

**Data Quality Services Performance Best Practices**

SQL Server Technical Article

**Author:** Ittai Gilat

**Contributors and Technical Reviewers:**

Jason Howell

Elad Ziklik

Omer Boker

David Faibish

Kumar Vivek

**Published:** April 2012

**Applies to:** SQL Server 2012 with Cumulative Update 1, Data Quality Services (DQS)

**Summary:** This article details high-level performance numbers expected, and a set of best practices on getting optimal performance when using DQS in SQL Server 2012.

Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft & SQL Server are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Data Quality Services Performance Best Practices

[Overview 2](#_Toc322090981)

[Expected Performance 2](#_Toc322090982)

[Hardware and Setup Considerations 3](#_Toc322090983)

[Recommended Server Hardware 3](#_Toc322090984)

[Processors Considerations 3](#_Toc322090985)

[Memory Considerations 4](#_Toc322090986)

[Disks Considerations 4](#_Toc322090987)

[Network Considerations 5](#_Toc322090988)

[Working Efficiently with DQS 5](#_Toc322090989)

[Understanding DQS Performance 5](#_Toc322090990)

[The Big Picture - DQS Data Quality Project Lifecycle 6](#_Toc322090991)

[Carefully Plan the Knowledge Structure 9](#_Toc322090992)

[Knowledge Acquisition - Acquire Knowledge a Chunk at a Time 10](#_Toc322090993)

[Interactive Cleansing - Enrich Knowledge with Common Errors 11](#_Toc322090994)

[Finally - Cleansing (Batch) and Matching 13](#_Toc322090995)

[DQS Scenarios 13](#_Toc322090996)

[Data size impact 13](#_Toc322090997)

[Knowledge Discovery Activity 14](#_Toc322090998)

[Cleansing Data Project Activity 14](#_Toc322090999)

[Matching Data Project Activity 14](#_Toc322091000)

[Cleansing with SSIS 17](#_Toc322091001)

[Advanced DQS Domain Features 18](#_Toc322091002)

[Managing DQKBs and Projects 20](#_Toc322091003)

[Working with Data Quality Client 20](#_Toc322091004)

# Overview

SQL Server Data Quality Services (DQS) is a knowledge-driven product that lets you perform a variety of data quality operations such as cleansing, matching, and data profiling. The heart of DQS is the knowledge base (DQKB) that contains a set of data domains and their relevant metadata information that you build using DQS, and then use it to perform data quality operations. The knowledge that is acquired in a knowledge base guides the data quality operations for fixing errors, standardizing, and verifying that data adheres to domain rules. It is also a key for achieving good performance in data quality operations.

This document is intended for the following two audiences:

* The [Expected Performance](#_Expected_Performance) and [Hardware and Setup Considerations](#_Hardware_and_Setup_1) sections are for **database administrators.**
* The [Working Efficiently with DQS](#_Working_Efficiently_with) and [DQS Scenarios](#_DQS_Scenarios) sections are for **DQS users** who plan and implement data quality projects.

The intention is to help plan the deployment of Data Quality Server and the Data Quality Client applications, and to provide insight into the performance of DQS for achieving good performance that leads to productive and efficient data quality projects and processes.

**IMPORTANT**: The information in this document is applicable only if you have installed [Cumulative Update 1](http://support.microsoft.com/kb/2679368) for SQL Server 2012. This update provides significant performance improvements as compared to the SQL Server 2012 RTM version. For compatibility reasons, previous implementations are maintained in the product so when reverting to a previous implementation, please refer to the [previous version](http://go.microsoft.com/fwlink/?LinkId=248561) of this document for performance guidelines.

# Expected Performance

When following the guidelines, best practices and [hardware recommendations](#_Hardware_and_Setup) described in this document, you can expect data quality operations in DQS to take the amount of time as detailed in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Size | Discovery | Cleansing | SSIS Cleansing | Matching |
| 10K | <1 minute | <1 minute | <1 minutes | <1 minute |
| 100K | 2-4 minutes | 1.5-5 minutes | 10-20 minutes | 1-2 minutes |
| 1M | 30-75 minutes | 25-75 minutes | 2-3.5 hours | 15-60 minutes |

These ranges are based on knowledge bases containing 4 to 6 enumerated string type domains with up to hundreds of thousands of values and syntax errors.

# Hardware and Setup Considerations

## Recommended Server Hardware

DQS is designed to work best on a computer system with the following operating system and hardware characteristics, depending on the expected volumes of data to be processed by DQS:

|  |  |  |
| --- | --- | --- |
| Volume | Up to one million records | One million records or more |
| Processors | One multiple cores processor | Two multiple cores processors |
| Memory | 8 to 16 GB | 16 to 32 GB |
| Disks | 1 or 2 Physical Disks | 3 to 5 Physical Disks in RAID 0 or 5 configuration |
| Network | 1 Gigabit LAN | |
| System | 64-bit Windows Server 2008 R2 | |

The implementation of DQS uses both SQL Server stored procedures and functions running in SQL CLR for getting the best of all worlds from both performance and reliability perspectives.

The following sections provide more detail regarding each of the system components above.

## Processors Considerations

### Scale-up

As a SQL Server platform implementation, DQS will automatically take advantage of as many cores that are available for SQL Server on the system for most of its queries. For queries that cannot be automatically parallelized, DQS was designed to use Service Broker facilities to allow parallelization of computational flows on separate chunks of data.

As a result, with more cores on the system, DQS can achieve a fixed amount of work faster. This characteristic is known as the scale-up factor, i.e. the amount of speed up or gained throughput that the application achieves when doubling the number of cores on the system. The scale-up factor for DQS in discovery, cleansing via project, and matching operations is about 1.6[[1]](#footnote-1). Cleansing with SSIS is an exception to the automatic scale-up capabilities of DQS and will be described in the section [Cleansing with SSIS](#_Cleansing_with_SSIS).

For other DQS operations, such as DQKB and data quality project operations (Create, Publish, Import and Export) and Server-side client support, no special DQS design optimizations were implemented to guarantee scale-up capabilities. Regardless, with multiple users simultaneously using a single Data Quality Server, it will scale-up to provide good performance for all users.

### Hyper Threading

Some modern processors have Hyper-Threading (HT) capabilities where several internal computational pipeline units are used to achieve higher processor throughput. With Hyper-Threading enabled, the operating system appears to have twice the number of logical processors as compared to when Hyper Threading is turned off.

Enabling Hyper Threading will improve DQS performance in the scalable operations (discovery, cleansing via project, and matching) and the expected scale-up factor is about 1.251.

## Memory Considerations

DQS uses SQL CLR extensively in order to get good performance in many of its basic operations. One of the drawbacks of this implementation is that SQL Server can mark the DQS AppDomain to be unloaded from memory in the event of memory pressure. The probability for these events to occur is directly related to the amount of memory installed on the system – the smaller the memory, the larger the probability for these events to occur. DQS handles most of these events properly so that when they occur DQS will recover the failing operation and continue to run from where it left off. But from a performance perspective, these events cause delays that slow down DQS. To diagnose this situation look at the SQL Server ERRORLOG for DQS AppDomain unload and reload messages. In the case they occur, what will directly help is to add more memory to the system. From DQS Performance testing, this situation rarely happens when the memory allocated is higher than the recommended memory size.

The Cumulative Update 1 for SQL Server 2012 includes several new SQL CLR In-memory implementations of selected algorithms that were previously implemented using SQL stored procedures. As a consequence, the memory usage in these implementations has increased considerably to allow significant performance speed up. Still, this situation rarely happens when using recommended memory size.

## Disks Considerations

DQS uses several databases to accomplish its tasks. DQS\_MAIN is the database where all published DQKBs are kept, DQS\_PROJECTS is the database storing all projects and any DQKBs being currently edited, and DQS\_STAGING\_DATA is optionally used for storing source and output data. By default, all these databases reside in the SQL Server instance data directory.

Every DQS operation, whether running a data quality process (discovery, cleansing or matching), managing a project or a DQKB, or accessing DQKB or project values through the Data Quality Client application, requires running queries to fetch data or write new data. And when working with large data volumes, the IO activity may reach its maximal throughput before any other resource, especially if the system does not have enough disks or if the SQL Server database files are not configured correctly.

Best results are obtained when:

1. The system has enough disks, and that all the disks are configured as one RAID-0 or RAID-5 volume (as defined above).
2. The SQL Server instance where DQS is running is defined to use these disks. This can be configured in two ways:
   1. When SQL Server is installed choose to place the SQL Server instance databases on the RAID volume.
   2. If the SQL Server instance running DQS is not configured with the default Data directory on a RAID volume, detach the DQS databases, move the corresponding files to the RAID volume, and attach.
3. Using Solid State Disks (SSD) most DQS operations will run about two times faster as compared to the same amount Serial Attached SCSI (SAS) disks on the same system.

The Cumulative Update 1 for SQL Server 2012 includes several new SQL CLR In-memory implementations of selected algorithms which were previously implemented using SQL stored procedures. As a consequence, the Disk usage in these implementations is reduced considerably to allow significant performance speed up.

## Network Considerations

Network cards are rarely the bottleneck in today’s high speed LAN networks, but slow WAN network connections across geographies are somewhat common, as well as high latency links for Wireless networks when laptops are used in the fringes of the wireless network’s reach. It is recommended to avoid using DQS over such networks and use the above recommendations.

DQS uses extensive network resources to move large amounts of data in the following cases:

1. Cleansing with SSIS using a DQS SSIS package running on one machine and Data Quality Server running on another machine.
2. Data Quality Client working on a large matching or cleansing project, or a large DQKB.
3. Importing or exporting from one DQS object to another across machine boundaries.

In all these cases it is highly unlikely to hit network latency issues across 1 gigabit LAN networks.

# Working Efficiently with DQS

## Understanding DQS Performance

Data quality works by applying acquired knowledge in the form of a “Data Quality Knowledge Base” (DQKB) to resolve errors and inconsistencies in source data. The DQKB consists of a set of data domains and each domain contains:

1. A list of **correct** **values** for the domain
2. A list of **common errors** and how to fix them
3. A list of **common variations** for terms in a domain and how to fix them
4. A list of **rules** for that the correct domain values must adhere to.

Data quality works fastest, when the vast majority of values in the source that is being cleansed or matched are found in the DQKB in one of the lists above. In that case there is an **exact match** between source and knowledge and the operation that has to take place for the specific value in question is straight forward, simple and fast.

When there is no exact match between a value in the source and the DQKB, DQS works much harder to find values in the DQKB that are **similar** to the value in question. Finding similar values involves computing similarity between the source data value and all the correct values of the relevant domain.

All DQS data quality operations, whether discovery, cleansing, or matching, are accomplished with a mix of exact and similar matching of values from the source data with those of the DQKB, and the key for getting good performance is by following two principles:

1. **maximize exact matching occurrences**
2. **maintain low cost similar matching**

**Maximizing exact matching occurrences (A)** is accomplished with **rich** knowledge. Knowledge richness is the degree to which the knowledge covers the source data that is being cleansed or matched. When the DQKB is rich, there is high probability for finding a source data value within the DQKB. With a rich DQKB most source data values are found. With a poor DQKB, a high percentage of source data values is not found.

**Maintaining low cost in similar matching (B)** is accomplished with **credible** knowledge. Knowledge is credible when the vast majority of correct values are verified or trusted. Using a credible DQKB makes sure no expensive similar matches are wasted on matching to domain values that are either invalid or errors of other values that are not in the DQKB.

These two principals will be examined in more details in the next sections.

## The Big Picture - DQS Data Quality Project Lifecycle

In its most general form, a DQS data quality project will have the following phases:

1. Knowledge acquisition
2. Interactive Cleansing
3. Cleansing & Matching

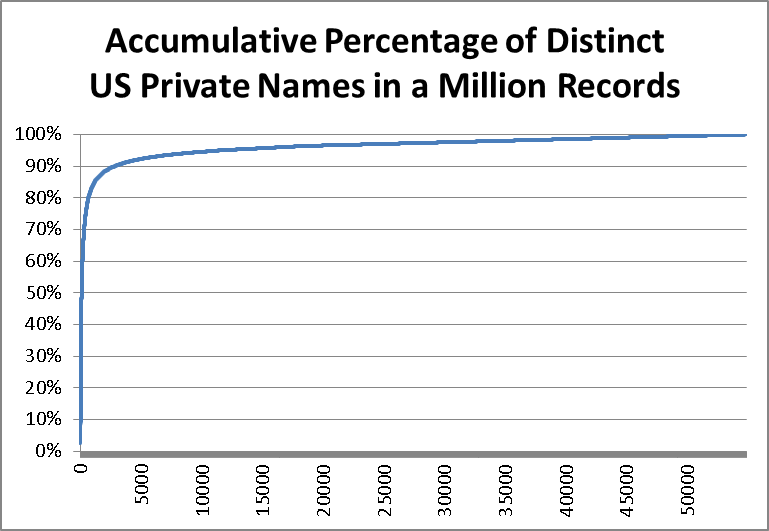
Since all of these phases involve human interaction and especially the first two, it is beneficial to use good planning practices that will reduce total end-to-end data quality project time and labor investments. The following describe each phase in more detail.

Knowledge acquisition is the phase focusing on building the initial DQKB. The goal of this phase is to create knowledge that contains most of the correct values in each domain. A powerful DQS operation to assist in this phase is ***Knowledge Discovery*** that can help extract knowledge from your data. This phase consists of several iterations of these three basic steps:

1. **Domain Management** - manually add or import new domain values, errors and rules to the DQKB
2. **Knowledge Discovery** – an automated process for discovering knowledge from source data
3. **Discovery Value Management** – manually go over the discovery results for each domain and fix issues that are found.

Due to steps (1) and (3) this phase is highly interactive and for complex domains with tens or hundreds of thousands of values it can become labor intensive, especially if not planned correctly.

Fortunately, most enumerated string type data domains satisfy the **popularity distribution**, a statistical property that considerably helps reduce the amount of labor in this phase. According to the popularity distribution (also known as the **80-20 rule,** or the **power law**) in most string data samples there are a relatively small number of distinct values that are very popular and a large number of values that are rare. The following graph illustrates the popularity distribution by showing the frequencies of private names of people in a table with 1,000,000 records:



The distinct names are ordered in descending order of frequency. This figure shows that there are about 55,000 distinct name values in this field, and about 5% (2,500) of them account for 90% of the values in the source. It also shows that the values from location 10,000 to 55,000 (45,000 or 82% of the distinct values found in the source) all have three or less occurrence in the table.

Due to the popularity distribution it is highly probable that after a few iterations in the knowledge acquisition phase the DQKB will have gathered most of the popular domain values that account for most of the source data. On the other hand, the popularity distribution also shows that most of the distinct values that are discovered in this phase are relatively rare and not credible by definition, and therefore have the potential of harming performance by violating performance principle (B) - Maintaining low cost in similar matching**.**

Interactive Cleansing is the next phase where the DQKB is rich enough for making cleansing work fast, that is the majority of the correct source data is found in the DQKB, but the DQKB still does not have a big enough repository of errors. Like knowledge acquisition, this phase is also iterative and interactive with the following steps:

1. **Cleansing** – an automated process for cleansing source data with a DQKB.
2. **Interactive Cleansing** – manually go over the cleansing results and fix issues that are found.
3. **Enrich Knowledge –** manuallyadd “new” knowledge that was found back into the DQKB.

Steps (2) and (3) are interactive and therefore this phase requires manual labor, but again due to the popularity distribution it is most likely to be a converging iterative process where each additional iteration will add fewer new elements to the DQKB as compared to previous iterations until it is ready for the next phase – batch cleansing and matching.

Cleansing & Matching is the last phase where the DQKB has reached its stable condition where it is the richest and most credible and can be used for efficient and high quality cleansing and matching projects.

After outlining the typical DQS data quality project lifecycle, the next sections will provide best practices for each phase with a focus on efficiency and performance.

Carefully Plan the Knowledge Structure  
Planning the DQS Knowledge structure should reflect business needs, but should also take into considerations the source data characteristics and the domains’ complexity. The knowledge design determines and defines the number of DQKBs, domains, values, rules and as a result will have an impact on performance and processes.

The following are design guidelines and best practices that follow good performance principles.

Use language and geography specific domains. In the case of multi lingual and multi geography mixed data, consider building geography specific DQKBs and domains. Building well defined geographical domains will enable using language specific domain properties as the speller more efficiently and create simpler rules in the case of different geography validations. This design guideline will result in smaller and simpler DQKBs and domains. It may complicate the quality process, because it requires preprocessing steps of splitting and arranging the source data in separate geographies and languages, but it will reflect in better performance, more accurate results and better tracking and controllability.

Composite Domains. Whenever a source data field is populated with data that represent different domains, strive to use composite domain. Single domains enable a more straightforward definition that reflect in smaller number of values and simpler definition. For example a Name source field that includes “Mr. David Faibish” should be defined as 3 separate domains: Salutation, Given name and Family name. This will allow faster processing and usage of advanced parsing functionalities. In addition it will allow the ability to correct/identify/enrich the Salutation info if needed.

## Knowledge Acquisition - Acquire Knowledge a Chunk at a Time

As described so far, **knowledge acquisition** is iterative and interactive. To get best results, here is a list of best practices for coping with large and complex domains:

Divide Discovery into Chunks of Records. Each iteration takes 5,000 to 50,000 records from the source to run knowledge discovery on. The number of records for each iteration has to be “big-enough” to be able to find a considerable amount of high confidence values, and “small-enough” to make the interactive discovery value management session manageable and efficient.

Verify Discovery Results. Invest time to go over all the results in the discovery value management session following knowledge discovery. Especially pay attention to **new[[2]](#footnote-2)** low frequency values since they are automatically marked as **correct** even though the confidence in their correctness is very low. Here are some possibilities:

* A **new** value that is probably an error of another **correct** value with high confidence, either existing or new, must be **corrected** to that value. Let’s illustrate with an example: suppose the source contains a single occurrence of the Private Name “Alekkxssandra” and knowledge discovery does not find any similar values for this name in the DQKB, so “Alekkxssandra” is marked **correct**. Still, a data steward viewing this name can recognize that it is an error of “Alexandra”. The data steward has two options in this case – either mark it as an **error** with a **correct-to** value of “Alexandra”, or delete it altogether if there is doubt concerning the **correct-to** value. Keeping “Alekkxssandra” **correct** in the DQKB will harm performance since it will probably not be found exactly matching any source data value in upcoming discovery or cleansing sessions, and will needlessly participate in all similar matches, wasting valuable time and resources.
* Every new value that is truly invalid must be explicitly marked as **invalid**. For example: “(name missing)”, “-----”, “12345” are invalid Private Name values that the DQKB must not contain as **correct** values, and wherever possible rules should be added to identify them and possibly correct them.
* When unsure if a **new** value is correct, or recognized as incorrect but in doubt regarding the **correct-to** value, it is generally best to delete the value. There is no point in keeping **correct** values in the knowledge base when there is no confidence in their correctness.

Verifying discovery results is a key element for keeping the DQKB accurate and credible throughout this phase. And due to performance principle (B) – failing to correct the bulk of low credible correct values found in discovery, will lead to a very large number of unnecessary similarity computation between source data and low credible correct DQKB values.

Keep track of knowledge growth. In each iteration of the knowledge acquisition phase, keep track of the number of new values that are added to the DQKB in that iteration. According to the popularity distribution this number will gradually decrease over time, since discovery will find more and more exact matches to previously acquired knowledge. When this number is low enough, it marks the end of the knowledge acquisition phase and from this point on the main focus is on finding errors of existing knowledge. This is when the project changes to the new phase - Interactive Cleansing.

Work on each domain separately. Knowledge discovery is a serial process and it is not designed for collaborative user interaction. This means that only one user at a time can build or update a DQKB. Still, since each domain is independent of other domains, it is possible for several users to work simultaneously in isolation for this phase if they work on separate domains, each in its separate DQKB. When all users complete their work, there is an additional step of manually importing the disparate knowledge that was acquired in each to a single unified DQKB containing all the domains.

## Interactive Cleansing - Enrich Knowledge with Common Errors

Once a DQKB contains most of the correct domain values, there is no point continuing with knowledge discovery since the algorithms in the discovery process are designed for identifying **new** **correct** values and at this stage those are coming in at a very low rate. But there are still many common errors in the source data that would be beneficial to add to the DQKB for increasing the quality of the following cleansing and matching processes.

The second phase, **Interactive Cleansing** is designed exactly for this – cleansing data while enriching the knowledge base with common errors. The cleansing process does not use the heavy weight knowledge discovery algorithms when examining the source data but it still uses similarity matching to identify possible errors of **correct** DQKB values. These errors should be verified and added to the DQKB with the **Import Project Values** feature. This phase is also interactive but unlike knowledge discovery, it can be performed in parallel by several users working on the same DQKB.

Here are some best practices for this phase:

Divide Cleansing into Chunks of Records. Each iteration takes 10,000 to 100,000 records of source data for each round of the Cleansing activity. This amount of records per sessions is recommended for keeping the interactive step in each round easily manageable.

Focus on New and Suggested Values. Since the DQKB was built so far with credibility in mind, the **correct** and **corrected** values in the cleansing interaction sessions are hopefully all valid. But since the DQKB is still not rich enough with common errors, all the verification work to be done at this stage is concentrated in the **suggested** and **new** values. Values on the **suggested** tab in Data Quality Client (having the **suggested** status) will contain suggestions for previously unseen values that were found in the cleansing activity to be similar to existing **correct** values in the DQKB. It is expected that most of these suggestions are good and therefore their verification is simple and fast.

**New** values will contain mostly error prone values for which cleansing did not find similarities to DQKB **correct** values. These will fall into two main categories:

* Correct New Values – since most of the popular **correct** domain values have already been found in the knowledge acquisition phase, there will be a relatively small number of new correct values to add. These values should be marked as **correct**.
* Error values for which similarity matching failed to provide good suggestions – these can be errors of rare domain values or just errors that similarity matching cannot indicate as errors with high confidence. These values should be marked as **errors** when there is high confidence in a **correct-to** value.

The same best practices that were good for the discovery verification step in Knowledge Acquisition, apply here as well:

* Invest time to thoroughly go over **New** and **Suggested** values.
* Change every **new** value that is an error to its **correct-to** value.
* Reject every new or suggested value that is invalid.

Enrich DQKB with Cleansing results. After finishing an interactive cleansing activity, there are new values and errors in the results of the data project that can enrich the DQKB, i.e. increase the probability for exact matches in the next cleansing sessions (contributing to performance principle (A)). This is achieved by revisiting the domain management activity and on a selected domain click the import button to **import project values**. As in knowledge acquisition, the same best practices apply here:

* Invest the time to thoroughly go over New and Suggested values.
* Correct every new value that is an error of another correct value.
* Reject every new value that is invalid.
* Delete every new low-confidence value.

Keep track of Knowledge growth. In every Import Project Values session that is performed, keep track of the number of values and errors that are actually added to the DQKB. These numbers may not necessarily converge to zero since due to the popularity distribution the source data contains a very large tail of rare values. But it is likely that they converge to some fixed level. From this point on, cleansing does not require any further interaction and cleansing can enter the batch processing phase.

Divide Manual work between several users. In the Cleaning with Interaction phase it is possible and recommended for several users to work simultaneously on different portions of the source data, each in his/her own separate DQS project. This will reduce overall elapsed time that is invested in this phase. For example you may batch 100,000 rows into separate cleansing data projects, and each user may work independently. Use staging tables to batch the rows into separate source SQL Server tables, or define SQL Server views to help divide large data sets into smaller parts with WHERE clauses, or separate Excel worksheets, or partition data based on categories of the data. There are many possible ways to divide the data before pointing to the source data in the DQS data projects. You may wish to carefully slice the parts based on category or divisions or a regional column in your business, so that you may better evaluate values that are somewhat similar within those parts when using divide and conquer approaches.

## Finally - Cleansing (Batch) and Matching

At this point, the data quality project has reached the state where the DQKB is rich and credible enough to allow efficient cleansing and matching processes to take place. This is the final state of the project where most of the source data will be processed. In this final phase, Cleansing can be accomplished with automated SSIS components that do not require any interaction. On the other hand, the DQS Matching operation requires interaction since it requires human decision making for getting high quality results. For this reason SSIS components are not provided for DQS Matching.

Best practices that are relevant for this phase include:

Constantly Monitor Cleansing Results. Even though the project has reached its stable state, it still requires constant monitoring for making sure that every source data still fulfills the expected statistical behavior that new values are seen at the expected rates.

# DQS Scenarios

This section describes how each scenario impacts performance and guidelines for achieving best performance in each.

## Data size impact

Usually data size has impact on DQS performance in several parameters:

* **Number of records of the source data**. Processing time tends to grow linearly with number of rows, with the exception of Matching in some circumstances (see [Matching](#_Matching) for details)
* **Number of domains**. Processing time tends to grow linearly with number of domains. The number of columns in the table does not have impact on DQS performance since DQS parses and analyzes only table columns that are mapped to DQKB domains.

## Knowledge Discovery Activity

According to the performance principles outlined in [Understanding DQS Performance](#_Understanding_DQS_Performance), it is recommended to build DQKBs that are both rich and credible. Failing to do so will have large performance impact on knowledge discovery. Here is the reason why:

Knowledge Discovery performs two main operations:

1. Parse the source data while identifying exact matches to existing DQKB values, either correct or error.
2. For new values in the source data (for which no exact DQKB matched were found), look for similarities within all correct DQKB values.

A DQKB that is not rich will cause many source data values to be unmatched during the first operation. This is the case in the early iterations of knowledge discovery since the DQKB is still relatively empty and young. A DQKB that is not credible may have a large number of values that are deemed correct but only with low confidence. These low-confidence **correct** values will cause a large number of similar matching operations to be evaluated internally by the DQS algorithms. This scenario happens if the data stewards are not investing the necessary manual efforts to delete the low confidence values, or when they automatically marked them as **correct** in the discovery result tabs. The total knowledge discovery processing time is a function of the multiplication of both. This means that the key to avoid long discovery runtimes is to:

1. Work in small enough chunks (i.e. 5,000 to 50,000 records at a time). [Acquire Knowledge a Chunk at a Time](#_Acquire_Knowledge_a) explains how the popularity distribution helps in this situation.
2. Keep credibility as high as possible, i.e. follow results verification best practices outlined in [Acquire Knowledge a Chunk at a Time](#_Acquire_Knowledge_a).

## Cleansing Data Project Activity

In general, cleansing performs the same operations as discovery but works much faster since it is applied at a stage where the DQKB is rich and uses a different set of algorithms for finding similarity between values. At this rich stage, we expect a relatively small number of **new** values to evaluate when DQS compares source data to DQKB domain values. Therefore the multiplication effect is relatively minor when DQS runs cleansing algorithms to compare source data values to domain values in a rich DQKB.

For understanding the performance impact on Cleansing when using advanced DQS features such as Composite Domains, Term-Based Relations, Rules, and Reference Data Services, refer to [Advanced DQS Domain Features](#_Advanced_DQS_Domain).

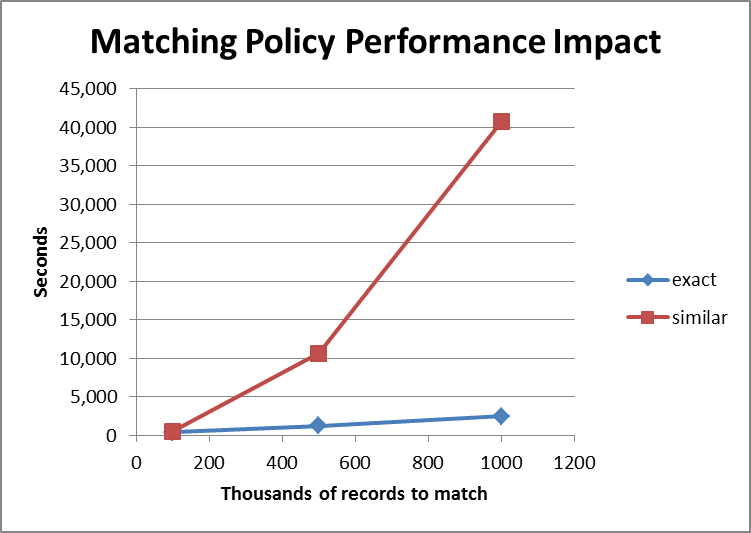
## Matching Data Project Activity

Matching activities perform best when there is a rich DQKB with the most popular known values saved in the DQKB domains. In absence of this kind of rich knowledge, consider the complexity of the general data matching problem. Since the source data is not yet cleansed, the data values may contain errors. The DQS approach to matching such values has to compute similarity between every pair of distinct values that are found across every domain that is mapped in the matching project source data. This problem inherently has quadratic complexity, i.e. 1,000 distinct values in a domain require 500,000 similarity computations, and 10,000 distinct values in a domain require 50,000,000 computations.

You can help DQS reduce the amount of similarity computations by applying two powerful optimization techniques:

1. When defining a Matching Policy in the DQKB, it is highly recommended to use pre-requisite or “Exact” fields within a matching policy rule. This setting helps the internal DQS matching algorithm at the later time when running the Matching data project by creating independent buckets of distinct values where matching is performed within buckets. For example, using the pre-requisite matching rule on a high confidence ZIP code column will allow DQS to divide the source data set into many small buckets. In each small bucket the matching algorithm computes similarity for street names that are only within the same ZIP code, and street names observed in records that have different ZIP codes will never be compared. Therefore DQS is able to divide and conquer the source data matching project workload in an optimized fashion.
2. Having **rich** DQKB knowledge with the most popular known correct values and corrections helps optimize matching data project activities by reducing the number of distinct values needed to be compared. For example, there is no need to compare misspelled values “Micael” with “Michail” for finding any potential matches, if both are found in the DQKB and corrected to “Michael”.

Utilizing these two techniques in a DQKB matching policy can practically transform the quadratic nature of the matching data project workload efficiency, to the point where it scales linearly, even up to a very large amount of records. The following figure demonstrates how a matching policy rule with exact-matching on a ZIP code domain impacts matching data project performance, as compared to the same matching data project on the same data set using a matching policy that does not have any rule with exact matching option specified.



When utilizing these two optimization techniques, its best to clean any exact-match columns especially before matching the data set. Given the impressive impact on matching performance, it is highly recommended to **run the cleansing** activity against the source data table and save the results, **prior** **to running the matching project** on those same results. For best efficiency, make sure that exact match columns are in the near perfect cleansed state prior to running the matching data project against them.

For example, consider a table with records of patients containing hospital name and city columns along with other patient information. After cleansing the hospital name and the city columns, these domains can be used as pre-requisite or exact match conditions since they are now relatively correct and clean, and the internal DQS matching project algorithms can split the table into a large number of small disjoint record sets before doing any similarity comparisons.

Another tactic that can be helpful in matching is to define a matching policy with several rules, each a permutation of similar domains staggered against the pre-requisite domains. In this policy each rule includes each domain in a different permutation. Whenever one domain is matched with a similarity condition (matching policy rule defined with Similarity=**Similar**) ensure all the other domains are included as pre-requisites (pre-requisite checkmark is checked) in the same rule. For example, to match a table with three columns A, B and C, the alternating similar domains matching policy has the following rules:

* A as similar, B and C as pre-requisite
* B as similar, A and C as pre-requisite
* C as similar, A and B as pre-requisite

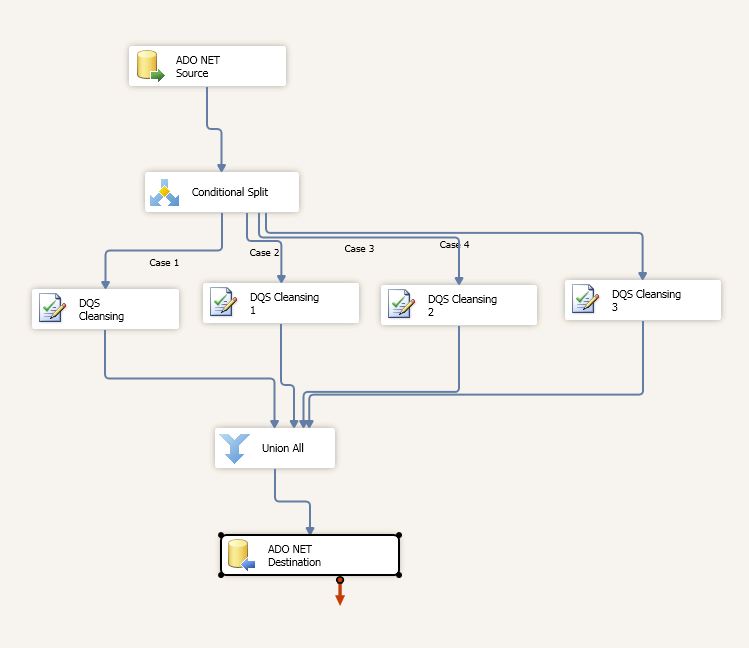
As you can see, a single similar condition alternates between all the columns that are matched, while all the other columns have pre-requisite condition. Using this policy, matching will run very fast as compared to an all-similar matching policy, but the quality of the results will be insufficient in cases where there are many records that have more than one error across all columns that are matched.

## Cleansing with SSIS

Cleansing with SSIS is used for performing cleansing without any manual interaction, so it is fit for running during out of office hours or on an automated schedule. It is designed as a background processing activity rather than being a foreground interactive cleansing activity using the cleansing project in Data Quality Client. Therefore the SSIS cleansing activity resource utilization footprint is intended to be much lower than when running Cleansing in Data Quality Client interactively. SSIS data cleansing can be executed in the background with minimal impact on system performance as compared to higher priority foreground processes.

Each DQS Cleansing Component in a SSIS Dataflow Task utilizes only a single CPU core on Data Quality Server and runs synchronously in the SSIS dataflow. When using Data Quality Client to run Cleansing-in-a-project, Data Quality Server utilizes multiple CPU cores. To better scale the SSIS DQS Cleansing Component usage of CPU cores, it is possible to design SSIS packages to work faster by combining several DQS components working in tandem within one single dataflow task to clean one set of source data. Each DQS Cleansing Component can run on a different subset of the source records, accomplished by a **conditional split component** in the SSIS dataflow to divide the source data set into parallel pipelines evenly. Alternatively you may also use multiple DQS Cleansing components across multiple separate dataflow tasks for cleansing different sets of source data in tandem to leverage multiple CPU cores.

Here is an example of an SSIS package Dataflow Task which uses four DQS Cleansing components on four parallel pipelines within a single Dataflow Task. The design uses a **Conditional Split** component configured with the modulo (%) operator applied on an integer ID columns in the source data to split the source data into four even parallel flow pipelines. After the cleansing components, a **Union All** component is used to join the resulting cleansed records into a single pipeline to be saved into the destination. This package achieves a 2.5 to 3 fold run time improvement as compared to a single DQS Cleansing component on a single dataflow pipeline containing the same rows.



## Advanced DQS Domain Features

DQS offers a variety of advanced domain features that help improve the quality of DQS results. These features include Composite Domains (CD), Term-Based Relations (TBR), Domain and Composite Domain Rules (DR and CDR), and Reference Data Services (RDS). With the exception of RDS, these features are designed to have low impact on the performance of DQS operations when used properly. The next sections describe how each works and how each should be used for proper DQS functioning.

### Composite Domains

Composite domains (CD) provide the ability to treat several single domains as one. This is useful when:

* It makes sense to validate several domains with composite domain rules. A composite domain rule is similar to the regular domain rule except that it evaluates several domains in a single rule and optionally corrects a value. See [Domain Rules](#_Domain_Rules) below.
* A column contains data from several domains, for example a *full address* contains the domains *house number*, *street name* and *city name*. In this case, there is an extra parsing operation on the column to separate the terms to the contained domains. DQS offers two possible parsing methods – ordered and DQKB guided. Ordered parsing just breaks the content of the field to the contained domains while DQKB guided parsing performs an exact match for each separated term in the field in every contained domain until there is a match. Both have a very low performance impact in any DQS data quality operation.
* It makes sense to use Reference Data Services for several domains. For example, group street name, city and postal code domains in a composite address domain to verify with an address RDS provider. RDS has significant impact on DQS performance due to network and service provider latencies. See [Reference Data Service](#_Reference_Data_Service) below.

The best practice for using CD is to use it only when necessary and useful (following the above guidelines) and add to it only those domains that are required for its proper function in DQS. For example, it is not good to define a *person* as a composite domain containing all domains that have any relation to it (e.g. name, address, social security number, occupation, etc.) but without any relevance to DQS. This causes extra and redundant domain definition having a total negative impact on performance.

### Domain Rules

Domain Rules (DR) and Composite Domain Rules (CDR) provide the ability to check validity of values in a domain. Most are evaluated after the source data is parsed into the DQS project using a T-SQL query where every rule is a query condition. This implementation uses the strength of SQL Server query processing and the overall impact of Domain Rules on DQS performance is very low.

An exception to this implementation is the regular expression evaluator which executes within SQL CLR. The regular expression evaluator is executed on new values so normally its impact on performance is also very low.

Domain Rules are designed for up to several dozens of rules per domain. Beyond that it becomes hard to manage and therefore not recommended.

### Term-Based Relations

Term-Based Relations (TBR) are useful for detecting optional values of terms composing a domain and correcting them to standard form. Whenever TBRs are defined for a domain, an extra parsing step and is performed on new source data values and the matching TBRs are fetched from the DQKB. Again, with the help of the popularity distribution, most source data values will not require this extra parsing and TBR lookup, so the bottom line is that TBRs impose a very low performance impact.

### Reference Data Service

Reference Data Service (RDS) provides the ability to use highly credible knowledge from external service providers in DQS Cleansing. In general, when a domain is configured to use an RDS, every new source data value is potentially sent to the service. The service then checks the validity of that value and sends back a response with a possible correction. All this happens over a WAN network link, so naturally there is a very high latency penalty. Therefore using RDS will noticeably impact the performance of DQS Cleansing as compared to not using RDS.

If using RDS, it is recommended to continuously [Monitor Data Exchange between DQS and RDS Providers](http://social.technet.microsoft.com/wiki/contents/articles/6831.monitor-data-exchange-between-dqs-and-reference-data-service-providers.aspx) in order to understand the implications of provider responsiveness on cleansing performance.

## Managing DQKBs and Projects

DQS provides the ability to manage DQKBs and Projects. The common management operations that have noticeable duration are:

* Create a DQKB an existing DQKB.
* Publish a DQKB
* Create a new Project
* Import DQKB from a DQS file
* Import Domain Values from a Project.

In general, these operations scale linearly with the size of the DQKB and can often take minutes for large ones that have hundreds of thousands of values. Most of these involve copying the published DQKB tables from DQS\_MAIN to DQS\_PROJECTS or vice versa, so optimizing the position of the data files onto several fast disks may help speed these operations.

## Working with Data Quality Client

In general, there are two types of views in the Data Quality Client application:

* DQS execution views – those in which a DQS data quality operation is activated and monitored (e.g. knowledge discovery, cleansing, or matching).
* DQS interaction views – those that are used to interact with DQKB domain values or results from Cleansing or Matching operations.

In execution views, Data Quality Client will trigger internal DQS service interfaces on the Data Quality Server side hosted in SQLServr.exe. Progress can be observed in the Profiler tab. The Profiler queries the server and displays monitoring and profiling information relevant to the progress of the currently running DQS operation. The data exchange is simply to observe the progress between Data Quality Client and Data Quality Server, and it is very low and therefore there is no special performance consideration to take into account.

In interaction views there are times when there can be a significant data exchange between Data Quality Client and Data Quality Server. Some scenarios include providing Excel or CSV files in discovery or project activities, when scrolling through numerous results in a project results page, when exporting project results to a CSV or Excel file, or when importing and exporting a DQKB or an individual domain to and from a DQS file. At these times, Data Quality Client and Data Quality Server exchange large amounts of data over the network. This can be a noticeable performance issue when Data Quality Client and Data Quality Server connect across WAN or low bandwidth wireless links.

1. This value in an average of performance measurements obtained on recommended dual processor hardware. It is highly dependent on the amount of the other hardware resources (memory, disks and network) and the specific data sets that were used in DQS performance testing. [↑](#footnote-ref-1)
2. In the following sections, bold formatting of the words **new**, **correct**, **corrected**, **error**, **correct-to**, and **invalid** refer to correction statuses for values in a DQS project or DQKB. When regular (not bold) they should be read as their semantic meaning. [↑](#footnote-ref-2)