

SQLskills Online Immersion Event

IEQuery: Immersion Event on Fixing Slow Queries,
Inefficient Code, and Caching/Statistics Problems

Part II: Removing Anti-Patterns in Transact-SQL

Jonathan Kehayias
Jonathan@SQLskills.com
@SQLPoolBoy





Jonathan Kehayias

Principal Consultant, SQLskills



Jonathan@SQLskills.com



[@SQLPoolBoy](https://twitter.com/SQLPoolBoy)



www.sqlskills.com/blogs/jonathan

Trainer/Speaker

In addition to consulting, I teach content for our IE0: Accidental DBA course, IEAG: Clustering and Availability Groups course, and our IEPTO2: Performance Tuning and Optimization course

Data Platform MVP

I have been thankfully recognized as an MVP by Microsoft since 2008

Author

Microsoft SQL Server 2012 Internals

Professional SQL Server 2008 Internals and Troubleshooting

Troubleshooting SQL Server: A Guide for the Accidental DBA

Schedule – Wednesday

- **Lecture 1:** Set based concepts for developers
 - How SQL Server processes different query patterns
 - Design considerations that affect performance
 - Data access patterns
 - Normalization and data-types
 - Eliminating row-by-row processing
 - CURSORS and WHILE Loops
 - Scalar User Defined Functions
 - Table Valued Functions
 - Using FOR XML and a table of numbers

How SQL Server Processes Queries

Row Mode

- Efficient for OLTP workloads
- Typical for row store format tables
- Execution tree operators read each required row across all columns

Batch Mode

- Efficient for data warehouse and scanning large amounts of data
- Typical for columnstore format tables
- Processes multiple rows as a batch and eliminates the need for an exchange operator during parallel processing

SELECT Statements

- Define the result set format and columns of data to return in the column list
- Define the tables that contain the source data in the FROM and JOIN clauses
- Defines how the tables logically are related to one another in the ON clause following a JOIN or the WHERE clause
- Defines the filtering conditions for the output rows in the WHERE or HAVING clause of a GROUP BY
- Does not define how SQL Server will execute the request unless specific hints are used (non-procedural)

Query Execution Plans

- Parsing
 - Syntax is checked to determine if the query is written properly
 - Produces a parse tree that is handed off
- Binding
 - Algebrizer analyzes the parse tree for name resolution
 - Do the tables exist, are columns in the where clause in the tables, permissions
 - Produces a query processor tree that is then handed to the optimizer
- Optimization
 - The optimizer attempts to determine the most efficient method of executing the request based on costs
 - Produces an execution plan that is used by the execution engine to process the request

Operator Precedence

Level	Operators
1	~ (Bitwise NOT)
2	* (Multiplication), / (Division), % (Modulus)
3	+ (Positive), - (Negative), + (Addition), + (Concatenation), - (Subtraction), & (Bitwise AND), ^ (Bitwise Exclusive OR), (Bitwise OR)
4	=, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Assignment)

Normalization

- Process of organizing data in a database into tables with relationships to other tables
 - Better flexibility – design changes
 - Reduce redundancy – wasted disk/memory space
 - Enforce data integrity – reduce where data changes must occur
 - Remove inconsistent dependencies – group data into related subjects
- Rules define a standard of database normalization
 - First Normal Form
 - Second Normal Form
 - Third Normal Form – typical for OLTP

First Normal Form (1NF)

- Eliminate repeating groups in individual tables
 - Does not store multiple values within a single column
 - Defines each value as atomic – cannot be broken to smaller pieces
- Create a separate table for each set of related data
 - Does not use multiple columns in a table to store similar data
 - E.g. Phone Numbers, vendor codes
- Identify each set of related data with a primary key
 - 1NF requires a unique constraint on the table - no exact duplicate rows exist

First Normal Form (1NF)

Name	Address	Phone1	Phone2	PostalCode	Country
Jonathan Kehayias	123 Disney Hwy, Orlando, FL 34582	407-528-1124	727-485-3325	34582	USA
Paul Randal	36 BAOSHAN JIUCUN, BAOSHAN DISTRICT	251-112-2425	927-555-1212, 855-243-8514	201900	CHINA

ID	Name	Address	Phone
1	Jonathan Kehayias	123 Disney Hwy	407-528-1124
2	Jonathan Kehayias	123 Disney Hwy	727-485-3325
3	Paul Randal	4432 Nowhere Lane	251-112-2425
4	Paul Randal	4432 Nowhere Lane	927-555-1212
5	Paul Randal	4432 Nowhere Lane	855-243-8514

Second Normal Form (2NF)

- Must first meet the requirements of 1NF
- Create separate tables for sets of values that apply to multiple records
 - Records should only depend on another table's primary key
 - E.g. Addresses stored in an address table and referenced by AddressKey in Customer, Orders, Shipping, etc tables
- Define relationships between tables with a foreign key
 - Enforce data integrity and prevent orphaned data

Second Normal Form (2NF)

PersonID	Name	Address	PostalCode	Country
1	Jonathan Kehayias	123 Disney Hwy	34582	USA
2	Paul Randal	36 BAOSHAN JIUCUN, BAOSHAN DISTRICT	201900	China

PhoneID	PersonID	PhoneNumber
1	1	407-528-1124
2	1	727-485-3325
3	2	251-112-2425
4	2	927-555-1212
5	2	855-243-8514

Third Normal Form (3NF)

- Ideal for OLTP applications
 - Reduces duplication of data
 - Typically free of INSERT/UPDATE/DELETE anomalies
- Must first meet the requirements of 2NF
- Eliminates columns that do not depend on the key
 - E.g. Cities, states, zip codes
 - May not always be followed for every table (see above examples for addresses)
 - Should be applied for frequently changing data that affects multiple relationships or records – only update parent table field

Third Normal Form (3NF)

PersonID	Name	Address	PostalCodeID
1	Jonathan Kehayias	123 Disney Hwy	1
2	Paul Randal	36 BAOSHAN JIUCUN	2

PhoneID	PersonID	PhoneNumber	PostalCodeID	PostalCode	Country	City	Provence
1	1	407-528-1124	1	34582	USA	Orlando	FL
2	1	727-485-3325	2	201900	China	BAOSHAN DISTRICT	Shanghai
3	2	251-112-2425					
4	2	927-555-1212					
5	2	855-243-8514					

Use Appropriate Data Types

- Understand the storage costs and implications of data types during schema design – especially for keys

Data type	Range	Storage
bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 Bytes
int	-2,147,483,648 to 2,147,483,647	4 Bytes
smallint	-32,768 to 32,767	2 Bytes
tinyint	0 to 255	1 Byte

Use Appropriate Data Types (2)

- Consider precision requirements for dates and times
 - DATETIME2 = 6, 7, or 8 bytes with nanosecond precision
 - 6 bytes for precisions less than 3
 - 7 bytes for precisions 3 and 4
 - All other precisions require 8 bytes
 - DATETIME = 8 bytes with fractions of second precision
 - Fraction of seconds rounded to .000, .003, or .007 seconds
 - SMALLDATETIME = 4 bytes with minute precision
 - DATE = 3 bytes
- Don't store dates or times as CHAR, VARCHAR, NCHAR, or NVARCHAR

Row-By-agonizing-Row Processing

- Certain constructs force SQL Server into RBAR (row-by-agonizing-row) processing of results
- Well-known:
 - WHILE loops
 - Cursors
- Less well-known:
 - Scalar user defined functions (changes in SQL Server 2019)
 - Correlated subqueries

Cursors and Loops

- Characteristics
 - Explicit cursor declaration
 - WHILE loop
 - SqlDataReader in application
- Problems
 - Row based processing over Set Based
- Replace with
 - Appropriate set based operation
 - Move looping code into SQLCLR or Middle Tier
 - Consume and dispose of SqlDataReader as quickly as possible

Correlated Sub-Queries

- Characteristics
 - Refer to the outer query in the inner query in SELECT statement
 - SELECT Statement used as column value in UPDATE
- Problems
 - May cause row-by-row processing to occur
 - Performance decreases exponentially as row count increases
- Replace with
 - Table Join
 - Derived Table Join
 - Cross Joined Table Valued Function

Scalar User Defined Functions

- Characteristics
 - Encapsulate common code blocks/business logic in a single call.
 - If columns are passed as parameters it is not inline
- Problems
 - Cause row-by-row processing to occur
 - Performance decreases exponentially with data access
- Replace with
 - Inline expressions
 - Derived Table Join
 - Cross Joined Inline Table Valued Function

Schedule – Wednesday

- **Lecture 2: Scalability and Testing**
 - Making row based processing scale when required
 - Understanding Sargability and Index Scans
 - Implicit Conversions
 - IN vs UNION ALL for performance
 - Profiling during development and testing
 - Fatal flaws to avoid

Scaling Row Based Processing

- Application design patterns
 - Multi-threaded apartment for asynchronous execution
 - Retrieve initial data set from SQL Server and execute row based processes on background or “worker” class threads in the application
 - Best for situations where initial data set is not changing
 - Watch for blocking conditions for long open result sets – fire hose cursor consumption of data
- SQL Server design pattern
 - Service Broker queue activation allows parallel processing of messages natively within SQL Server
 - Transactional based and ensures restart of processing after failure
 - Implemented entirely with TSQL as a part of the database

Sargability Matters

- A query is sargable (Search ARGument ABLE) if an index seek can be used to speed up the execution of the query
- Anti-patterns to sargable expressions include:
 - Functions in the WHERE clause
 - Implicit/Explicit data type conversions on a column
 - Leading wildcard expressions with LIKE '%<SearchTerm>'
 - Catch all queries and search procedures

Functions on WHERE Clause Columns

- Characteristics
 - Used to change the data stored to match criteria being checked
 - Conversion of data to a different type
- Problems
 - Causes Table/Index Scan over Seek
- Replace with
 - Appropriate Table Design to support business needs
 - Indexed/Persisted Computed Column
 - Indexed View
 - Other coding paradigm to eliminate Scan

Implicit/Explicit Column Conversions

- Characteristics
 - Column data type is of lower precedence than filtering parameter / joining column data type
 - Common in LINQ to SQL/EF and other ORMs
- Problems
 - Causes Table/Index Scan over Seek
- Replace with
 - Higher precedence column data type
 - Matching data type for filtering parameter

Catch-All Search Queries

- Characteristics
 - Used to search across multiple columns using parameters
 - Not all parameters require input values
 - WHERE clause similar to (`@Param1 IS NULL OR Column1 = @Param1`)
- Problems
 - No optimized execution plan
 - Causes Table/Index Scan over Seek
- Replace with
 - Separate search procedures for different parameters passed
 - Parameterized Dynamic SQL

Replacing IN with UNION

- Characteristics
 - WHERE clause similar to (Column 1 IN (12, 16))
 - WHERE clause similar to (Column1 = 12 OR Column1 = 16)
- Problems
 - May causes Table/Index Scan over Seek
 - May cause a range seek
- Replace with
 - Separate SELECT statements with WHERE clause for each criteria using UNION ALL to concatenate results to final output
 - Dynamic SQL to build the UNION ALL query string

Profile Your Workload During Development

- Learn to use Extended Events (2012+) or SQL Trace to profile your workload during testing
- Know the important events to watch for during development
 - Statement/Batch/RPC completed events
 - SP completed/Module End events – (procedure/trigger/function executions)
 - Execution warnings (sort, hash, missing join predicate)
- Profiling during development can uncover nasty RBAR issues and performance effecting side effects of trigger executions
- Be aware of “*observer overheads*” but not typically a problem with development/test workloads

Develop and Test Against Realistic Scale

- Development databases often do not contain realistic datasets which can hide/mask potential performance problems
 - Key Lookups on small data sets may become index scans on larger data sets
 - Missing index impacts may be hidden by data residing in memory for small data sets
 - RBAR problems are often hidden until data sizes scale up
- Testing a single execution in isolation is not load testing
 - Testing needs to be performed at scale through load generation to measure accumulated effects
 - Only testing at scale can identify *"death by 1000 cuts"* problems

Review

- Set based concepts for developers
 - How SQL Server processes different query patterns
 - Design considerations that affect performance
 - Eliminating row-by-row processing
- Scalability and Testing
 - Making row based processing scale when required
 - Understanding Sargability and Index Scans
 - Profiling during development and testing

Additional Resources

- SQLPerformance.com Articles
 - <https://www.sqlperformance.com/>
- Paul White's Blog
 - http://sqlblog.com/blogs/paul_white/
- Jonathan's Posts
 - Parallel Maintenance
 - <https://www.sqlskills.com/blogs/jonathan/parallel-maintenance/>
 - Implicit Conversions
 - <https://www.sqlskills.com/blogs/jonathan/finding-implicit-column-conversions-in-the-plan-cache/>
- Paul's Posts
 - DBCC CHECKDB performance and computed-column indexes
 - <https://www.sqlskills.com/blogs/paul/dbcc-checkdb-performance-and-computed-column-indexes/>

Thank you!

