SQL Optimization Essentials

blackbaud™
power your passion

ABOUT ME

- David Hendershot
- Performance Engineer for Blackbaud CRM
- 8 years with BBCRM, 4 on performance
- Live in Denver, CO

AGENDA

- SQL Performance Principles
- Tools for Measuring and Analyzing Performance
- Using Indexes Effectively
  - SARGABLE
  - Parameter sniffing
- Restructuring Queries for Performance
  - CTEs, table variables, temporary tables
  - Dynamic SQL

SQL PERFORMANCE PRINCIPLES

- Allow SQL to eliminate rows as early and efficiently as possible
  - Write clauses that allow SQL to eliminate rows using data from a single table (i.e. no OR clauses)
  - Use highly selective where clauses
  - Filter on columns with indexes
- Operate on small result sets in the OLTP database
  - A data warehouse is more appropriate for pulling large datasets
- Request only the columns you need
  - More columns mean less efficient indexes or key lookups
- Do not repeat work
  - Consider storing data in variables and temp tables

MEASURING AND ANALYZING PERFORMANCE

- SQL Server Profiler and SQL Server Extended Events
  - Captures event based information such as a statement finishing or a query plan being calculated
  - Can specify which events to collect
  - Can choose what column information to include (costs, process, etc.) and filter on those columns
  - Captures exact costs
  - If not properly filtered, degrades performance
- Query Execution Plans
  - Shows the order and method SQL Server uses to get the requested information
  - Can be viewed in SQL Server Management Studio or tools like SQL Sentry
  - Collected through SSMS, Profiler, Extended Events, or the Dynamic Management Views

USING INDEXES EFFECTIVELY

HOW DOES SQL DECIDE WHAT INDEXES TO USE?

- Statistics
  - SQL Server keeps a statistics histogram of each field
  - Allows for fairly accurate estimates for # of rows matching a criteria
- Indexes
  - Clustered and non-clustered
    - Clustered – The ordering of the actual data (like name in a dictionary)
    - Non-clustered – A reference to where the data can be found (like an index in the back of a book)
  - Scans and seeks
    - Scan – Read the entire index sequentially
    - Seek – Look up specific records non-sequentially
    - Seeks are costlier per row but cheaper overall if small numbers of rows are selected
  - Key lookups
    - Occur when a non-clustered index is not “covering” (doesn’t have all the fields)

KEEP CLAUSES SARGABLE

- Sargable
  - **S**earch **Arg**ument **Able**
  - Means an index can be used to run a query more quickly
- Non-sargable conditions
  - “OR” statements
  - “LIKE” statements beginning with wildcards
  - Functions
  - Arithmetic on the indexed field

PARAMETER SNIFFING

- Upon first execution of a stored procedure, SQL creates a plan to retrieve the data
- SQL uses the parameters of stored procedures/sp_executesql to generate plans
- Does not create a new plan for new parameters
- Plans for each statement are generated on compilation of the stored procedure not execution of the statement
  - A plan for the statement is generated even if it is not executed
  - Does not apply to stored procedures or dynamic SQL
- Only applies to parameters
  - Variables declared inside a procedure are treated as unknown

HOW TO RESOLVE PARAMETER SNIFFING ISSUES

- Recompile
  - Recompile hints can be used at the statement or stored procedure level
- Separate out different paths into different functions/procedures
- Use dynamic SQL

RESTRUCTURING QUERIES

CTES, TABLE VARIABLES, TEMPORARY TABLES

- CTEs
  - Virtually the same as a sub-select (derived table)
  - If accessed multiple times, will be run multiple times
- Table variables
  - Treated as single row tables
    - No indexes or statistics (except primary key)
  - Less overhead: No transaction log, locks, rollbacks
  - Does not count as a schema change for recompilation purposes
    - Can be created in functions
- Temporary tables
  - Indexes and statistics can be created
  - More overhead: Transaction logs, locks, rollbacks, recompiles

DYNAMIC SQL

- Features of dynamic SQL
  - Compiled at execution time
  - New plan generated for each difference in the SQL text (but not parameters)
- Advantages of dynamic SQL
  - Take parameter values into account to make efficient SQL
    - Avoid unneeded joins and “Field = @Parameter or Field is null” statements
  - Can take advantage of variable values set at runtime
- Disadvantages of dynamic SQL
  - Using literals instead of parameters has more overhead
    - Can flood plan cache
    - Extra compilations cost CPU and memory
  - Harder to read and write

RESOURCES AND FURTHER STUDY

- SQL Sentry - www.**sqlsentry**.com/
- Grant Frichey - *SQL Server Execution Plans*
  - http://download.red-gate.com/ebooks/SQL/eBOOK_SQLServerExecutionPlans_2Ed_G_Fritchey.pdf
- SQL Pass - www.**sqlpass**.org/

QUESTIONS?

- Contact me at david.hendershot@blackbaud.com