Database Performance Review For [redacted]

Table of Contents

1 Executive Summary

All [redacted] data is stored on a single SQL Server database. The number of tables, stored procedures and other objects has grown to thousands. The database is performing at near peak loads and is a bottle neck in the process. Developers have tried to optimize their code by limiting database calls but some processes still take a long time, even minutes, to execute. The purpose of this project is to assess the database health and suggest how to improve it.

2 Scope of Review

An initial pass was made on the production database and we used monitoring to determine which SPs were taking a while to run. It was immediately evident that there was a lot that could be done to optimize the database.

SpiralEdge IT then provided us a list of SPs that are running slow. We were asked to optimize some code to prove we could do what we are recommending. Results of those optimization efforts are included in the attached documents.

3 Short Term Fixes

3.1 Incorrect indexing strategy

A maximum of 5 to 10 indexes should be created per table. Some of your tables have literally hundreds of indexes. An example is the PRODUCTS table over 120 indexes. By having so many indexes it is going to slow down database operations to that table significantly.

3.2 Incorrect or missing clustered indexes

A clustered index is a special kind of index that reorders the way records in a table are stored. Due to the nature of this index, only one is possible per table. If the clustered index is not created correctly it can lead to performance issues. Great care must be taken to ensure that your high volume tables have the correct clustered indexes. In many cases we looked at, a table did not have a clustered index. The counts of tables with missing clustered indexes are noted in Section 7.2.

3.3 Too many reads

By viewing the execution plan many of the stored procedures taking a long time to run is due to too many reads. In many cases cpu was not high and memory not high but each read is 8 kb and there are thousands of reads. By rewriting these SP it is possible to get the exact same results but with much fewer reads. Better indexing can also lower the number of reads.

3.4 Incorrect TempDB design.

In order for SQL Server to perform well, temp db has to be designed correctly. The number of files is a function of the number of CPU cores. Secondly the temp files should be of roughly the same size. In the production TempDB there are both too many files and the size of the files appears to be random.

3.5 Irrelevant data being returned

In some cases, too many rows are returned. In the file include in the attachment almost 20,000 rows were returned, when only 3 should have been returned. Irrelevant data is returned for many stored procedures and this adds to the database load unnecessarily.

3.6 Expensively written stored procedures and select statements.

Much of the database code is written in select statement format and, in our opinion, are not well written. Over and over again, we see the pattern of a string being constructed and then executed. While this technique can be applied in certain cases, it appears to be used as a first option. Because of the way the string is constructed, no execution plan is stored in memory and the stored procedure is compiled over and over – continuous compilation - each time it is run.

Some SPs are 100s of lines long. Rewriting the stored procedures would mean that first time runs might be slow but subsequent requests would be much, much faster. By using this string approach, seemingly across the board, the full power of SQL Server is not being utilized and the stored procedure would execute as a slower speed similar to if the statement was embedded in an ASP page.

The problems we are seeing with the database tables and coding is quite common. In our experience it is usually a function of the developers writing their own SQL queries. In a test environment or on a smaller subset of database, this code might run very quickly. But in a production environment this kind of coding is not optimal and cannot scale as web traffic increases.

3.7 Some SQL Server settings may not be set up correctly

Some server settings did not appear to be set up correctly. Additional discovery would be needed to delve into the details with the DBA to determine if there are valid reasons why they are set like this.

3.8 Backups

All production databases are running under Simple Recovery Mode. In the event of a database crash, all data that were generated after the completion of the database backup will be permanently lost.

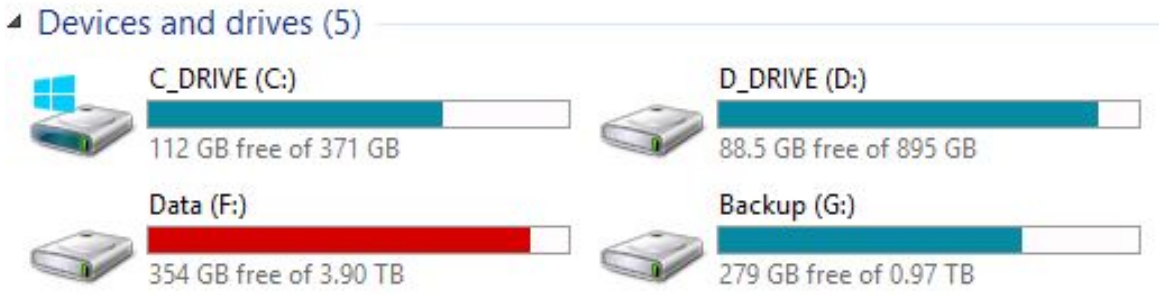
Recommendation: Change database Recovery Mode from Simple to Full for primary production databases and run log backups frequently between Full and Differential backups. This will prevent the chances of critical production data loss.

3.9 System Analysis

It was discovered in the process of analysis that CPU, Memory and Disk space are highly utilized, which results in performance bottlenecks on the server.

The recommended amount of free space should be around 25% of available drive space. However, in the case of production server. Couple of observations:

- Drive space for drive F: has critically low amount of free space.
- Drive D: has low amount of free space.



3.10 Proactive database monitoring

When it comes to monitoring, you want something constantly watching your database systems to provide useful information so you can take necessary action before any critical failure. We recommend Spotlight & Foglight which are two of the top monitoring tools available in the market. They are inexpensive but provide you with lot of information about your system. Spotlight does the monitoring whereas Foglight runs a very sophisticated analysis on the data to give you the various trends about your system. These systems add between 1% to 3% performance overhead to the servers which should be completely acceptable.

Medium to Long Term Recommendations

3.11 Re-design sections of the database

There are literally thousands of database objects and they appear to be random collections of things. Tables, indexes, views, etc. are being created seemingly on an ad-hoc basis vs. any kind of well architected plan. What we are seeing is not uncommon; it is usually the result of a small database being created followed by years and years of requests for new features and functionality which are added without re-architecting the database every so often. The end result of unchecked database design is high CPU and memory utilization.

Recommendation: Redesigning the entire database would be a gargantuan undertaking. What most companies do is start in one area and remove unused tables and objects to see what is actually needed.

3.12 Remove SQL Queries embedded in code

Reviewing the SQL logs we see a lot of calls that are coming from your front end web servers or other places. This code is also not cached on the SQL Server side. Moving the database code into SQL Server and optimizing would greatly help performance. It would also result in centralized code which is easier to maintain and is generally more secure.

Recommendation: Look to take code from the web pages, optimize and add to SQL Server.

3.13 Stage database and production database structures do not match

Many of the tables and stored procedures that exist in production are not in the stage environment. This suggests that code is being pushed directly to production and perhaps only tested on a DEV or QA instance.

Recommendation: Restore the latest full production backups on STAGE such that it mirrors production. Sensitive customer and credit card information can be scrambled. Then queries could be tested more thoroughly before being pushed live.

3.14 Disaster Recovery and Load Balancing

Production server is a stand-alone server and does not provide disaster recovery options. In an event of server failure, business loses its ability to deliver data to production sites and business units.

Recommendation: Consider adding database redundancy. There are various options available from Microsoft to safeguard against a disaster. Clustering is one solution where 2 servers run as active & passive. In case of a failure to the active node, the passive node becomes active thus providing fault tolerance. This is simple and cost effective however, this only works in a single geographical location; both servers need to be in the same data center.

A more modern and recommended approach uses what's called AlwaysOn Availability Groups. This feature is a high-availability and disaster-recovery solution. It maximizes

the availability of databases for an enterprise, supports a set of read-write primary databases and one to eight sets of corresponding secondary databases. Optionally, secondary databases can be made available for read-only access to support data warehousing, ad-hoc queries and/or some backup operations.

3.15 Revisit the process of database changes

An audit of the process in which changes are introduced to the production database is in order. Are the right checks and balances in place? This applies to new tables, new indexes, new SPs, new scheduled tasks, etc.

3.16 Add senior database engineer(s)

It appears that SpiralEdge may be stronger on the software developer side and weaker on the database side. It's also possible that there just aren't enough dedicated database people and, due to the lack of bandwidth, developers are able to create tables and make changes to the database.

The vast majority of DBAs and developers, even senior folks, are not be able to optimize and redesign your database effectively. A key hire would be for the IT group to add a seasoned SQL Server engineer who has experience with high traffic databases who can slowly start implementing changes as well as act as a gatekeeper for new changes.

4 Optimization Samples

4.1 Stored procedure rewrite sample

Based on the list presented to us, we chose to rewrite the following two stored procedures.

exec [dbo].**AQF_Get_Box_Suggestions_Logic_V1** @shimentID=N'7372939',@shipmenttype=N'CINN'

exec ***AQP_OrderVerify_Queue_SelectAll_test_CC*** @s_fromsite=N'',@ShippingMethod=N'',@OrderColumn=N'orderdate',@sortPath=N'ASC',@RowFrom=1, @RowTo=100

Optimized code is in the attached Excel document.

4.2 Clustered index optimization sample

One of the SP were running slow and it was determined that no code rewriting was necessary and the index had to be created. The clustered index code is also in the attached document.

5 Appendix

Below is the analysis of all objects in each database on the production server.

5.1 List of databases and objects:

1. Database – beta_[redacted]_com

    - Total number of tables – 1148
    - Total number of stored procedures – 3857
    - Total number of views - 338
    - About 300 tables have no clustered index.
    - Large number of indexes – 9804

2. Database – beta_[redacted]_com_extended

    - Total number of tables – 343
    - Total number of stored procedures – 1212
    - Total number of views - 29
    - About 45 tables have no clustered index.
    - Large number of indexes – 2944

3. Database – beta_[redacted]_com_extended

    - Total number of tables – 153
    - Total number of stored procedures – 15
    - Total number of views - 29
    - About 28 tables have no clustered index.
    - Large number of indexes – 840

4. Database – selocation

    - Total number of tables – 109
    - Total number of stored procedures – 6
    - Total number of views - 3
    - About 99 tables have no clustered index.
    - Large number of indexes – 410

5. Database – so-archived

    - Total number of tables – 268
    - Total number of stored procedures – 6
    - Total number of views - 0
    - About 168 tables have no clustered index.
    - Large number of indexes – 1695

6. Database – so-backup

    - Total number of tables – 1811
    - Total number of stored procedures – 14
    - Total number of views - 0
    - About 1781 tables have no clustered index.
    - Large number of indexes – 6165