Company 123 – SQL Server Review Detailed Report

- **Written By:** Eitan Blumin, Madeira Data Solutions
- **Reviewed By:** Evyatar Karni & Guy Glantser, Madeira Data Solutions
- **Created:** 10/12/2020
- **Last Updated:** 09/02/2021

1. About This Document

This document describes the Business Environment and Technological Environment of Company 123. It then details our findings and recommendations based on our SQL Server review.

If you are interested only in the bottom line, then you can read the Executive Summary. If you are interested in the details, then jump to the Findings section.

You can use the **Navigation Pane** in Microsoft Word to easily navigate the document. Go to the "View" menu, and in the "Show" section, you will find the "Navigation Pane" checkbox.

2. Executive Summary

The following sections describe the bottom-line in each one of the following business categories: performance, availability, security, and cost savings. A red color means the current situation is poor with high risk and should be addressed ASAP. A yellow color means the current situation is not optimal, but there's nothing urgent. A green color means the current situation is good.

Performance

The CRM analytical system has a very good baseline in terms of database design best practices. But there's still a lot of room for improvement.

Several "small" mistakes seem to have a large negative impact on performance and should be easily rectifiable.

Some "larger" issues would require significant redesign on the database structure (such as utilizing a separate "staging" database) but can provide a significant boost to performance and availability.

Availability

Currently, the main database (CRM_DB) is not included in an Availability Group, and as such is at risk of losing data in case of a disaster. There is an "alternative" solution in place implementing a pseudo-log-shipping mechanism to a remote environment, but it cannot be frequently utilized compared to Always On Availability Groups.

The main issue with adding CRM_DB to Always On is the rate of data modifications which causes an unacceptable overload on the AG synchronization.

By optimizing the data modification processes to "reduce their DML footprint", we can provide an acceptable behavior that would allow the CRM_DB database to be added to the AG.

Most of these recommendations bleed into the "Performance" topic.

Security

There are several high-risk factors to the SQL Server security in the CRM server. However, since the server is located in an isolated network, the impact of these risk factors is reduced.

Cost Savings

There are a few cost-savings opportunities found during the review.

The most notable of these is the fact that the secondary DR server is fully licensed, even though it doesn't need to be.

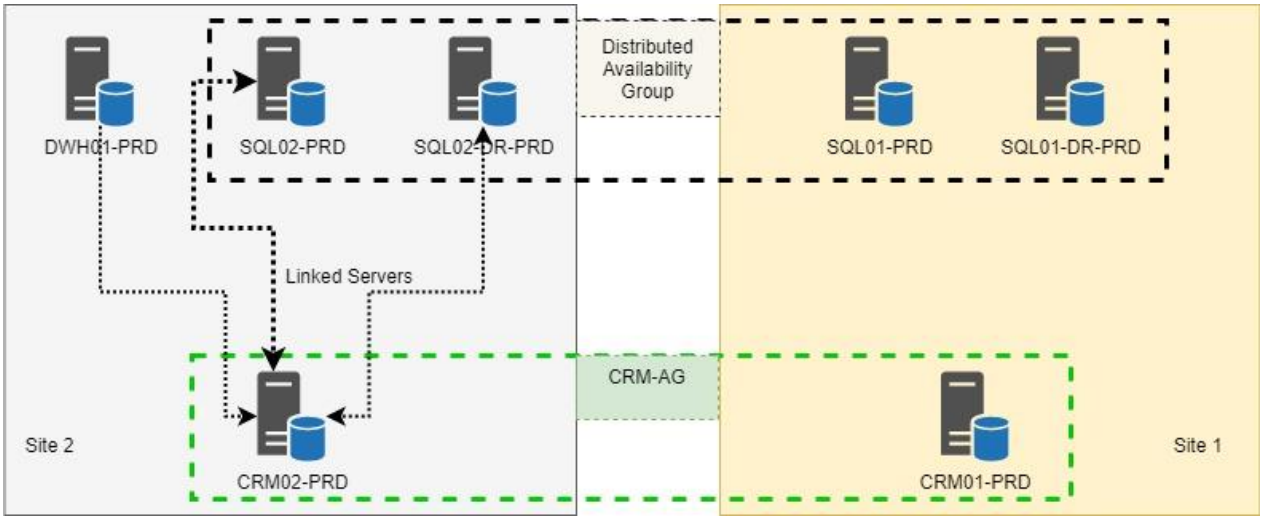
There is also a significant amount of data that could potentially be reduced or compressed, possibly leading to cost savings on the data disk(s).

3. Business Environment

The CRM server is used for various analytical operations and generation of “campaigns”. These operations are done manually by the CRM Analyst team via the built-in tools of SQL Server (SSMS, SSIS, etc.).

4. Technological Environment

Our focus is on the CRM database server and its integration with surrounding environments, so here is the relevant architecture:



The main server in question, CRM02-PRD, is located in a data center in Herzliya. It has an Availability Group set up with a replica server CRM01-PRD, located in New Jersey.
It communicates with other servers via linked servers:

- DWH01-PRD
- SQL02-PRD
- SQL02-DR-PRD

5. Review Scope

- We analyzed all aspects of the data environment, including performance, availability, security, and cost, with a focus on performance especially.
- Most of the findings are based on data collected between **11/10/2020** and **07/12/2020**.

6. Findings

Following is a thorough list of all of our findings. Each finding is categorized according to its business implications (performance, availability, security, and/or cost savings). The list is sorted according to the impact in descending order, and then according to the effort required to implement the recommendation in ascending order. So, if you are looking for quick wins, they are at the top of the list.

The impact of each finding corresponds to how much it affects the system. For example, a high-impact finding with an Availability business implication means that your system is at high risk of business continuity. You should apply the recommendations for high-impact findings ASAP.

The recommendation effort represents the time and effort needed to apply the recommendation. It corresponds to the complexity of the solution, as well as to the time it will take to implement the recommendation. For example, a recommendation with a low effort, which is related to a finding with a Performance business implication, means that we can solve the problem and improve performance relatively quickly.

We also present the risk associated with implementing each recommendation. A high-risk recommendation can negatively affect availability or performance and should be carefully tested and monitored.

The following table includes a summary of all the findings. You can click on each finding to jump to the detailed description.

Finding	Business Implications	Impact	Recommendation Effort	Recommendation Risk
Secondary Server is Fully Licensed	Cost Savings	High	Low	Low
Insufficient Database Integrity Check Configuration	Availability	High	Low	Low
Eligibilities Sync Performance Tuning	Performance	High	Medium	Low
CardHolderDetails Sync Performance Tuning	Performance	High	Medium	Low
ContactHistory Performance	Performance	High	Medium	Low
Campaign Analysis Performance Tuning	Performance	High	Medium	Low
Reduce Load on AG via External Staging Tables	Performance	High	Medium	Low
Backups and DB Files on the Same Physical Disk	Availability	High	Medium	Low
xp_cmdshell Enabled	Security	High	Medium	Low
SQL Memory Dump Files Found	Availability	High	Medium	Medium
Secondary Server is not Monitored	Availability Performance Security	Medium	Low	Low
Unused Indexes	Performance Cost Savings	Medium	Low	Low
Outdated Statistics	Performance	Medium	Low	Low
Remote DAC should be Enabled	Availability	Medium	Low	Low

Maintenance History Cleanup Jobs not Scheduled	Performance Availability	Medium	Low	Low
Intermittent Availability Group Disconnections	Availability	Medium	Medium	Low
Foreign Keys without Matching Indexes	Performance	Medium	Medium	Low
Tables with High Unused Space	Performance Cost Savings	Medium	Medium	Low
Tables with high Data Compression Savings	Performance Cost Savings	Medium	Medium	Low
Large Heaps (Tables without Clustered Indexes)	Performance	Medium	Medium	Medium
DB Files Located on OS Drive	Availability	Medium	Medium	Medium
Redundant Indexes	Performance Cost Savings	Low	Low	Low
Orphaned Database Users	Security Availability	Low	Low	Low
Job Failover Solution for HA/DR	Availability	Low	Low	Low
Tables To Be Deleted	Performance Cost Savings	Low	Low	Medium

Following is a detailed description of each finding and the corresponding recommendation.

a. Secondary Server is Fully Licensed

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Cost Savings	High	Low	Low

- Description

  On November 1st, 2019, Microsoft announced several High Availability and Disaster Recovery benefits related to SQL Server licensing. One of them is the ability to install and run passive SQL Server instances in a separate operating system environment for disaster recovery in anticipation of a failover event.

  The meaning of a passive SQL Server instance is that the instance is only synchronized with the primary replica in anticipation of a failover event, but it is not used directly for any other purposes.

  This benefit is only available to software assurance customers of SQL Server.

  Currently, the secondary server is fully licensed and doesn't leverage this benefit.

- Recommendation
  - Verify the existence of SQL Server software assurance.

- If available, contact Microsoft or your reseller to apply this feature and update payment agreements.

b. Insufficient Database Integrity Check Configuration

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability	High	Low	Low

- Description

  Several databases were found to be lacking integrity checks for a long time. Performing regular integrity checks is essential for detection and pre-treatment of corruption and data loss, to avoid malfunction and possible loss of availability.

- Recommendation
  - Install the **latest** version of the Ola Hallengren maintenance solution scripts as further detailed in the section Outdated Statistics.
  - Make sure that your existing SQL Server jobs "**DatabaseIntegrityCheck - SYSTEM_DATABASES**" and "**DatabaseIntegrityCheck - USER_DATABASES**" are covering all the databases in the server.
  - For large databases, consider using the **PHYSICAL_ONLY** option to minimize check duration.
  - For very large databases, you should consider "splitting" the integrity checks by gradually checking a different "bucket" of tables each weekday. Such an implementation is available in the Tiger Toolbox open-source repository from Microsoft.

c. Eligibilities Sync Performance Tuning

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	High	Medium	Low

- Description

  Company 123's main concern when requesting this SQL Server Review was to improve their main BI/DWH synchronization and calculation processes. The stored procedure **BI_Eligibilities** is one of them, responsible for synchronizing the **Eligibilities** table.

  In terms of **Duration**:
  - The longest-running parts in this procedure seem to be the ones querying the large datasets from the **SQL-REPLICA** linked server.

  In terms of **Writes**:
  - The part incurring the most disk IO writes is the last **MERGE** statement at the very end of the procedure.

- Commands that incur too much data writes are the main cause for why the Availability Group may not be able to replicate the data in time to the secondary server, and to cause the transaction log file to become bloated (which will also increase the transaction log backup size and duration).

- Recommendation
  - To overcome the network bandwidth limitations, if and wherever possible, try to **reduce the amount of data queried from the source** linked server. The best thing would be to have some kind of indicator of when was the last time the synchronization was performed, and get only the data modified since that time.
  - Even if this indication is not possible for all tables involved, **at least do this for as many of the subset tables as you can**, to minimize the overall dataset size.
  - To **reduce the write impact** of the **MERGE** command, follow the guidelines below:
    - In the **WHEN MATCHED THEN** clause, add conditions to verify that indeed at least one of the columns is different so that only records that were modified would be affected by the update.
    - See **Appendix P1** for an example code snippet utilizing the **EXISTS/EXCEPT method**, as explained here.
    - Performance-wise, comparing the values of a few tens of columns would still be preferable to updating the entire table (update operations still happen in the SQL engine even if all new values are identical to the old ones).
    - To reduce the blocking overhead of the MERGE command, consider "splitting" it into several phases:
      - Find and mark only the relevant records in the source (temp table) that need to be updated because at least one of the columns changed. Use the EXISTS/EXCEPT method, as explained here. This phase can be done using a NOLOCK hint on the destination table.
      - Update the records in the destination table (only those that were marked to be updated by the previous phase).
      - Insert all missing records that don't exist yet.
      - See **Appendix P2** for an example Proof-Of-Concept script demonstrating and measuring the different methodologies.

d. CardHolderDetails Sync Performance Tuning

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	High	Medium	Low

- Description

Company 123's main concern when requesting this SQL Server Review was to improve their main BI/DWH synchronization and calculation processes.

The stored procedure **Incremental_CardHolderDetails_Summary** is one of them, responsible for synchronizing the **CardHolderDetails** table. Indeed based on long-term performance tuning data in SentryOne, this is by far the heaviest procedure, with the longest total duration, highest CPU, reads, and writes utilization. See **Appendix P0** for more details.

In terms of **Duration**:

- The longest-running part in this procedure seems to be the one joining between the various sets of temporary tables (LEFT JOIN).

In terms of **Writes**:

- The part incurring the most disk IO writes, are the last **TRUNCATE** and **INSERT** statements at the very end of the procedure.
- Commands that incur too much data writes are the main cause for why the Availability Group may not be able to replicate the data in time to the secondary server, and to cause the transaction log file to become bloated (which will also increase the transaction log backup size and duration).

- Recommendation
  - If and wherever possible, try to **reduce the amount of data queried from the source** linked server. The best thing would be to have some kind of indicator of when was the last time the synchronization was performed, and get only the data modified since that time.
  - Even if this indication is not possible for all tables involved, **at least do this for as many of the subset tables as you can**, to minimize the overall dataset size.
  - To improve the performance of the main calculation query performing all the **LEFT JOINs** with the temporary tables, consider trying to **split some or all** of the temporary tables to separate UPDATE commands, filling up data as necessary. This would especially be useful for temporary tables that hold data for a **significant minority** of the cardholders, and thus the performance impact would be only for that minority instead of all the cardholders.
  - To **reduce the write impact** of the **TRUNCATE** and **INSERT** commands, follow the guidelines below:
    - Unless the vast majority of the table needs to be updated, avoid using the TRUNCATE/INSERT methodology. Replace it with better-focused **UPDATE** or **MERGE** commands that only update the data that needs to be updated.
    - Avoid using a staging table in the same database. Use a #temporary table instead, or a staging table in a local database (not involved in an Availability Group).
    - Performance-wise, comparing the values of a few tens of columns would still be preferable to updating the entire table (update operations still happen in the SQL engine even if all new values are identical to the old ones).
    - To reduce the blocking overhead of a MERGE command, try "**splitting**" it into several phases:

- Find and mark only the relevant records in the source (temp table) that need to be updated because at least one of the columns changed. Use the **EXISTS/EXCEPT method**, as explained here. This phase can be done using a NOLOCK hint on the destination table.
- Update the records in the destination table (only those that were marked to be updated by the previous phase).
- Insert all missing records that don't exist yet.
- See **Appendix P2** for an example Proof-Of-Concept script demonstrating and measuring the different methodologies.

- To reduce the overhead on your destination table even further, consider implementing the synchronization logic in "**chunks**":
  - Assign an **identity** column to the **source temporary table**.
  - Implement a **WHILE** loop that performs the existence check and/or update for a **subset** of the source temporary table, using the identity column values as range indicators. For example, the first iteration would check for source records with IDs 1 to 100, the second iteration would check for records with IDs 101 to 200, etc.
  - Reduce the overhead even further by using a **WAITFOR DELAY** command after each iteration (for example, to wait for a duration of half a second before continuing to the next iteration).
  - See **Appendix P3** for an example Proof-Of-Concept script demonstrating the methodology.

- The main temporary table should be properly **indexed** to optimize the relevant queries (JOINing with the target table during updates, checking WHERE NOT IN/EXISTS during insertions).

e. ContactHistory Performance

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	High	Medium	Low

- Description

  The **ContactHistory** table is a central table used for a wide variety of use cases by the CRM analytics team. It is a very large table (~350 million rows, ~500 GB).

  Unfortunately, there's no particular kind of query or operation that can be specifically tuned for this table. Most operations query vast amounts of data from this table based on one or more "Campaigns", which is connected to the ContactHistory table via the **BulkDistribution** table via the **BulkID** column. For example:

```
SELECT …
FROM BulkDistribution AS bd
INNER JOIN ContactHistory AS ch ON bd.BulkId = ch.BulkId
WHERE bd.CampaignID IN (123, 456, 789);
```

- Recommendations
  - There are a few things we could try to optimize the performance of this table, although none of these methods is guaranteed to solve all performance issues on its own.
  - **Enable Data Compression on the ContactHistory table** to reduce its performance impact on the data disk. Please see section Tables with High Data Compression Savings and **Appendix C** for more details.
  - Reduce the table's size and DML overhead by **removing unused and redundant indexes**. See sections Unused Indexes and Redundant Indexes for more details.
  - Copy the **CampaignId** column to the **ContactHistory** table as well. This would make filtering by CampaignId more efficient by accessing the ContactHistory table directly instead of relying on the join with the **BulkDistribution** table.
  - **Index Optimization** is recommended to cover the most frequent use-cases of this table. For example, having an index on **BulkID, SendDate**, and **INCLUDE** on **ContactStatus** and **CardHolderId**. Also, it's recommended to utilize **Filtered Indexes** to better support, especially frequent use-cases. For example:

```
CREATE NONCLUSTERED INDEX [IX_ContactHistory_BulkID_Incl] ON
[dbo].[ContactHistory] ([BulkID], [SendDate])
INCLUDE ([ContactStatus], [CardholderId])
WHERE [CardholderId] != '12345' AND [ContactStatus] != 0;
```

    NOTE: Your queries would have to logically fit the index filters for the SQL engine to be able to use it.
  - When designing your views and queries, remember to maintain "**SARGEability**", by "isolating" table columns to one side of a predicate's "equation". As such, avoid using **CASE** expressions that later would be used in **WHERE** clauses. For example, see **Appendix P4** for an execution plan where the heaviest statement is querying from the **VW_Contacts** view and filtering on IsTest = 0, which translates to:

```
CASE WHEN [dbo].[ContactHistory].[CardholderId] = '12345' THEN 1 ELSE
[dbo].[DistributionBulks].[IsTest] END = 0
```

    The entire CASE expression is on one side of the predicate "equation", and therefore it's not SARGEable. The preferred alternative is to change the CASE expression into a more straight-forward AND expression like so:

```
[dbo].[ContactHistory].[CardholderId] != '12345'
AND [dbo].[DistributionBulks].[IsTest] = 0
```

You may have to make changes in the definition of your view or even create an entirely new view(s) to support this change (for example, by creating a view specifically dedicated to non-test data). Also, matching **filtered indexes** could help here as well.

- This private issue of **CardHolderId = '12345',** which represents a special case, badly affects various design choices down the line (such as the creation of computed columns, filtered indexes, etc.). All of these bad design choices only work to further "cement" the special case instead of properly resolving it. It's best to receive the difficult choice and properly resolve this case. By cleaning up the data, and/or by making proper schema changes.
- Create missing **Foreign Keys** between the tables *SegmentChannels, FlowOrders,* and *Segments, Flows, CampaignTracks*. Such foreign keys can assist in **JOIN Elimination** while querying from the relevant view(s) without returning data specifically from those tables, and thus improve performance.
- You may want to consider alternative **NoSQL** platforms that could better fit your unpredictable use cases for this table, such as **Columnar Databases** or **Document Databases**. For example, **Azure Data Explorer (Kusto), Azure CosmosDB, Cassandra, HBase, MongoDB, Couchbase**, and so on. One of these may give you a better solution than SQL Server.

f. Campaign Analysis Performance Tuning

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	High	Medium	Low

- Description

  This section encompasses the various analytic jobs and stored procedures executed by the CRM analytics team, which implement some kind of analytical process on various campaign-related tables.

  The following queries below were taken as a representative sample based on long-term SentryOne performance data. There are different issues and recommendations per each.

- Recommendations

  - **Appendix CM1: Campaign Contact History and Responses**
    - The heaviest command in this script appears to be the creation of the *#VW_CampaignStructure* temp table based on a query from *VW_CampaignStructure*, after which several other temporary tables are created as intermediary phases. What we should try here is to reduce the overhead generated, by **removing unnecessary "lookup" columns**, such as *CampaignName, FlowName, SegmentName, ChannelTypeDesc*. These columns are not required for the script's aggregations and calculations. They're only needed in the very last output. As such, their retrieval **should be deferred to as later step as possible in the script**. Otherwise, you're wasting

valuable memory and disk resources every time you query them, store them in a temporary table, query them again and store them again in another temporary table.

- **Appendix CM2a to CM2e: Process_CampaignsDistribution**
  - A significant portion of the top heaviest queries in the database seems to have been caused by executions of the stored procedure **Process_CampaignsDistribution**, in which the heaviest part is the **trigger on ContactsEligibilityStatusReasons**, which was incorrectly updating the **UpdateDate** column based on **CampaignID** of the affected data. This appears to have been a human error due to incorrect copy-pasting from somewhere else.
  - You should improve SARGEability when filtering on date-time ranges, such as the **RunDate** column. In other words, this:

    ```
    DATEDIFF (D, RunDate, GETDATE ()) <= 1.5
    ```

    should be changed into this:

    ```
    RunDate >= GETDATE () - 1.5
    ```

    Once the search predicate is filtering on a **column** instead of an **expression**, this should allow the SQL Optimizer to make good use of column statistics, and if/when it's indexed, then it would be able to use such an index. In turn, this would make for better row estimations and better execution plans.

g. Reduce Load on AG via External Staging Tables

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	High	Medium	Low

- Description

  The CRM team complained about not being able to add the CRM_DB database into the Availability Group because otherwise, the AG cannot keep up with the heavy load caused by the massive data changes performed during analytical processes and synchronizations.

- Recommendation
  - In addition to all of the other recommendations in this document, meant for "reducing the DML footprint" of your analytical and synchronization operations, you should also consider **separating your staging tables and operations to an external database**. What this means is that, instead of saving all of your "intermediary" data in the same CRM_DB database, you would do so in a separate database ("**Staging_DB**" for example) which would NOT be included in an AG. The only data you would have remained in the CRM_DB database would be just the "**final result**" data.

- This would mean, of course, that in the event of a fail-over, the intermediary staging data would not be accessible to the secondary node, and any ongoing analytical/synchronization processes would have to be re-calculated from scratch.
- Considering the HA/DR benefit that this is expected to provide, the risk of having to re-calculate a process from scratch in the rare event of failover should be acceptable here.

h. Backups and DB Files on the Same Physical Disk

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability	High	Medium	Low

- Description

  2,217 backup files along with 13 database files were found on drive G.

  Having both backup files and database files on the same drive exposes the company to permanent data loss in case of hardware failure (such as a damaged disk).

- Recommendation
  - Either change the backup location policy to a different drive or move the database files (the latter option may require downtime maintenance).
  - Alternatively, you could also periodically copy/move the backup files to another server, thus ensuring data protection and increase recovery options.
  - You may use this script to find all backup files and database files located on the same drive: https://github.com/MadeiraData/MadeiraToolbox/blob/master/Best%20Practices%20Checklists/Backup%20and%20DB%20files%20on%20same%20physical%20volume.sql

i. xp_cmdshell Enabled

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Security	High	Medium	Low

- Description

  xp_cmdshell is a SQL Server configuration option allowing execution of cmd commands for external processing using SQL Server extended stored procedure. As it is useful for scripts and general programming, this configuration option is disabled by default, due to the security risks it may impose – malicious users could use it to elevate their privileges, affect system configurations, and even reach additional servers in the network.

- Recommendation
  - Find all procedures and processes which require the xp_cmdshell configuration option enabled.

- Evaluate whether a permanent activation of this option is necessary. Consider modifying the above procedures and processes found to enable xp_cmdshell only for the duration of the actual task – or better yet, look into other alternatives to achieve the same tasks more securely, such as CLR or SSIS.

j. SQL Memory Dump Files Found

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability	High	High	Medium

- Description

  Two SQL memory dump files were created on 17/05/2020.

  When a severe error occurs, SQL Server would sometimes automatically create a memory dump file (.**mdmp**) which contains a snapshot of the SQL Server memory state (containing thread call-stacks, CPU register states, and modules loaded).

  The 2 dump files created on 17/05/2020 happened with the current SQL version installed (14.0.3223.3).

  See **Appendix E** for a summary of these memory dump events.

  Also, an additional memory dump file was created on the **Secondary** server on 07/09/2020.

- Recommendation
  - Your current SQL Server version (14.0.3223.3) is very far behind (August 2019). It may very well be possible that the issue causing these crashes was already resolved in a later cumulative update.
  - Without being on the latest SQL update, you will not be able to use Microsoft Support for assistance with these crashes.
  - Please update your SQL Server version to the latest cumulative update as soon as you can. More details here: https://sqlserverbuilds.blogspot.com/#sql2017x.

k. Secondary Server is not Monitored

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability Performance Security	Medium	Low	Low

- Description

  While the primary server is being monitored using the SQL Sentry platform, the secondary server isn't. It is important to monitor the secondary server as well, as it can also cause availability, performance, and security issues.

Also, if there is a failover, and the secondary server becomes the primary replica, then the (new) primary server will not be monitored. Furthermore, a non-monitored server may have unnoticed faults which may only be revealed when it's already too late and it has become primary and might influence the server's availability during critical events.

- Recommendation

  We recommend purchasing a SentryOne monitor license for the secondary server as well and monitor both servers at all times. In addition to data collection, the platform also has very rich alerting capabilities, and we recommend leveraging them and implement useful alerts.

  We recommend purchasing one of our DBSmart plans, which also includes discounted licenses of SQL Sentry as well as the implementation of alerts and ongoing maintenance and support. It's also important to note that **SentryOne and DBSmart monitoring costs for secondary DR (passive) servers are at a half-price discount**.

I. Unused Indexes

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance Cost Savings	Medium	Low	Low

- Description

  On the primary server, a few unused indexes were found.

  Those indexes were probably created for past usage, but now either the data has changed or other more efficient indexes are being used instead.

  Although not used, in addition to unnecessary storage occupancy, they are being updated in each maintenance and DML operation, making it longer than needed.

- Recommendation
  - Check if additional information is available for the following indexes:

DB Name	Schema	Table Name	Index Name
CRM_DB	dbo	ContactHistory	ix_campaignname_campaigndate
Test	ReplicaData	PartnerList	IX_PartnerID_Incl
CRM_DB	dbo	DistributionBulks	IX_DistributionBulks_CampaignName
DBA_Local	dbo	JobOwnershipLog	IX_JobOwnershipLog_EventDate

  - If possible, create a dropping plan for the relevant indexes, preferably using long time intervals between them to allow beneficial rollback in case of need.

m. Outdated Statistics

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	Medium	Low	Low

- Description

  There are many outdated statistics in the **CRM_DB** database, which were either never updated, or updated as far back as 3 years ago or more.

  Despite the stale state of their statistics, these tables are still operational and have high row modification counter values, which indicate the number of rows changed since the last statistics update.

  Outdated statistics may have a severe impact on the performance of queries.

- Recommendation
  - Download and install the **latest** Ola Hallengren maintenance solution scripts.
  - Set up and schedule jobs to periodically update the statistics in the database(s).
  - It's recommended to have a statistics update as an additional step right after regular index optimization (rebuild/defrag) in the same job.
  - Example command to execute **IndexOptimize** to update all modified statistics (without rebuilding/defragmenting indexes):

```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationLow = NULL,
@FragmentationMedium = NULL,
@FragmentationHigh = NULL,
@UpdateStatistics = 'ALL',
@OnlyModifiedStatistics = 'Y',
@MaxDOP = 1,
@LogToTable = 'Y';
```

  - Please see **Appendix F** for more details about which statistics were found to be outdated during the review.

n. Remote DAC should be Enabled

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability	Medium	Low	Low

- Description

SQL Server provides a special diagnostic connection (**dedicated administrator connection**) for administrators when standard connections to the server are not possible. This diagnostic connection allows an administrator to access SQL Server to execute diagnostic queries and troubleshoot problems even when SQL Server is not responding to standard connection requests.
The configuration in question ("**remote admin connections**") determines whether the SQL Server would allow such connections to be made **from outside the instance**. If it's turned off, then DAC connections can only be made to "**localhost**".

- Recommendation
  - Rule of thumb: in clustered environments, the setting should be enabled. Otherwise, it should remain disabled.
  - However, in most cases, the customer would prioritize their server availability rather than its security. Therefore, in the vast majority of cases, **this setting should always be enabled**.
  - The setting can be enabled by running the following script:

```
sp_configure 'remote admin connections', 1;
GO
RECONFIGURE;
GO
```

o. Maintenance History Cleanup Jobs not Scheduled

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance Availability	Medium	Low	Low

- Description

  We can see that the Ola Hallengren maintenance solution is already installed on your servers. However, some of the related history cleanup jobs are not scheduled.
  This oversight may cause some “bloating” of the job history data in MSDB as well as maintenance output files in your SQL Server’s LOG directory.
  That, in turn, can make it difficult to troubleshoot job execution history (because it would take a long time to open the relevant tables and/or directories), not to mention the unnecessarily wasted disk space.

- Recommendation
  - Add a schedule to the relevant jobs that are missing a schedule:
    - On the **Primary** server:
      - **Output File Cleanup**
      - **sp_delete_backuphistory**
    - On the **Secondary** server:

- **sp_delete_backuphistory**
- **sp_purge_jobhistory**

p. Intermittent Availability Group Disconnections

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	Medium	Medium	Low

- Description

  Based on the SQL Server Error Log, and the AlwaysOn_Health extended events session, there are intermittent disconnection events between the Availability Group replicas.

  Example error message in the logs:

```
A connection timeout has occurred on a previously established connection to availability replica 'CRM01-PRD' with id [FC395142-21E0-402E-91B7-D6D002D9872F].  Either a networking or a firewall issue exists or the availability replica has transitioned to the resolving role.
```

  These errors could also be a cause for **intermittent job failures**.

  The errors seem to indicate some kind of latency issue between the replicas which may have something to do with the network. But the exact root cause is currently unclear.

- Recommendation
  - Do whatever can be done to reduce the bandwidth demands between the servers, by minimizing the amount and/or rate of data modifications on the primary server.
  - Troubleshoot possible network latency issues. Please check the network bandwidth between the two data centers, and see if anything can be done to increase it.
  - If the latency issues cannot be resolved, consider making timeout threshold settings more lenient (the current **Session Timeout** setting for the **WSFC** is **10 seconds.** You could increase it to something like 40 seconds, for example).

q. Foreign Keys without Matching Indexes

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	Medium	Medium	Low

- Description

  Multiple tables were found to have un-indexed foreign key columns.

  These tables may have a significant performance impact when the data in the “parent” table of the foreign key is deleted or joined with its “child” records in a query.

- Recommendation

- Consider creating indexes for these tables, as detailed in the attached Excel file (see **Appendix B**).
- Note that some of these foreign keys may not necessarily require an index if we're to assume that the parent table always remains unchanged. Please review each recommendation accordingly before applying any remediation script.

r. Tables with High Unused Space Percentage

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance Cost Savings	Medium	Medium	Low

- Description

  There are a few tables, specifically in the **CRM_DB** and **Optimus** databases, that have high unused space percentages. This can happen when a table has a high deletion/update rate, and/or when these tables are heaps (without a clustered index), and/or when the FILLFACTOR setting is too low.

- Recommendation
  - Consider creating a clustered index for the tables which are heaps.
  - Review the FILLFACTOR setting on these tables/indexes and if it's not 100, consider changing it.
  - Consider rebuilding the indexes of the tables to reclaim the unused space.
  - Review the usage methodology of these tables, and see if it can be improved. For example: Replace DELETE+INSERT logic with a simpler and better-focused UPDATE/MERGE logic.
  - Consider implementing Table Partitioning for the tables that are periodically deleted based on time.

s. Tables with High Data Compression Savings

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance Cost Savings	Medium	Medium	Low

- Description:

  Several tables in the CRM_DB database may significantly benefit from Data Compression.

  We've performed a data compression savings estimation check and found that as much as 144 GB could potentially be saved in disk space utilization if we are to apply compression in the right places. See **Appendix C** for more details.

  It should also be noted that these recommendations are based on the **current** usage statistics in your database. If you're able to improve your database operations by minimizing the rate at which

data is updated or deleted, then even more opportunities for data compression could potentially be found.

In general, data compression in SQL Server can improve not only disk space utilization but also disk IO performance and memory buffer utilization. This is because data compression is retained for the pages in the memory buffer as well, and the more you can fit in a data page – the fewer operations would be needed for retrieving the same amount of data (both for disk IO and memory buffer).

The "price" to pay for data compression comes in the form of somewhat increased CPU utilization since SQL Server has to automatically compress and decompress each data page that it accesses. The rate of this CPU utilization increase depends mostly on the update rate of compressed data, which is why it's important to carefully analyze the compression feasibility of each table based on its usage stats (which is exactly what was done as part of this check).

More details here.

- Recommendation:
  - Rebuild the relevant indexes with data compression, as detailed in **Appendix D**.
  - If the tables in question are too big, and/or your maintenance windows are too short, you may run the remediation script in stages, limiting its execution to specific days or times of day, until all relevant indexes are compressed as needed.
  - Note that this check was only performed on the **CRM_DB** database. If you wish to perform the same check for other databases as well, you may use the following T-SQL script: https://github.com/MadeiraData/MadeiraToolbox/blob/master/Utility%20Scripts/ultimate_compression_savings_estimation_whole_database.sql.

t. Large Heaps (Tables without Clustered Indexes)

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance	Medium	Medium	Medium

- Description

  There is a large number of large "Heap" tables across multiple databases.

  Heap tables are generally bad for performance and maintenance.

  For more info please refer to the following:

  https://eitanblumin.com/2019/12/30/resolving-tables-without-clustered-indexes-heaps/.

- Recommendation
  - Consider creating clustered indexes for the heap tables, as detailed in the attached Excel file (see **Appendix A**).
  - The recommendations in the appendix are best-guess only, based on each table's structure and usage statistics. Please review and verify each recommendation before applying.
  - Both remediation and rollback scripts are provided in the appendix, for your convenience.

- Most of the heap tables in **CRM_Workspace** could be dropped altogether (which I'm guessing due to the word "Test", "Temp", "Testing", or a person's name in the table names).

u. DB Files Located on OS Drive

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability	Medium	Medium	Medium

- Description

  The system database MSDB is located on drive C which also contains the operating system.
  Placing database files on the system volume puts the operating system in danger in case of DB overgrowth – which may result in server shutdown and difficulties to recover.

- Recommendation

  Consider moving MSDB to a different disk volume (will involve downtime maintenance).
  Click here for more details.

v. Redundant Indexes

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance Cost Savings	Medium	Low	Low

- Description

  In the **CRM_DB** database, a few redundant indexes were found.
  These are either duplicate indexes or indexes with key columns contained within the key columns of other indexes.
  Every seek operation on these redundant indexes can also be handled by one or more other indexes, which is what makes them redundant. In addition to unnecessary storage occupancy, they are being updated in each maintenance performed, making it a bit longer than needed.

- Recommendation
  - Check if further information is available for the following indexes:

Schema	Table Name	Redundant Index	Containing Index
dbo	NotAllowedContact	ID	CardHolderId
dbo	CampaignTestLog	CycleID	LanguageID
dbo	ContactHistory	ix_CardholderID	IX_ContactHistory_CardholderId

- If possible, create a dropping plan for the relevant indexes, preferably using long time intervals between them to allow beneficial rollback in case of need.

w. Orphaned Database Users

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Security Availability	Low	Low	Low

- Description

  "Orphaned Database Users" happen when a Database User is no longer associated with its relevant Server Login. Users are mapped to their Logins by their **SID**, not by their names. And every time you create a new SQL Login, it receives a new, random **SID**.

  Orphaned users often happen when the Server Login is deleted (even if it's recreated later), or when the database is moved or restored to a different SQL Server. You can find some more info on it in this article from Microsoft.

- Recommendation
  - Use the T-SQL script provided in this link to automatically detect all orphaned database users in the server, and generate remediation commands for them, based on the following use cases:
    - If a SQL login already exists with the same name, modify the database user to be linked to that login.
    - Otherwise, drop the database user.
    - If the orphaned user is the owner of database schemas and needs to be dropped, then a script is also generated to first transfer the ownership of its schemas to [dbo].
  - The best solution for this problem is to have consistent SIDs to your Logins across all your SQL Servers. So that even when a database is moved/replicated to a different server, it could still use the same SID that it was created for. And also, when you recreate a previously deleted Login, you'd need to create it with the same SID that it originally had.
  - You may use the "**sp_help_revlogin**" stored procedure published by Microsoft to generate CREATE LOGIN scripts that maintain the original login SIDs. One drawback of this procedure, though, is that it only provides the creation script for the Login itself, but not for its permissions, roles, etc.
  - If you ever need to migrate Logins from one SQL Server to another, including their roles and permissions and such, I recommend the easy-to-use Powershell library "**dbatools**" which contains the cmdlet **Copy-DbaLogin**. More info here.

x. Job Failover Solution for HA/DR

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Availability	Low	Low	Low

- Description

  Currently, job failover for the Availability Group is implemented using a preliminary step in each job that checks whether the local instance is the primary. If it's not, then an error is raised and the job completes with success. This is an inefficient methodology and is difficult to maintain.

  It fills up the MSDB database with useless job history "junk" and makes it difficult to troubleshoot actual execution failures of these jobs.

- Recommendation
  - We recommend using the "Master Control Job" methodology instead, where a single job controls the enabled/disabled status of all HADR-dependent jobs based on HADR role change events.
  - The following open-source solution was developed by Madeira Data Solutions and is publicly available for free. It is very robust and easy to set up: https://git.madeiradata.com/mssql-jobs-hadr/.

y. Tables To Be Deleted

Business Implications	Impact	Recommendation Effort	Recommendation Risk
Performance Cost Savings	Low	Low	Medium

- Description

  The "**OldObjects**" database contains many tables with "**_ToBeDeleted_**" in their names, specifying a date, which is, I assume, the date at which they can be deleted. All of these dates are far in the past, and yet these tables still exist. Some of them are very large and unnecessarily take up disk space.

- Recommendation
  - Drop these tables and free up space.

7. Appendix

The following additional files are provided alongside this document:

- Appendix_A_HeapTables.xlsx
- Appendix_B_ForeignKeysUnIndexed.xlsx
- Appendix_C_DataCompressionSavingsDetail.xlsx

- Appendix_CM1_CampaignContactHistoryAndResponses.pesession
- Appendix_CM2a_CM_Process_CampaignsDistribution_2.pesession
- Appendix_CM2b_CM_Process_CampaignsDistribution_3.pesession
- Appendix_CM2c_CM_Process_CampaignsDistribution_3_actual.pesession
- Appendix_CM2d_CM_Process_CampaignsDistribution_4.pesession
- Appendix_CM2e_CM_Process_CampaignDistribution.pesession
- Appendix_D_DataCompressionApply.sql
- Appendix_E_SQLDump_Summary.txt
- Appendix_F_Outdated_Statistics.xlsx
- Appendix_P0_TopSQL_Summary.xlsx
- Appendix_P1_WHEN_MATCHED_AND.sql
- Appendix_P2_MERGE_Variants_POC.sql
- Appendix_P3_UpdateInChunks_POC.sql
- Appendix_P4_contact_history_ExecPlan.sqlplan