SQL Server Performance Tuning

Steve Stedman – Programmer and Database Consultant

Email: Steve@SteveStedman.com

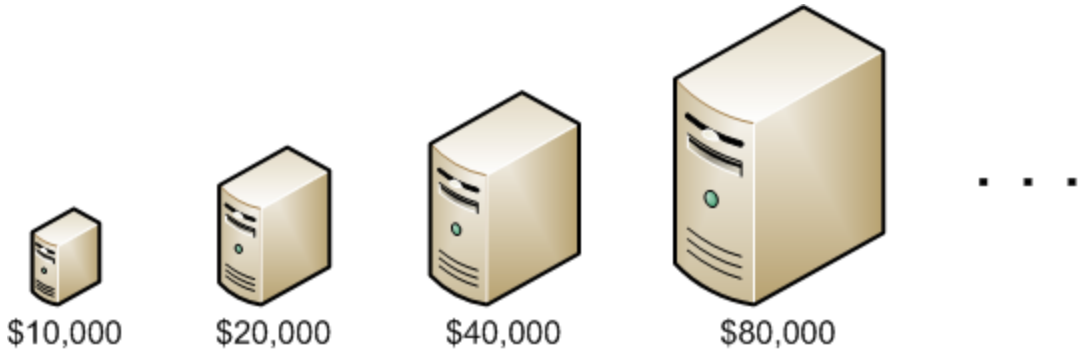
Overview

- General Scaling Options
- General Performance Tuning
- Indexes
- Query Tuning
- Procedure Cache Tuning
- Understanding Table Sizes

- All referenced queries and stored procedures are available at http://SteveStedman.com/sql

Scaling SQL Server with Bigger Hardware

- Purchase a larger server, and replace the existing system.
- Works well with smaller systems.
- Cost prohibitive for larger systems.
- Can be a temporary solution.



Scaling SQL Server with More Hardware

- Purchase more hardware and split or partition the database.
- Partitioning can be either vertical or horizontal.
  - Vertical: Split the databases based on a specific demographic such as time zone or zip code.
  - Horizontal: Split components out of one database into another.

Scaling SQL Server without adding hardware.

- Adjusting and rewriting queries.
- Adding indexes.
- Removing indexes.
- Re-architecting the database schema.
- Moving things that shouldn't be in the database.
- Eliminating redundant work on the database.
- Caching of data.
- Other performance tuning techniques.

Which Scaling Option is Best?

- Bigger hardware?
- More hardware?
- Tuning without adding more hardware?

- There is a time and a place for each of these depending on your specific environment.
- This presentation covers tuning without adding more hardware.

Why is Performance Tuning Necessary?

- Allowing your system to scale.
  - Adding more customers.
  - Adding more features.
- Improve overall system performance.
- Save money by not wasting resources.
  - The database is typically one of the most expensive resources in a datacenter. Make the most of it.

Indexes

- Non-Clustered Indexes
  - Traditional Indexing – contains pointers to the data.
- Clustered Indexes
  - Reorganizes the actual data on disk.
- Covered Indexes
  - Returns query results without accessing the base table.
  - Can lead to major performance increases.
  - Applies to Non-Clustered Indexes.

Non-Clustered Indexes

- Contain only the data specified in the index.
- Do not change the base layout of the tables.
- Use pointers to get to the data.
- Can be created on most data types including char(), varchar(), and uniqueidentifiers.
- Only one non-clustered index can be used per table reference in a query.
- Can improve performance with multiple columns.

Clustered Indexes

- Causes base table structure to change.
- Only one clustered index per table.
- Should never contain char(), varchar(), varbinary(), uniqueidentifiers, or other large or widely distributed identifiers.
- Can significantly increase the size of a table and the database.
- Can increase performance.

Database Space Used by Indexes

Table Name	Rows	Disk Space (MB)	Index Space (MB)	Total Space (MB)
Table1	2,034,998	450	872	1,322
Table2	2,423,745	476	639	1,115
Table3	2,194,080	318	656	974
Table4	396,241	598	60	658
Table5	1,812,884	223	371	595

Use the DiskUsageByTable stored procedure to access these numbers.

Index Usage

- Find out which indexes are being used and which are not.

```
SELECT o.name AS object_name, i.name AS index_name,
      i.type_desc, u.user_seeks, u.user_scans,
      u.user_lookups, u.user_updates
  FROM sys.indexes i
  JOIN sys.objects o  ON i.object_id = o.object_id
  LEFT JOIN sys.dm_db_index_usage_stats u ON i.object_id = u.object_id
        AND i.index_id = u.index_id
        AND u.database_id = DB_ID()
 WHERE o.type <> 'S' -- No system tables!
 ORDER BY (ISNULL(u.user_seeks, 0) + ISNULL(u.user_scans, 0) + ISNULL(u.user_lookups, 0) +
    ISNULL(u.user_updates, 0)), o.name, i.name
```

- If indexes are not being used, find out why, and if they are really needed. If they are not needed, then remove them.

Index Usage Terminology

- An Index Scan accesses all the rows in the index.
- An Index Seek uses selective rows in the index.

- The Seek is much quicker than the scan.

Query Tuning

- Avoid SELECT * FROM … instead select specific column names. Just ask for the columns you are looking for.

- Using LIKE clause
  - WHERE ____ LIKE ‘ste%’ uses indexes.
  - WHERE ____ LIKE ‘%son’ cannot use an index.

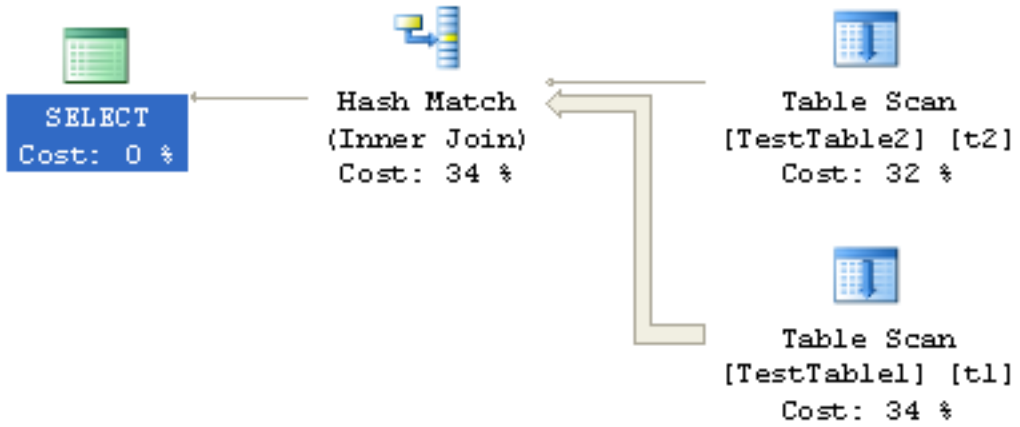
Functions and Calculations in the WHERE Clause

- WHERE age + 5 > 65
  - Instead use WHERE age > 65 – 5

- WHERE ISNULL(order_date, 'Jan 01,2003') > 'Jan 01, 2002 12:00:00 AM'
  - Instead use WHERE ord_date IS NOT NULL
    AND ord_date > 'Jan 01, 2002 12:00:00 AM'

Understand the Execution Plan

- When running queries through the SQL Server Management Studio, turn on “Include Actual Execution Plan” from the Query Menu.
- SQL Server 2008 will give recommendations on missing indexes.



Procedure Cache

- The procedure cache caches more than just procedures, it also caches parsed queries.
- Performance tuning the Procedure Cache reduces waste on the SQL Server.
- You don't have control over the size of the procedure cache, but you do have control over how it is used.
- Reuse in the procedure cache allows queries and procedures to run faster.
- Tuning is accomplished by a number of methods:
  - Using Parameterized Queries.
  - Removing temp tables from Procedures.
  - Implementing database Coding Standards

Finding the Contents of the Procedure Cache

- You can examine the queries in the procedure cache with the following query.

```
SELECT  qs.execution_count,
        st.text, total_elapsed_time
FROM    sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
WHERE st.encrypted = 0
ORDER BY st.text
```

- Scroll through results and find similar queries taking up space in the procedure cache.

Parameterization of Queries

- Allows for already parsed queries to be re-used.
- Replace hard coded values with parameters.

```
SQLCommand cmd = new SqlCommand("SELECT column1 from table_name
    where column2 = 'Customer Name'", conn);
reader = cmd.ExecuteReader();
```

- Replace with

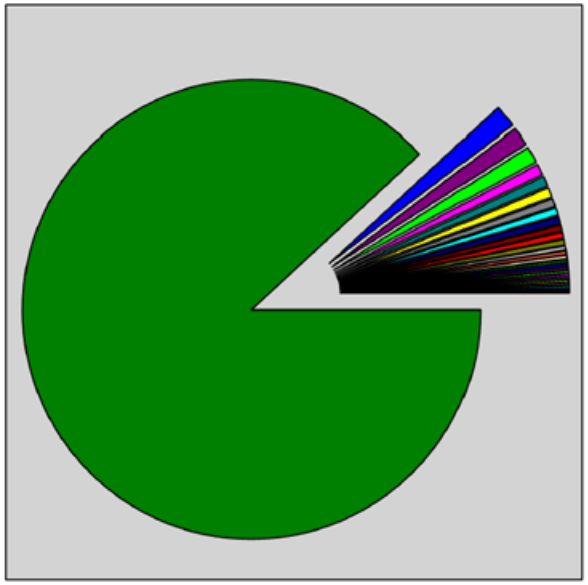
```
SQLCommand cmd = new SqlCommand("SELECT column1 from table_name
    where column2 = @PARAM1", conn);
SqlParameter param = new SqlParameter();
param.ParameterName = "@PARAM1";
param.Value = "Customer Name";
cmd.Parameters.Add(param);
reader = cmd.ExecuteReader();
```

Eliminating One Time Use Queries

- A One Time Use Query is a query that is compiled in procedure cache run once and never used again.
- If a query takes up 5mb of memory just for the parsing, this can add up quickly if there are dozens or hundreds of instances in memory.
- These can be reduced with parameterization.
- Use the stored proc FindOneTimeUseQueries

Understanding Table Sizes

- The table size is determined by the following:
  - Data in the table.
  - Clustered indexes on the table.
  - Non-Clustered indexes on the table.
- DiskUsageByTable stored procedure
- Don't let one table take over your whole database.
- Focus on the problems.



Server Best Practices

- Turn off any unneeded services.
  - FTP
  - Webserver
  - Others…
- “Maximize Throughput for network Applications”.
- Turn off any screensavers.

Summary

- There are many ways to performance tune your SQL Server. You can use any or all of the methods outlined in this presentation.
- Determine which method is best for your needs:
  - Tuning Queries
  - Tuning Indexes
  - Tuning the Procedure Cache
  - Analyzing Table and Index Sizes
  - Adding Hardware

Further Information

- Visit my website:
  - Download this presentation, stored procedures and queries:
  - http://SteveStedman.com

- Contact Me:
  - Steve@SteveStedman.com

Questions and Answers