASE.
- Better to convert explicitly for clarity.

Simplified Precedence List

1. Highest: sql_variant.
2. Date and time.
3. Numbers.
4. Strings.
5. Binary.

Reference:

Complete Precedence List

- | | | | |
|---------------------|----------------------|----------------|------------------------|
| 33. binary (lowest) | 24. uniqueidentifier | 18. tinyint | 9. time |
| 32. varbinary | 23. timestamp | 17. smallint | 8. date |
| 31. varbinary(MAX) | 22. image | 16. int | 7. smalldatetime |
| 30. char | 21. text | 15. bigint | 6. datetime |
| 29. varchar | 20. ntext | 14. smallmoney | 5. datetime2 |
| 28. varchar(MAX) | 19. bit | 13. money | 4. datetimeoffset |
| 27. nchar | | 12. decimal | 3. xml |
| 26. nvarchar | | 11. float | 2. sql_variant |
| 25. nvarchar (MAX) | | 10. real | 1. CLR types (highest) |

ISNULL vs. COALESCE

[07_ISNULL_coalesce.sql](#)

- `SELECT ISNULL(A, B), COALESCE(A, B)`
Both return A if A is non-NULL, else B.
- ISNULL is a function and takes exactly two arguments.
 - The return type is always the type of the first argument.
- COALESCE is a shortcut for a CASE expression. It can take any number of parameters.
 - Return type follows the normal precedence rules.

ISNULL vs. COALESCE, cont'd

- Watch out for `COALESCE(<subquery>, val)`, same as `CASE WHEN <subquery> THEN <subquery> ELSE val END.`
- Subquery may be evaluated twice – and may return NULL the second time. This not an issue with ISNULL.
- COALESCE is ANSI-standard, ISNULL is proprietary.
- Stick with ISNULL – the traps are fewer.
- It's also easier to spell...

UNION and UNION ALL

[08_UNION.sql](#)

- UNION and UNION ALL combine the result sets of two or more queries into a single result set.
- Column names are taken from the first query.
- ORDER BY can only be at the end; applies to the full query.
- UNION removes duplicate rows from the result sets, *including* duplicates within the sets.
- UNION ALL retains duplicates.

More on UNION ALL

- Tip: Never use UNION, always use UNION ALL. This is what you want 95 % of the time anyway.
- Checking for duplicates comes with a cost in performance.
- If you want distinct values, wrap UNION ALL in an outer SELECT DISTINCT – makes it clearer what you are doing.

CTEs and Derived Tables

[09_CTE temptables.sql](#)

- CTEs (Common Table Expressions) and Derived Tables are *logical* building blocks that you can use to build a more complex query from smaller queries.
- They may never be computed as such, as the optimizer may recast computation order.
- They are very similar in nature. CTEs have a name, derived tables only an alias.
- Which to use is a matter of taste.

Derived Tables

- Instead of placing a table or a view after FROM or JOIN, you can use a subquery, that is a *derived table*.
- A derived table cannot refer to anything in the outer query.
- It *must always* have an alias.

Common Table Expressions (CTE)

- On the top of your query, you can define one or more CTEs.
- The first CTE is introduced by `WITH` followed by the name, remaining CTEs are just introduced with comma + name.
- You use the name of the CTE in the rest of the query just like a table.
- Each occurrence of the name is replaced by the text of the CTE.
- When you have multiple CTEs, they are often refinement of each other – but they can also be independent.

Temp Tables and Table Variables

[09_CTE temptables.sql](#)

- Temp tables: CREATE TABLE, name starts with a single #.
- Table variables: DECLARE @tvar TABLE (...).
- Both serve the same purpose: a working area private to the process and not visible to other processes.
 - Two processes can create a temp table or declare a table variable with the same name at the same time without conflict.

Temp Tables and Table Vars, cont'd

- When defined in an SP they go away when procedure exits.
- A temp table created on top level goes away when process exits, or you say DROP TABLE. A table variable only lives for a single batch.
- Temp tables can be accessed by inner procedures. Table variables cannot.
- Both live in tempdb.
- Myths have it that table variables are memory-only and are not being logged. Not true!

Temp Tables vs Table Vars, Performance

- Table variables are known to be prone to bad performance.
- Tip: if you have a performance issue with a query that uses a table variable, try replacing it with a temp table.
- But the full story is more complicated and sometimes a table variable gives better performance.
- Table variables are OK if you only have a handful of rows, but if you expect more, use a temp table.

CTE vs Temp Tables

- Using a temp table for an intermediate result means that it is always computed as such and materialised.
- Thus, a CTE can be expected to be faster than a temp table.
- With complex or convoluted CTEs, it may be difficult for the optimizer to get estimates right.
- In that case, you can help the optimizer by storing partial results in a temp table.
- Using a temp table can also make your debugging easier.

Summary I

- GO is not an SQL statement – it's a batch separator.
- NULL is an unknown value, all comparisons with NULL yield UNKNOWN.
- Use IS [NOT] NULL to check for NULL.
- NOT IN (subquery) does not give any results when the subquery returns one or more NULL values.
- Add WHERE IS NOT NULL to avoid the trap.
- Always use column aliases when there is more than one table in a query.

Summary II

- In SQL, CASE is an expression not a statement.
- SQL is a statically typed language. (But with (too) many implicit conversions.)
- If you mix datatypes in a (CASE) expression, the return type is the type with highest precedence.
- Use ISNULL over COALESCE to avoid traps with COALESCE, not the least with subqueries.

Summary III

- UNION and UNION ALL permit you to combine two or more result sets into a single one.
- UNION removes duplicates, UNION ALL does not.
- Always use UNION ALL. Wrap in DISTINCT to make it clear what you want!

Summary IV

- CTEs and derived tables are logical tools to build complex queries.
- They are necessarily not computed as such.
- Temp tables and table variables are private work areas.
- Temp tables often give better performance than table variables, particularly with larger amounts of data.
- CTEs and derived tables are generally faster than temp tables – but exceptions are commonplace.

Session evaluation

Your feedback is important to us



Evaluate this session at:

www.PASSDataCommunitySummit.com/evaluation



Thank you

Slides and scripts:

<http://www.sommarskog.se/present>

SQLTitbits database: [SQLtitbits db.sql](#).

Northgale: [instnwnd.sql](#) + [Northgale.sql](#).

Erland Sommarskog

esquel@sommarskog.se



Temp Tables and Name Resolution

[10_tmptbl_TV.sql](#)

- When you create a procedure that creates a temp table, the table typically does not exist at that time.
 - Therefore, SQL Server suppresses errors about missing tables.
- While helpful, this can serve to mask errors with other tables in a query with a temp table.
- Tip: attempt to give your temp tables specific names, and do not use generic names like #temp, #t etc.
 - This can save you from nasty surprises with nested procedures.
- Table variables do not have these problems.
 - Use while developing and change to temp tables when you are done?