SQL Server Query Performance Tuning:
A 12–Step Program

CONFIO is now part of solarwinds

START

CHECK FOR TABLES + ROWCOUNTS

1

Make certain you know the rowcount

Make sure you're working with tables (not views)

EXAMINE THE FILTERS

2

Examine WHERE and JOIN clauses — what is the filtered rowcount?

KNOW THE SELECTIVITY OF THE TABLES

3

Make sure to work with the smallest possible logical set

ANALYZE QUERY COLUMS

4

Most useful for RIGHT, LEFT, OUTER joins

Know when the predicate is applied – should be earlier rather than later

Look for select * or scalar functions (the more data brought back, the less optimal it may be to use certain functions)

Look for CASE, CAST, CONVERT

Sub-queries

SARG-able (make an index searchable so a full scan isn't needed)

REVIEW EXISTING KEYS, CONSTRAINTS AND INDEXES

5

Know what objects exist (avoid duplicating efforts later on)

Check to see if covering indexes can be created (avoid duplication, overlapping)

What is primary key definition, and is it clustered?

EXAMINE THE EXECUTION PLAN

6

Set statistics on (set statistics IO on + set statistics time on)

Check the actual plan, not the estimated plan

Run the plan

RECORD RESULTS

7

Look for logical reads (fewer logical I/Os means faster query)

Focus on logical I/O (number of logical reads)

ADJUST THE QUERY

8

Focus on most expensive operations first

RE-RUN THE QUERY

9

Record the results and compare

CONSIDER ADJUSTING INDEXES

10

Make small changes

Consider covering index—an index that includes every column that satisfies the query

Look to reduce logical i/o

Consider a filtered index (but not if you have parameterized statements)

RE-RUN THE QUERY

11

ENGINEER OUT THE STUPID

12

Code-first generators (EMF, LNQ, nHibernate) can be mis-used and bloat the plan cache

Cursors and row-by-row processing

Scalar functions

Abuse of wildcards (*) — pulling back too many rows

Join/query/table hints

Nested views that go across linked servers