

# SQLskills Immersion Event

## SQL Server Programming for Developers

### Module 7: Optimizing Procedural Code

#### Part 2: Triggers, Functions, and Views

Bob Beauchemin  
BobB@SQLskills.com



### Overview

- Triggers
- Views
- Types of User-Defined Functions
- Single Statement vs Multi-statement UDFs
- Stored Procedures v. Scalar Functions
  - Recompilation
  - Row-based operations
  - Using SQLCLR functions



## Triggers

- **Triggers fire as a result of**
  - DML Action Statements INSERT/UPDATE/DELETE
  - DDL
  - Login (SQL Server 2005 SP2 and above)
- **SQL Server supports**
  - AFTER TRIGGER
  - INSTEAD OF TRIGGER
- **Triggers fire once per statement**
  - No matter how many rows affected
  - MERGE only fires triggers once
- **Trigger is part of the same transaction that caused the data modification**

## Coding inside a trigger

- **SQL Server exposes pseudo-tables in triggers**
  - INSERTED and DELETED
  - Materialized in tempdb since SQL Server 2005
  - These tables are not indexed
- **You can update base table through**
  - UPDATE WITH JOIN
  - MERGE (SQL Server 2008)
  - Cursor
- **You can determine columns updated**
  - UPDATE(col) or COLUMNS\_UPDATED

## Views

- **A view is usually treated as a macro**
  - View expanded as part of query parsing
  - View can be updatable – subject to limits
  - Some joins may be eliminated for some queries
  - Multiple nested views can impact performance
- **Views can be indexed**
  - Entire view materialized
  - Indexed view is treated as a table
    - Enterprise edition differs in view substitution
    - EXPAND VIEWS / NOEXPAND hint
  - Kept in sync with base table when changes

## Specialized Non-Indexed Views

- **Partitioned Views**
  - Based on "multiple copies" of same table with different key ranges
  - Allows for partition (table) elimination
- **Distributed Partitioned Views**
  - Tables exist on different instances
  - More overhead when remote partitions not eliminated

## Types of User-Defined Functions

- **Table-valued functions**
  - Single statement - Like a view with parameters
  - Multi statement - Similar in appearance to stored procedure, but not as optimizable
- **Scalar functions**
  - Return single value
  - Executed once-per-row
  - Database access is optional (e.g. formulas)
  - If deterministic and precise can be used in
    - Indexed views
    - Persisted computed columns (SQL Server 2005)

## Functions vs. Stored Procedures

- **Scalar Functions** force row operations
- **Stored Procedures** can perform set operations
- **Table Functions** cannot use INSERT EXEC to populate the table variable
- **Table Function's** table variables can only have "key" indexes created
- **Tables** created in **Stored Procedures** can have any type of index created
  - i.e. non-unique or composite indexes

## Function Limitations

- **Cannot affect the state of the server**
  - Environment or updating rows
- **Cannot execute a stored procedure (or dynamic SQL)**
- **Cannot create or use temp tables**
  - Can use table variables
- **Parameters with defaults may not be omitted, must use DEFAULT on invoke**
- **Functions must be invoked by 2-part name**
  - Unless executed with EXECUTE

## Function Additions

- **RETURN NULL ON NULL INPUT**
  - If any parameters are NULL, just return NULL
  - Performance improvement
- **Can specify WITH SCHEMABINDING**
- **Can be DETERMINISTIC**
  - Always returns same value with same input
- **Can be precise**
  - Only uses precise numeric types
- **Determinism and precision**
  - are required for some function uses
  - are declared when .NET function
  - are determined by engine when T-SQL function

## Schemabinding – Functions & Views

- **Schemabinding indicates that objects used by UDF/View must remain the same**
  - Attempted updates to these objects fail unless UDF is dropped
  - Like schemabinding in views
- **Schemabinding can affect performance**
  - UDFs can bind to functions that update data
  - "Update anomaly" protection added if SQL Server cannot guarantee no updates
  - This protection can greatly affect performance
    - Even if UDF actually does not access/update data

## Single vs. Multi-statement UDFs

- **Single statement UDF is like a VIEW with parameters**
  - Indexes on the base table can be used
  - IO counts based on base table(s) access
- **Multi-statement UDF must populate a TABLE variable**
  - Similar to selecting into a temp table or table variable and querying from there
  - IO counts include base table(s) and final product tables

## TVF Performance (1)

- **Single statement TVF produces three query plan objects**
  - One for the TVF body
    - Parse tree for a view
    - Reused each time
  - One for the query containing the TVF
    - Optimized as an adhoc query
    - Reused only if query text is exactly the same
  - Also, parse tree for CHECK constraint

## TVF Performance (2)

- **Execution of Multi-statement TVF yields two query cache objects**
  - One for the TVF body
    - Optimized as a procedure
    - Reused each time
  - One for the query containing the TVF
    - Optimized as an adhoc query
    - Reused only if query text is exactly the same
- **Multi-statement TVFs have no statistics**
  - Better cardinality estimates with inline TVF or temp table

## Simple Scalar Functions

- **Simple Scalar Functions executed per-row**
  - These functions should not access the database (this access will be per-row)
  - These functions will be faster in .NET code
    - .NET Functions faster if no database access
  - Scalar functions cannot optimized by the engine, unless index is created
    - Use persisted computed column (SQL Server 2005)
    - Can even create index on this column - function must be deterministic to create index
    - Trades storage for speed in per-row execution

## Scalar Function Uses

- **Computed columns**
  - These can be persisted
  - Persisted computed must be deterministic
- **Check constraints**
  - Be careful with multi-row INSERTs
  - Can refer to other columns in same table
- **Defaults**
  - Cannot refer to other columns in same table
- **SQL statements**
  - Column list - not inline



## Scalar UDF Query Plan Reuse

- **Execution of scalar UDF yields two query cache objects**
  - One for the UDF body
    - Optimized as a procedure
    - Reused each time
    - Plan is EXECUTED N times (once per row in query containing the UDF)
  - One for the query containing the UDF
    - Optimized as an adhoc query
    - Reused only if query text is exactly the same

## Using .NET Scalar UDFs

- **SQLCLR UDFs are precompiled**
  - No T-SQL interpreter
  - Better for:
    - Mathematics
    - String manipulation
    - Array processing
  - But... database operations in .NET UDFs are Dynamic SQL
    - Can cause recompiled
    - Subject to same security restrictions
    - Subject to same SQL injection potential
  - Pass in database values (from T-SQL) if needed
  - Invoked once-per-row, just like T-SQL UDFs

## Summary

- **Triggers**
  - Part of triggering transaction
  - Pseudo-tables materialized in tempdb
- **Views**
  - Vanilla, indexed, and partitioned
- **Stored Procedures v. Scalar Functions**
- **.NET for simple scalar functions**
  - No database access
  - Persisted computed column possible
- **Watch out for row-based operations for database-accessing functions**

# Questions?