

# **SQL TITBITS FOR THE INEXPERIENCED**

**Erland Sommarskog  
SQL Server MVP**

**Virtual SQL Saturday #971 Oslo  
Aug 29<sup>th</sup> 2020**





**Erland Sommarskog**  
**Independent consultant based in Stockholm**  
**SQL Server MVP since 2001**  
**<http://www.sommarskog.se>**  
**[esquel@sommarskog.se](mailto:esquel@sommarskog.se)**

Slides and scripts are available on  
<http://www.sommarskog.se/present>  
and the SQL Saturday web site



# These are the Titbits

- GO is not an SQL statement.
- NULL values and the NOT IN trap.
- EXISTS and NOT EXISTS.
- The CASE expression.
- Some on Data Types in SQL Server.
- UNION and UNION ALL.
- CTEs and Derived Tables.
- Temp Tables and Table Variables.



# GO Is Not an SQL Statement

[00\\_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 in a column or variable.
- TRUE OR UNKNOWN => TRUE.
- TRUE AND 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);
```



# EXISTS / NOT EXISTS

[04 EXISTS.sql](#)

```
SELECT * FROM dbo.CustomerResponsibles CR
WHERE EXISTS (SELECT * FROM dbo.Customers C
              WHERE C.CustRespID = CR.CustRespID);
```

- The subquery with [NOT] EXISTS is *logically* evaluated for every row in the outer table.
- If the subquery returns at least one row, EXISTS returns TRUE, else FALSE. (But never UNKNOWN!)
- [NOT] EXISTS is great for multi-column conditions.
- The WHERE clause for an EXISTS subquery should always have at least one condition to a table in the outer query.



# 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 there is no ELSE.
- 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.



# Data Types in SQL Server

[06\\_Datatypes.sql](#)

- SQL is a *statically* typed language.
  - But with more implicit conversions than in other languages.
- An expression always returns the one and same data type.
- If two types meet in an expression, the one with lower precedence is converted to the type with higher.
  - If an implicit conversion exists, that is. Else, there is a compilation error.
- Better to convert explicitly for clarity.



# Simplified Precedence List

5. Lowest: (var)binary.

4. Strings.

3. Numbers.

3c. Integer numbers.

3b. Decimal.

3a. Floating-point.

2. Date and time.

1. sql\_variant.



# 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)



# Comparing Strings to Other Types

- Since strings have low precedence, you often get errors if you compare strings to numbers or datetime values:

```
WHERE stringcol = 1234  
WHERE stringcol = @dateval
```

Not all values in the string column may be numbers/dates.

- For the computer `1234` and `'1234'` are very different.
- Convert number/date variables to string explicitly.
- Or convert the string value with `try_convert` which returns NULL if conversion is not possible.



# UNION and UNION ALL

[07 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.
- If the data type is different for the same column in the individual queries, there will be conversion according to the precedence rules, just as for CASE.
- Better to convert explicitly for clarity.
- ORDER BY can only be at the end; applies to the full query.



# More on UNION [ALL]

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



# CTEs and Derived Tables

[08\\_CTE temptables.sql](#)

- CTEs (Common Table Expressions) and Derived Tables are *logical* building blocks that permits you 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

[08\\_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 that is 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.
- Myth has it that table variables are memory-only. Not true!



# **Temp Tables vs Table Vars, Performance**

- Table variables are known to be prone to bad performance.
- Tip: if you run into a performance issue with a query that uses a table variable, try replacing it with a temp table.
- But the full story is a lot 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 (or table var) 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 does not give any results when the subquery returns one or more NULL values.
- Add WHERE IS NOT NULL or use NOT EXISTS to avoid the trap.
- In SQL, CASE is an expression, not a statement.



# Summary II

- SQL is a statically typed language. Expressions and columns in a result set have a static data type.
- Many implicit conversions are permitted. They follow a precedence list that defines which type “wins”.
- UNION [ALL] permits 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 III

- 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.



# The Final Titbit

Erland Sommarskog – [esquel@sommarskog.se](mailto:esquel@sommarskog.se).

Slides and scripts on <http://www.sommarskog.se/present> as well as the SQL Saturday web site.

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

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