TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Konstantin Dmitrijev 175798IDSR

Analysis of Data Quality Management Service Component in Financial Institution's Data Warehouse

Diploma thesis

Supervisor: Nadežda Furs
MBA

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Konstantin Dmitrijev 175798IDSR

Andmekvaliteedi juhtimise teenuskomponendi analüüs finantsettevõte andmeaidas

Diplomitöö

Juhendaja: Nadežda Furs
MBA

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Konstantin Dmitrijev

17.05.2021

Abstract

The aim of the thesis is to update existing Architectural Description of the Data Quality Management Component of Enterprise Data Warehouse environment in Financial Institution N., and to do its performance analysis in EDW.

In the scope of the thesis, in its first part, author performed component analysis and updated existing documentation and created a DQM component introduction tutorial for component users. In second part author did a performance analysis of the component's executable file's source code and its validation scripts. As a result, author described issues that caused bad performance, suggested solutions, and made an appropriate update.

The results of the work helped updated common knowledge base of component and drastically increased performance of the EDW processes built upon the DQM Component.

This thesis is written in English as official working language of the Financial institution is English. Thesis is 41 pages long, including 4 chapters, 10 figures and 4 tables.

Annotatsioon

Käesoleva diplomitöö eesmärgiks on täiendada finantsettevõte andmeaida andmekvaliteedi juhtimise teenuskomponendi (*ingl.k* Data Quality Management Component) puuduliku arhitektuurse dokumentatsiooni ja teostada jõudluse analüüsi andmeaidas. Töö metoodikaks on komponendi analüüs, mis eeldab, et enne praktiliste probleemide lahendamist, tuginedes olemasolevatele informatsiooni allikatele tehakse selgeks vaadeldava komponendi struktuur ja töökäik.

Püstitatud eesmärkide saavutamiseks teostas autor teenuskomponendi analüüsi, dokumenteerides komponendi struktuuri ja selle töökäigu. Analüüsi aluseks oli kehtiv arhitektuurne kirjeldus, komponendi programmi lähtekood ning andmeaida arhitekti ja komponendi praeguse arendaja konsultatsiooni ajal saadud informatsioon. Komponendi analüüsi tulemuseks oli täiendatud arhitektuurne kirjeldus ja uus komponendi tutvustatav koolitus.

Komponendi analüüsile järgnevas jõudluse analüüsi esimeses osas vaatas autor üle teenuskomponendi programmi lähtekoodi ning tegi selgeks kehva jõudluse põhjuse ja pakkus lahenduse ning lõi toimiva prototüübi. Prototüüp sai testitud ja testimise tulemused esitatud teenuse juhile. Teises osas vaatas autor üle valideerimise SQL skripte jõudluse ja selle tulemusena uuendas tulemuste resümeerimiseks käivitatava skripti.

Töö esimeses pooles annab autor üldise ülevaate andmeaitadest, selle olulisematest komponentidest, laadimise protsessist, selgitab andmekvaliteedi mõistet ja selle olulisust. Töö teine pool sisaldab teostatud analüüsi tulemusi: toodud välja jõudluse põhjustatavad probleemid ning nende lahendused.

Lõputöö on kirjutatud inglise keeles kuna finantsettevõte ametlik töökeel on inglise keel. Lõputöö sisaldab teksti 37 leheküljel, 4 peatükki, 8 joonist, 3 tabelit.

List of abbreviations and terms

AMP	Access Module Processor - Linux process responsible for handling its individual share of data
DM	Data Mart
DQM	Data Quality Management
EDW	Enterprise Data Warehouse
ETL	Extract, transform, load
ODI	Oracle Data Integration tool
SA	Staging Area
SQL	Structured Query Language
UML	Unified Modeling Language

Table of contents

List of figures

List of tables

1 Introduction

1.1 The Background and the Problem

Data Quality Management Component (DQM component) is a service component within the Financial Institution's N. data warehouse (EDW) environment developed for the purposes of data validation. Its aim is to perform validation of data in EDW based on user's input parameters and predefined validation scripts, store results of validations and inform interested parties regarding validation results.

Throughout the lifecycle of the service component several modifications and migration to a different platform were applied. As a result, all the mistakes made during the initial development and modification process over time started to cause poor performance on certain stages of the execution of the Component and big part of implemented and modified functionality was not properly documented.

In context of increasing number of new validation scripts added to DQM component the cumulative effect of the poor performance caused the significant part of the EDW resources to be used by EDW processes built upon the DQM component.

Additional difficulties also appeared after transferring the maintenance and development responsibilities to another team within the company. New team lacks full overview of the DQM component and its structure.

1.2 The Task of the Thesis

Based on the problems described above aims of the thesis are:

1. Analyze the structure and workflow of the Data Quality Management Service Component and document its current state.

2. Based on results of the analysis

    a. create a comprehensive Architectural Description to provide developers and users of the DQM component with a relevant knowledge asset.

    b. analyze poor performance of the DQM component

c. work out the solution to improve performance

Parties, interested in results of the thesis, is the maintaining team of the Component, Data Quality team and the Data Warehouse department in general. The estimated benefit of the work is to prepare a solid knowledge base for future maintenance and enhancements and to decrease resource consumption and the overall processing time of DQM processes in the EDW and its' infrastructure.

1.2.1 Role of the author

Author of the thesis performed analysis of the DQM component and of its initial state. During the analysis phase author collected all the existing documentation, source code of the component, interviewed the EDW architect and current developer of the component. Based on collected information author supplemented an architectural description of the component.

Author of the thesis conducted the performance analysis of the component on the database level. Based on the results in cooperation with colleague developer author created a prototype of the component with significantly improved performance. Working version of the improved component is currently in the testing phase and will be released after the testing and approving according to the company's release s procedures. Additionally to the new version performance of some validation scripts were also analyzed and improved.

During the work on the tasks author cooperated with two colleague developers and EDW architect. All stages of development and implementations were performed according to company's efficient process of work organization.

1.3 Methodology

Thesis methodology is built upon Component Analysis. Component Analysis will include study of existing documentation, component source code and database structure and based on the results of the analysis author will create the description of the key processes.

Based on the result DQM component analysis author will introduce new version of DQM Service Component architectural description to responsible party.

After the Component Analysis author will review the performance issues of the Component. To do so author will analyze the DQM processes' executions data stored in EDW metadata schema. After that author is dedicated to solving the issue or in case solution cannot be performed by the author suggest a solution measures to a responsible development team.

1.3.1 Unified Modeling Language

For the purpose of the visualization of the current state of the component and future enhancement Unified Modelling Language (UML) is planned to be used.

The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes [1].

There are two major types of UML diagrams: structure diagrams and behavioral diagrams (and within those categories lie multiple others, see Figure 1). These variations exist to represent the numerous types of scenarios and diagrams that different types of people use [1].

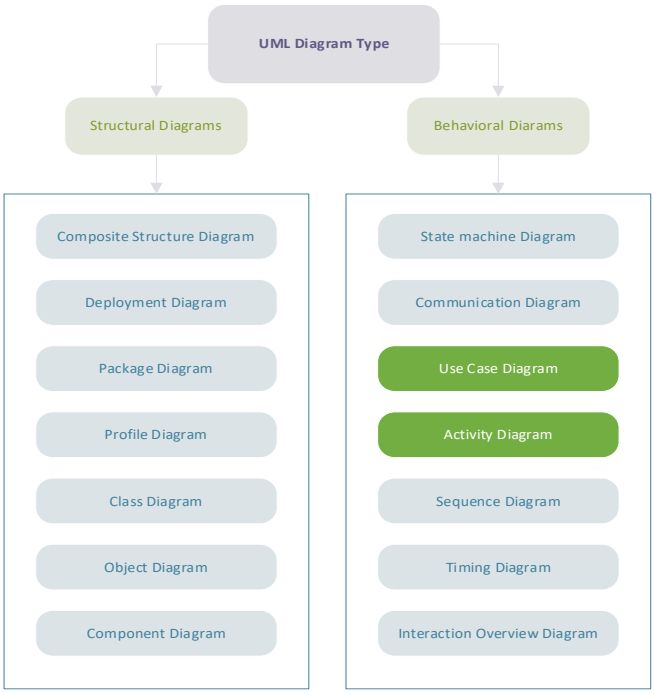

Figure 1. UML diagram types according to UML specification [1]

In this thesis author uses Use Case and Activity diagrams to describe work of the DQM Component.

1.4 Overview of the Thesis

Thesis consists of three parts. In the first part author gives an overview of the concepts of Data Warehousing and data loading procedure, Data Quality Management and its importance.

In second part author will introduce the results of DQM Component analysis and describe its outcome.

Third part will include the description of the performance analysis and measures applied for its improvement.

In the end of the thesis author summarizes all activities done in the scope of the thesis.

2 Data Warehousing and Data Quality Management

It is commonly recognized that data is a vital business asset [2]. Collected data and information formulated upon this data give a valuable insight on customers, products, and services. It helps to identify problems in business models, improve offered services or products and reach strategic goals. All that is only possible if data management activities are applied in everyday work of organization and if organization can verify its data to get strategic value from it.

Below in this chapter there will be given an of data warehousing, its related services, and data quality management concepts.

2.1 Data Warehousing

Data Warehouse enables organizations to integrate data from range of sources into a common data model. The primary driver for data warehousing is to support operational functions, compliance requirements and Business Intelligence activities. Data Warehouse (DW) is a combination of two primary components: an integrated decision support database and the related software programs used to collect, cleanse, transform and store data from variety a of operational and external sources [3, p359].

According to Oracle Database Data Warehousing Guide Data Warehouse is ‘designed to enable business intelligence activities: it exists to help users understand and enhance their organization's performance. It is designed for query and analysis rather than for transaction processing, and usually contains historical data derived from transaction data, but can include data from other sources. Data warehouses separate analysis workload from transaction workload and enable an organization to consolidate data from several sources’ [4].

There are different common DW architectural solutions that are widely used across the world. DW architecture vary depending on each organization’s needs and requirements. Some may or may not include Staging Area or Data Marts, source data may be loaded from external database, form source file or from both simultaneously. Figure 2 is illustrating an Enterprise Data Warehouse architecture with different types of Data Sources, Staging Area and Data Marts.

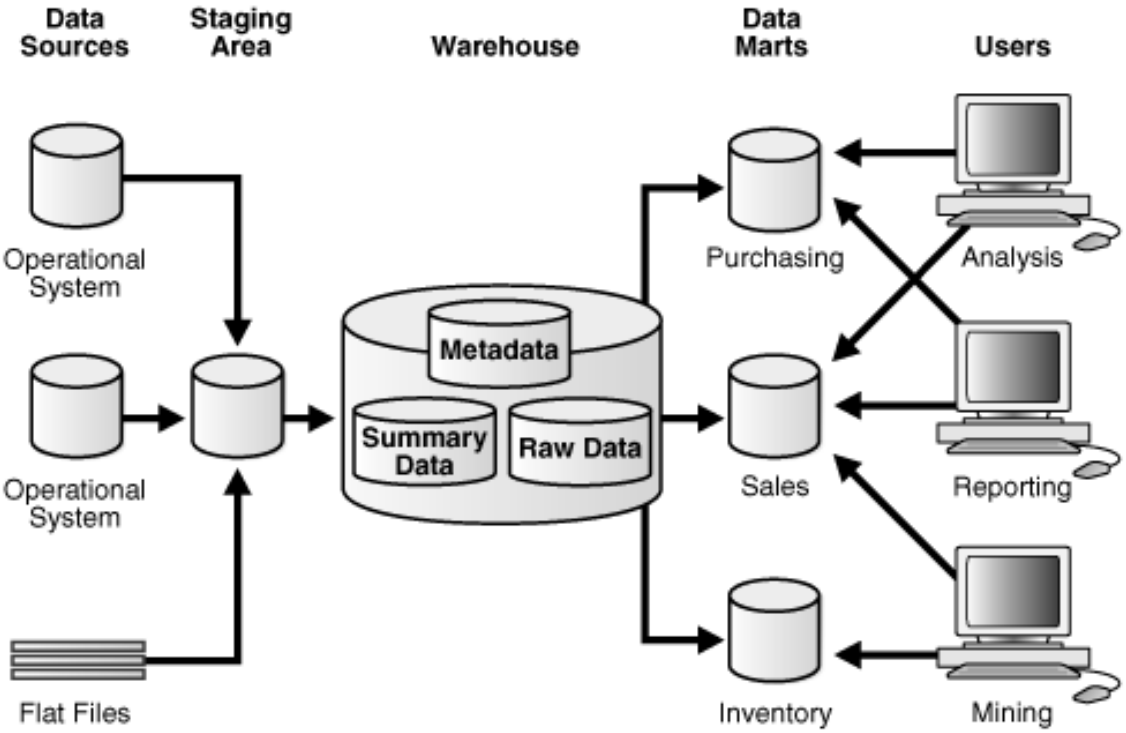

Figure 2. EDW architecture with a Staging Area and Data Marts [4, Figure 1-3]

The core part of the data warehouse is a consolidated storage that keeps all the relevant business information. Before data gets loaded it must be cleaned and processed. For that most data warehouses use staging area. It simplifies cleansing and processing for data coming from multiple sources. After data is loaded it can be accessible to its end-users via the data marts – databases that are designed for a part of organization and includes only certain type of data or through the direct access to data warehouse.

2.2 Data Integration and ETL processes

Data Integration describes processes related to the movement and consolidation of data within and between data stores, applications, and organizations. Integration consolidates data into consistent forms wither physically or virtual [3, p257]. A common problem that organizations face is how to gather data from multiple sources, in multiple formats, and move it to one or more data stores. The destination may not be the same type of data store as the source, and often the format is different, or the data needs to be shaped or cleaned before loading it into its destination. That task is fulfilled by three basic processes: extract, transform and load (ETL). Extract, transform, and load (ETL) is a data pipeline used to collect data from various sources, transform the data according to business rules, and load

it into a destination data store. The transformation work in ETL takes place in a specialized engine, and often involves using staging tables to temporarily hold data as it is being transformed and ultimately loaded to its destination [5]. Figure 3 is illustrating common ETL process flow.

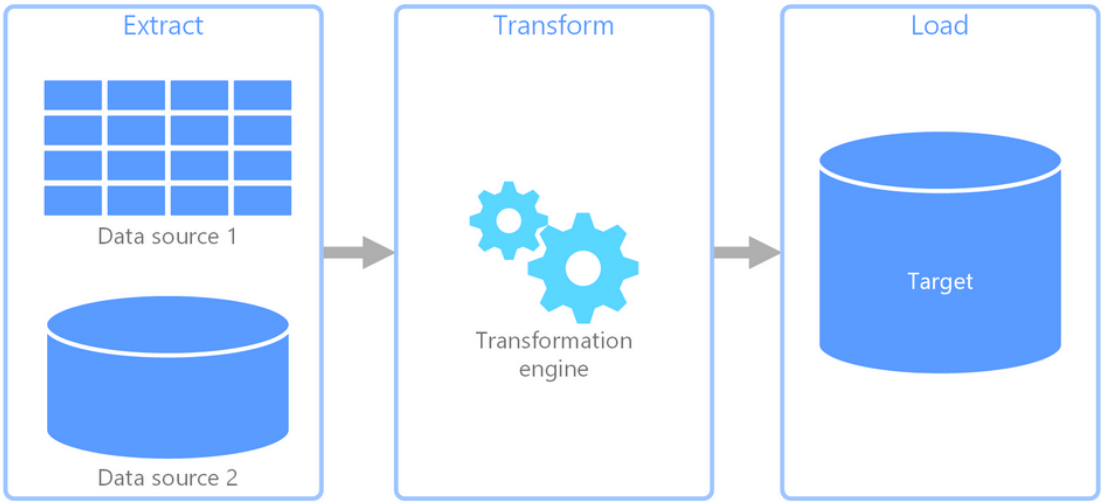

Figure 3. ETL process flow [5, Figure 1]

The extract process includes selecting the required data and extracting it from the source. After extraction depending on the architecture it can be loaded to staging area or directed straight to the transformation process. At that next step data need to be transformed to make data compatible with the structure of the target database. Final process loads transformed data for target system.

There is also an alternative type of ETL processes – ELT. ELT allows transformations to occur after data is loaded to the target system. Figure 4 is illustrating alternative ELT process flow.

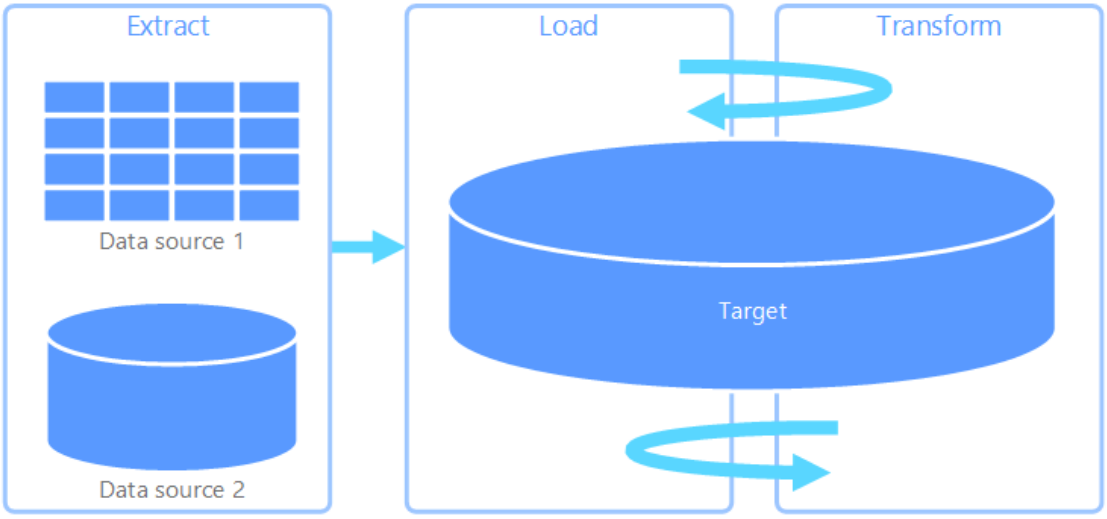

Figure 4. ELT process flow [5, Figure 2]

Extract, load, and transform (ELT) differs from ETL solely in where the transformation takes place. In the ELT pipeline, the transformation occurs in the target data store. Instead of using a separate transformation engine, the processing capabilities of the target data store are used to transform data. This simplifies the architecture by removing the transformation engine from the pipeline [5].

2.2.1 Performance of the ETL processes in Data Warehouse

As part of the general ETL workflow performance of each ETL process is crucial for common stability and efficiency. For example, simple movement of 1TB of data between a Data Store and Data Warehouse (which use magnetic disks with a typical 200MB/s) takes 2.7 hours by an ETL process. Resource consumption and the overall processing time are further increased by complex tasks executed within the process, including integrating, cleaning, and de-duplicating data [6].

The problem behind the performance lay in the constant growth of the data amount. As new requirements and changes increase amount of data old satisfying solution may become poorly performed and cause delays in the whole system. To achieve acceptable performance each process should be periodically reviewed on performance issues and optimized if needed. As a criterion for periodical review and optimization ETL best practices that are accepted in organization should be used [7].

2.3 Data Quality

Using data as a valuable asset is only effective if data itself is reliable and trustworthy. Data is of high quality to the degree that it meets the expectations and needs of consumers. That is, if the data is fit for the purposes to which they want to apply it. It is low quality if it is not fit for those purposes [3, p427]. Data quality is a critically important and underestimating or unconsciously ignoring that fact may cost an organization a lot. According to the research done by IBM in 2016 economy of the United States loses 3.1 billion US dollars yearly because of decisions made from low-quality data [8]. Many of the costs thereby are hidden and indirect and therefore hard to measure. There is also a risk that low-quality data may be misunderstood, and decision made from that could be wrong.

Organization only get value from data if it is high quality – it is accurate, up to date, has no or not relevant gaps, complete, consistent, and relevant. The corresponding benefits of high data quality includes: [3, p26]

- Improved customer experience
- Higher productivity
- Reduced risk
- Ability to act on opportunities
- Increased revenue
- Competitive advantage gained from insights from customers, products, processes, and opportunities
- Correct regular reporting

According to study "A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing" made by Ranjit Singh and Dr. Kawaljeet Singh, data quality issues can arise at any stage of data warehousing: in data sources, in data integration and profiling, in data staging, in ETL and database modeling [9] (see Figure 5).

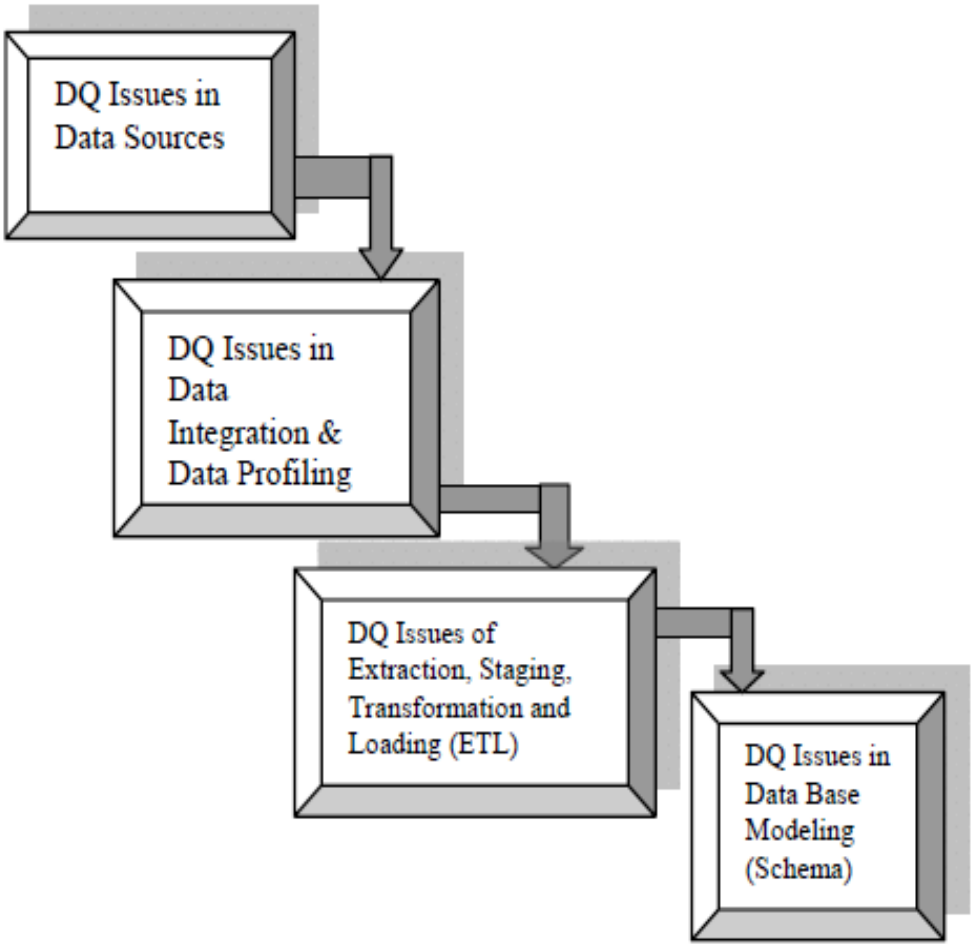

Figure 5. Stages of Data Warehouse Susceptible for DQ Problems [9, Figure 3]

Quality of data can be compromised depending upon how data is received, entered, integrated, maintained, processed (Extracted, Transformed and Cleansed) and loaded. Data is impacted by numerous processes that bring data into your data environment, most of which affect its quality to some extent. All these phases of data warehousing are responsible for data quality in the data warehouse [9].

2.3.1 Data Quality Management and data profiling

Data quality management (DQM) refers to planning, implementation and control of activities that apply quality management techniques to data, to assure it is fit for use of data consumers. Formal data quality management is similar to continuous quality management for other products. It includes managing data through its lifecycle by setting standards, building quality into the process that create, transform, and store data, and measuring data against standards [3, p424].

Data profiling is one of the key processes of data quality management. Data profiling is an assessment that uses a toolbox of business rules and analytical algorithms to discover, understand and potentially expose inconsistencies in data. This knowledge is then used to improve data quality as an important part of monitoring and improving the health of these newer, bigger data sets [10].

Data profiling involves [11]:

- Collecting descriptive statistics like min, max, count and sum.
- Collecting data types, length, and recurring patterns.
- Tagging data with keywords, descriptions, or categories.
- Performing data quality assessment, risk of performing joins on the data.
- Discovering metadata and assessing its accuracy.
- Identifying distributions, key candidates, foreign-key candidates, functional dependencies, embedded value dependencies, and performing inter-table analysis.

3 Current state of the DQM Service Component

In this chapter there will be given an overview of current state of the DQM Service Component and brought out main problems that are currently actual.

3.1 General description

Data Warehouse DQM Service Component (hereinafter DQM component) is a custom data profiling tool that is used on daily basis to validate loaded data and perform assessment of quality characteristics. DQM Service Component is built of several parts:

- Oracle Data Integrator tool package (hereinafter ODI_DQM_PROCESS)
- Java-executable file (hereinafter JAR_DQM_EXECUTOR)
- Validation scripts (stored in dedicated EDW schema *DQMRULE*)
- Multiple EDW schemas for storing metadata, validation results, error messages etc.

DQM Service Component is used for different validations by various data consumers but for each consumer target and result of validation would be individual and depend on the input parameters. To use DQM Component for a particular set of parameters new unique ODI_DQM_PROCESS should be created in Oracle Data Integrator tool. Execution of the ODI_DQM_PROCESS can be performed manually on ODI agent server or automatically on periodical basis through the workflow automation tool (Scheduler).

3.1.1 Use-Case view

To represent a high-level functionality and how user is handling the component author describes use cases using UML Use-Case diagram.

A use case is a written description of how users will perform tasks on within the system. It outlines, from a user’s point of view, a system’s behaviour as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal, and ending when that goal is fulfilled [12].

Based on the information collected from the current documentation and DQM component users author created a superordinate use-case ‘Validate data’. The result is presented in Table 1 as a use-case description and on Figure 4 as a use-case diagram.

Table 1. Superordinate Use Case ‘Validate data’

Use case	Validate data
Active Actor	Data Steward, Scheduler
Passive Actor	DQM Database
Use case overview	1. Actor initiates the execution of the Component and passes required information for validation (validation reference, period, country context etc.). 2. Executable performs data validation a. Executable runs validation scripts (superordinate use-case ‘Validate data’) b. Executable logs execution flow (subordinate use-case ‘Log execution’) c. Executable registers validation results in dedicated EDW schema *DQMResult* (subordinate use-case ‘Register results’)
Alternative flow	If any of the validation scripts fails, executable raises an alert, sends an error notification email, and proceeds the execution (subordinate use-case ‘Alert error’).
Trigger	Data validation is initiated manually by Data Steward or automatically by Automated Workload Scheduler

In the below use case diagram (see Figure 6), there are three actors - Data Steward, Scheduler and DQM DB and four use cases – superordinate use case ‘Validate data’ and three subordinate use cases ‘Register results’, ‘Log execution’ and ‘Alert error’.

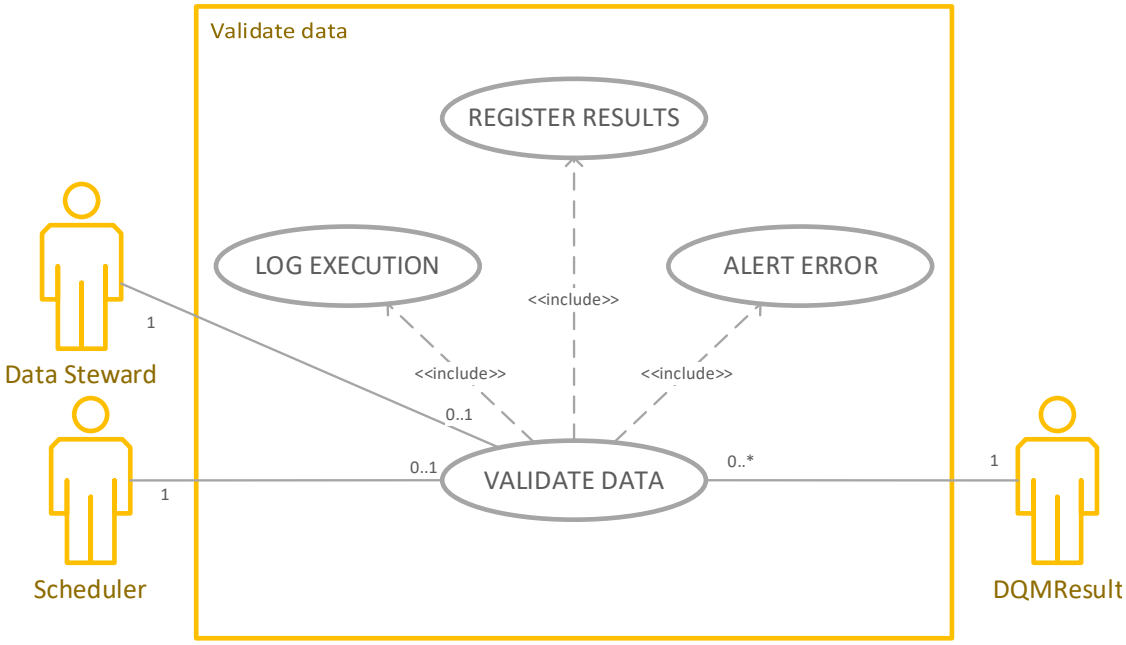

Figure 6. EDW architecture with a Staging Area and Data Marts

Both Data Steward and Automated Workload Scheduler (Scheduler) are active actors who may initiate the validation process. *DQMResult* is a passive actor - it stores validation results in dedicated EDW *DQMResult* schema for further use of interested parties.

3.1.2 Oracle Data Integrator tool package

Oracle Data Integrator (ODI) features an active integration platform that includes all styles of data integration: data-based, event-based, and service-based. ODI unifies silos of integration by transforming large volumes of data efficiently, processing events in real time. It also provides data integrity control features, assuring the consistency and correctness of data [13].

Being part of daily execution routine data validations are executed as a regular ETL process. To do that for each implementation of the Component should be created a unique ODI_DQM_PROCESS. Within the ODI_DQM_PROCESS there are certain parameters (e.g. Validation_ShortName, Country_ShortName and System_Version_No, etc.) need to be defined to make each implementation unique and aim at the target data set.

Each ODI_DQM_PROCESS is made up of a sequence of steps organized into an execution order. The flow of the activities within the ODI_DQM_PROCESS is described on Figure 7.

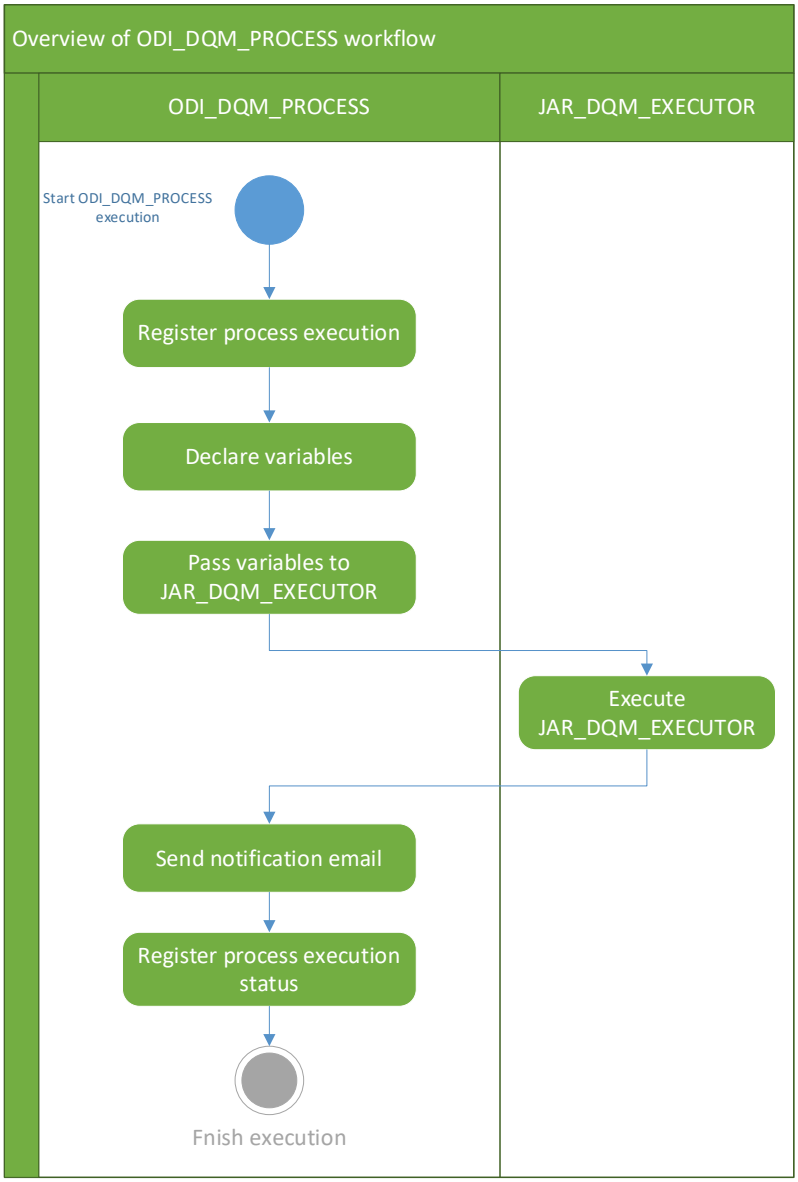

Figure 7. Activity diagram of ODI_DQM_PROCESS execution

Series of initial steps is responsible for preparing mandatory parameters (e.g. period, country context etc.) for further package execution. When those mandatory parameters are prepared, they are passed to the step that executes JAR_DQM_EXECUTOR that performs actual validation activity. After validation is finished ODI_DQM_PROCESS finalizes execution by logging it and performing required preparation for next execution (status and period bookmarks update, email notification).

3.1.3 Java-executable file

JAR_DQM_EXECUTOR is responsible for actual data validation processing. It receives required parameters from ODI_DQM_PROCESS package and depending on theirs values prepares and performs validations. The flow of the activities within the ODI_DQM_PROCESS is described on Figure 8.

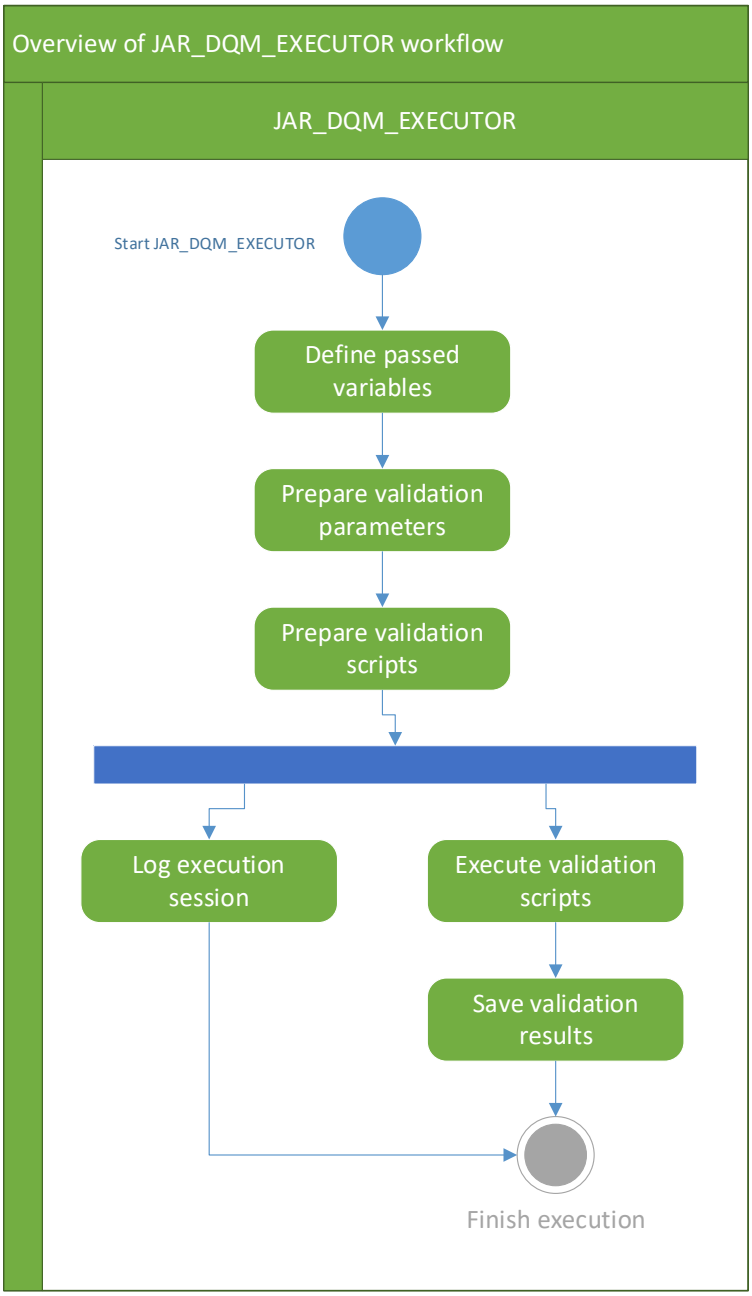

Figure 8.Activity diagram of JAR_DQM_EXECUTOR execution

At start of the execution JAR_DQM_EXECUTOR declares variables and sets values from passed parameters. Based on these variables it queries for validation scripts from dedicated EDW schema *DQM.RULE* and prepares them for the execution. Throughout

the whole process it logs all the execution in the EDW schemas *DQMRESULT.RUN_PARAM* for the execution session parameters' values and *DQMRESULT.RUN* for the execution on query level. After validation scripts are prepared JAR_DQM_EXECUTOR performs the validation itself, saves and summarizes results of the validation and passes the status of the execution and summary for the email notification to ODI_DQM_PROCESS.

3.2 Outcome of the Component analysis

As a result of the Component analysis author created description of the key components of the DQM Service component, described its workflow and based on that information updated existing architectural description. Additionally to that author prepared the introduction tutorial for new users of the component. In the scope of the Component analysis author also performed a brief review of third-party alternative tools for data quality assessment.

3.2.1 Alternative data quality tools

There are several data quality tools currently available worldwide that serve the aim of data quality processing. Some of them are standalone solutions and others are integrated into more complex data management systems. Below author gives a short overview of three existing solutions and their functionality.

3.2.1.1 AB Initio

Ab Initio is an American company specialises in high-volume data processing applications. Its software platform 'Ab Initio' is built of several built-in components. Additionally to data transformations, selection/filtering, de-duplication and joining/merging components it also includes data quality processing component.

Data quality processing component includes the following functionality:

- A subsystem that detects and possibly corrects data quality problems (using user defined validation rules)
- A data quality reporting system (integrates with the rest of an enterprise's metadata, data quality metrics and error counts, and data profile results)

- An issue reporting database

This DQ processing component is typically run as part of existing applications. If an application has been built with Ab Initio, the DQ component can easily be plugged into it. For applications not built with Ab Initio, the DQ processing component has to be explicitly invoked. The DQ component can also be implemented as an independent job that sources data directly [14].

3.2.1.2 Informatica Data Quality tool

Informatica Data Quality tool is developed by an American company Informatica and it is built upon a cloud-based Data Management Platform. Informatica Data Quality tool includes the following functionality:

- Pre-built business rules and accelerators and re-use common data quality rules
- Manage and transform data with data standardization, validation, enrichment, de-duplication, and consolidation capabilities
- Review, correct, and approve exceptions throughout the automated process
- Profile data and perform iterative data analysis to uncover relationships and better detect problems

Data Quality tool may also integrate plug-ins and may be integrated to other platforms (Informatica PowerCenter) [15].

3.2.1.3 Datamartist

Datamartist developed by a Canadian company nModal Solutions. Datamartist is a flexible, visual, data profiling and data transformation tool. Datamartist includes the following functionality:

- Create data profiling and transformation tasks that detect data quality issues
- Schedule automated runs to write the results with timestamps to allow tracking
- Monitor trends in data quality and take action quickly when new issues arise

Datamartist is a visual tool that is installed on a user's workstation and can access multiple data sources, (SQL Server, MySQL, Oracle, ODBC, Text files and Excel) [16].

4 Performance enhancement

One of the main problems behind the DQM Service component is its poor performance. Throughout the whole period of using DQM component it was noticed that executions take abnormally much time and EDW server resources (detailed information regarding poorly performed processes is introduced in section 4.2.2 of the thesis). To investigate the performance issues author reviewed performance in two stages: review of the java component source code and process CPU consumption of the validation queries.

4.1 JAVA Component source code review

As it was noticed that execution time of the DQM processes vary depending on the validated data set size so author assumed that the performance issue may be caused by the database multiple connections. A common problem with Java applications that access a database is that they sometimes access the database too often, resulting in long response times and unacceptable overhead on the database [17].

4.1.1 Source code review results

As a result of the review it was discovered that potential issue that affected performance is an implementation of EDW connection within the result submission functionality. The flow of the activities within the result submission functionality is described on Figure 9.

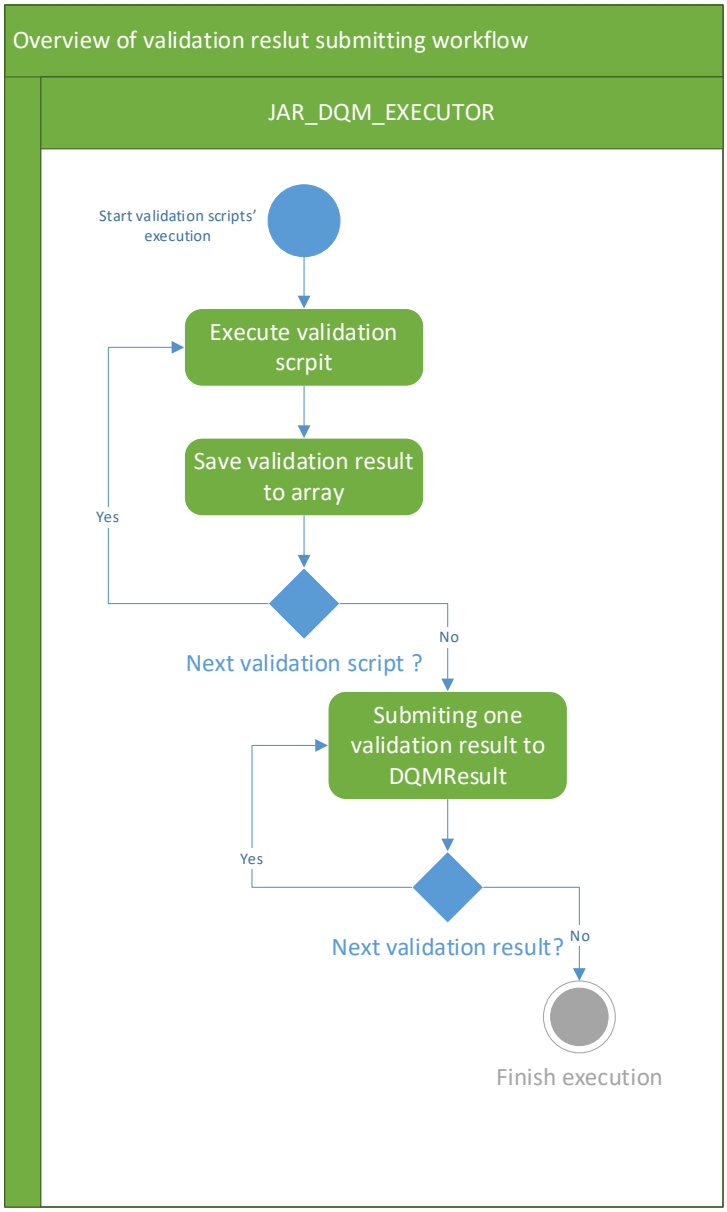

Figure 9. Activity diagram of validations execution in JAR_DQM_EXECUTOR

After the execution of each validation script the results of validations are stored in an array. Submitting results is performed by parsing the array one by one and for each iteration of result submission new connection is created. As creating multiple connections for passing results is causing unnecessary load for the EDW and JAR_DQM_EXECUTOR itself this part of code had to be refactored.

4.1.2 Solution for connection issue

The issue of multiple connections can be solved by applying Teradata Fastload utility. Fastload is parallel load utility used to load data in bulk mode to Teradata Database. This

command-driven utility feeds one table per job. Fastload can be used interactively or in batch mode using scripts [18].

Fastload was developed to load millions of rows into empty Teradata tables so it expected to be fast. Fastload will create a Teradata session for each AMP to maximize parallel processing. This gives good performance in loading data. [19]

There are several reasons why Fastload is more efficient for data loading than regular load functionality [19]:

1. No Secondary Indexes are allowed on the Target Table
2. No Referential Integrity is allowed:
3. No Triggers are allowed at load time
4. Duplicate Rows (in Multi-Set Tables) are not supported
5. No AMPs may go down (i.e., go offline) while Fastload is processing
6. No more than one data type conversion is allowed per column

According to Teradata Documentation. Fastload Reference [18] to implement Fastload utility there are several actions had to be done:

1. Creating new table in Staging area for passing validation results from file
2. Refactor source code to enable DQM component to:
    a. Save validation results to file
    b. Execute Fastload to load data from file to Staging area
    c. Insert data from staging area to *DQMResult* schema
3. Disabling of previous solution

Figure 10 is illustrating proposed workflow of the result submission functionality.

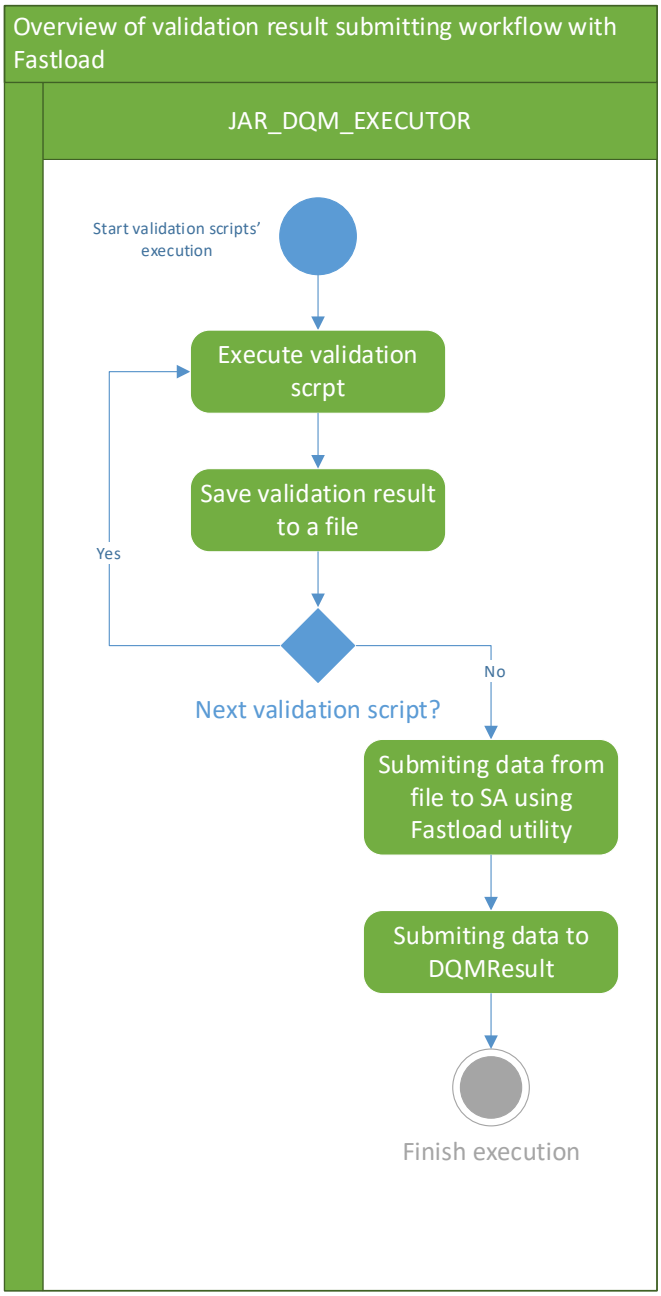

Figure 10.Activity diagram of validations execution in JAR_DQM_EXECUTOR after refactoring

After defining required steps for the change author also created a prototype in development environment of EDW for further estimation of possible improvements in DQM component performance.

4.1.3 Comparison of the validation results

For testing purposes author compared the total consumed CPU and processing time of execution of one of the DQM processes with original result submission functionality and with applied Fastload utility using same amount of source data. Results brought out in Table 2.

Table 2. Comparison of the executions of the Component before and after source code refactoring

DQM Process Name	Loading type	Total CPU Impact (CPU-second)	Processing time
DQM_PROCESS_1	ORIGINAL	3,170.88	0:09:29
DQM_PROCESS_1	FASTLOAD	1,659.46	0:00:51

As it was expected execution of the DQM process with Fastload utility applied showed a better performance in terms of CPU consumption (48% less CPU-seconds than original solution) and in term of processing time - 8 minutes 38 seconds less than the original solution.

4.1.4 Current state of the change

Results of the applied changes were demonstrated to stakeholder and it is decided to proceed with updating of the Fastload utility functionality to the production environment. Currently the updated component is in testing phase and will be applied after the required testing and validations of the change are finalized.

4.2 Review of the CPU consumption of the validation scripts

Data warehouse performance is dependent on a number of factors including the nature of the queries being run against the physical database, the hardware and software resources available and the underlying physical distribution of the data in the database [20]. In the scope of the performance analysis author review the issues that may be caused by validation scripts.

Analyzing performance of the validation scripts is more complex task comparing to executor source code analysis as it comes from the perspective of big amount of the scripts and from their consistency. According to frequently changing business requirements validation complexity and amount may vary (in most case it is constantly increasing). For that reason, scripts performance analysis is constantly ongoing process.

4.2.1 Performance review results

To review the performance author was using the collected metadata of the executed validation scripts stored in dedicated system information schema of the EDW (*sys_info* schema). As a metric total CPU impact is used to estimate the performance of the script.

CPU impact is chosen for a metric as there is a company standard for evaluating performance based on CPU consumption limits for a single query [7].

Author figured out that there were 20 DQM processes that exceeded the limit of allowed 10000 CPU-seconds [7]. Analysis of poorly performed scripts brought out the issue of the one script templates used by all DQM processes for summarizing the validation and reporting the results. Problem of poor performance was related to the fact that indexed parameters used in template was not enough for sufficient select request. Using of additional indexed input parameter "*query_id*" helped to reduce CPU consumption.

After implementation of the changes in production environment new results were collected for the same processes (see Table 3).

Table 3. Comparison of CPU usage of the DQM processes validation scripts before and after update

DQM Process Name	Total CPU before script enhancement (CPU-seconds)	Total CPU after script enhancement (CPU-seconds)
DQM_PROCESS_1	20170.44	118.98
DQM_PROCESS_2	19818.74	8,263.18
DQM_PROCESS_3	19615.76	216.35
DQM_PROCESS_4	16030.79	79.37
DQM_PROCESS_5	14698.66	171.5
DQM_PROCESS_6	12284.36	63.74
DQM_PROCESS_7	11867.97	107.11
DQM_PROCESS_8	11769.05	60.23
DQM_PROCESS_9	11640.83	4,248.00
DQM_PROCESS_10	11633.91	123.6
DQM_PROCESS_11	11618.6	50.23
DQM_PROCESS_12	11533.26	56.1
DQM_PROCESS_13	11508.48	964.02
DQM_PROCESS_14	11043.56	159.12
DQM_PROCESS_15	10470.89	154.62
DQM_PROCESS_16	10447.79	1,703.68
DQM_PROCESS_17	10402.07	108.29
DQM_PROCESS_19	10318.57	714.55
DQM_PROCESS_19	10244.96	90.25
DQM_PROCESS_20	10168.7	115.58

Applied script enhancement made a significant improvement of CPU consumption for all reviewed processes decreasing it in average for 93.8%. Additionally, total processing time for the DQM processes was compared.

Table 4. Comparison of the execution time of the DQM processes before and after script update

DQM Process Name	Processing time before validation script enhancement (hh:mm:ss)	Processing time after validation script enhancement (hh:mm:ss)
DQM_PROCESS_1	00:06:56	00:01:54
DQM_PROCESS_2	00:06:50	00:01:27
DQM_PROCESS_3	00:06:08	00:01:03
DQM_PROCESS_4	00:01:55	00:00:42
DQM_PROCESS_5	00:07:54	00:01:27
DQM_PROCESS_6	00:05:27	00:00:17
DQM_PROCESS_7	00:05:02	00:02:47
DQM_PROCESS_8	00:11:47	00:01:12
DQM_PROCESS_9	00:16:10	00:03:44
DQM_PROCESS_10	00:09:51	00:00:22
DQM_PROCESS_11	00:02:08	00:00:17
DQM_PROCESS_12	00:04:36	00:03:38
DQM_PROCESS_13	00:08:45	00:40:56
DQM_PROCESS_14	00:07:18	00:01:32
DQM_PROCESS_15	00:01:53	00:00:51
DQM_PROCESS_16	00:17:23	00:05:18
DQM_PROCESS_17	01:50:41	01:03:20
DQM_PROCESS_19	00:03:24	00:00:27
DQM_PROCESS_19	00:07:18	00:01:09
DQM_PROCESS_20	00:04:47	00:00:34

As for CPU consumption processing time spent on process execution also significantly decreased.

4.2.2 Current state of the change

To improve overall validation script quality and to avoid further after-release script corrections it was proposed to use new testing requirements for all new DQM processes and validation scripts. Previously validation scripts could be changed directly in EDW Schema *DQMRule.Query* table bypassing general release process and testing. From now on all validation scripts are a tested and are not allowed to production environment unless performance tests are passed.

5 Summary

The aim of the thesis was to update existing Architectural Description of the Data Quality Management Component of Enterprise Data Warehouse environment in Financial Institution N., and to do its performance analysis in EDW. For that purpose, author of the thesis performed a Component analysis of DQM component and reviewed performance issues.

As a result of the thesis author created description for the key parts of the DQM service component, described its workflow and based on that information updated existing architectural description.

Author also investigated the performance issues of the Component and found out that poor performance was caused by bad solution of validation result submission functionality in Java-executable file's sources code and insufficient use of indexes in validation summarizing script. Author defined activities for refactoring Java-executable file and created a prototype to test it. Testing results showed that prototype used 48% less CPU-seconds than original solution. Testing results were presented to stakeholders and it was decided to proceed with the change to the production environment as performance has significantly increased. Also, validation summarizing script got updated by adding additional indexed parameter into common validation summarizing script template. Applied script enhancement decreased CPU consumption in average for 93.8%. This change was applied to production environment right after testing activities were done.

To avoid further need for validation scripts' performance problems review, it was decided that the release process for validation scripts must be updated. According to new release process all validation scripts must pass testing for performance issues.

After the introduced changes in Java-executable file sources code and in validation summarizing script were applied the overall performance of the DQM processes has drastically improved.

References

[1] "Unified Modeling Language", Object Management Group, 12.2017. [Online], Available: https://www.omg.org/spec/UML/About-UML/

[2] Paramita Ghosh, "The Evolution of Data as an Asset", 9.12.2020. [Online]. Available: https://www.dataversity.net/the-evolution-of-data-as-an-asset/#

[3] "DAMA-DMBOOK. Data Management Body of Knowledge. Second edition", Technics Publications, New/Jersey, 2017.

[4] "Oracle Database Data Warehousing Guide. Introduction to Data Warehousing Concepts", Oracle Corporation, 2017. [Online], Available: https://docs.oracle.com/database/121/DWHSG/concept.htm#GUID-C3AD27A3-970A-442D-8B18-86B79D643F25

[5] "Azure Data Architecture Guide", Microsoft Corporation, 2.12.2018. [Online], Available: https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl

[6] M. Bodziony, S. Roszyk, R. Wrembel, "On Evaluating Performance of Balanced Optimization of ETL Processes for Streaming Data Sources", DOLAP 2020, Copenhagen, Denmark, 30.03.2020.

[7] "Performance evaluation guide of EDW processes", Financial Institution's N. Internal Documentation, Tallinn, Estonia, 2014.

[8] T. C. Redman, "Bad Data Costs the US 3 Trillion Per Year", Harvard Business Review Digital, 2016. [Online], Available: https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year

[9] R. Singh, Dr. K. Singh, "A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing", International Journal of Computer Science Issues, 2.05.2010. [Online], Available: http://www.ijcsi.org/papers/7-3-2-41-50.pdf.

[10] J. Bauman, “What is data profiling and how does it make big data easier?”, SAS Insights. [Online], Available: https://www.sas.com/en_us/insights/articles/data-management/what-is-data-profiling-and-how-does-it-make-big-data-easier.html#close

[11] “Building an Analytics Stack: A Guide”, Panoply. [Online], Available: https://panoply.io/analytics-stack-guide/data-profiling-best-practices/

[12] “Use Cases”, U.S. General Services Administration, 2006. [Online], Available: https://www.usability.gov/how-to-and-tools/methods/use-cases.html

[13] “Fusion Middleware Getting Started with Oracle Data Integrator”. Oracle Corporation. [Online], Available: https://docs.oracle.com/cd/E17904_01/integrate.1111/e12641/overview.htm#ODIGS111

[14] “Data Quality”, AB Initio, 2020. [Online], Available: https://www.abinitio.com/en/system/data-quality

[15] “Cloud Data Quality”, Informatica, 2021. [Online], Available: https://www.informatica.com/products/data-quality/cloud-data-quality-radar.html

[16] “A Visual easy to use Data Profiling tool”, nModal Solutions Inc., 2021. [Online], Available: http://www.datamartist.com/a-visual-easy-to-use-data-profiling-tool

[17] S. Haines, “Top 10 most common Java performance problems”, AppDynamics, 2014. [Online], Available: www.rockvalleycollege.edu%2Fwebadmin%2Fupload%2FTop-10-Java-Performance-Problems.pdf&usg=AOvVaw0h2zxNIguBbm2PArV1plZZ

[18] “Teradata Documentation. Fastload Reference”, Teradata, 2021. [Online], Available: https://docs.terada.com/r/r_6Z4JwVMhANtZFCIKEx7Q/r6v__Jr1IohFGHmWlqeidA

[19] “Fast Load”, Teradata Wiki, 2013. [Online], Available: https://www.teradatawiki.net/2013/10/teradata-utilities-Fastload.html

[20] A. Taylor, “Practical Advice for Managing Poor Warehouse Performance”, The Data Administration Newsletter, 2.09.1998. [Online], Available: https://tdan.com/practical-advice-for-managing-poor-warehouse-performance/4262

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Konstantin Dmitrijev

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Analysis of Data Quality Management service component in Financial Institution’s Data Warehouse”, supervised by Nadežda Furs
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

17.05.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Validation result summary for DQM process

***This is an automatically generated email, please do not reply! ***

The JAR_DQM_EXECUTOR package verified validation results at

(Process: DQM_PROCESS_1)

Validation results were verified (using predefined expected result values).

Validation results verification summary at validated period

_____

Total count of verified rules = 1
Count of failed rules = 1
Count of not-failed rules = 0
Total count of verified records = 15
Count of failed records = 9
Count of not-failed records = 6

Validation results changes in time verification summary at validated period

_____

Total count of verified rules = 1
Count of failed rules = 1
Count of not-failed rules = 0
Total count of verified records = 6222
Count of failed records = 18
Count of not-failed records = 6204

Query_id: 12706 failed in verification at validated period
Query_id: 12704 failed in change in time verification at validated period
Query_id: 12701 failed in change in time verification at validated period

Appendix 3 –Source code of Fastload prototype added to DQM component source code

a) Creating new table in Staging area for passing validation results from file

```
.SET SESSION CHARSET "ASCII";
.LOGMECH KRB5;
.LOGON */hostname\*;
DATABASE SA ;

drop table  SA.SSESSION_DQM;
drop table SA.SSESSION_DQM_ERR1;
drop table SA.SSESSION_DQM_ERR2;
CREATE TABLE SA.SSESSION_DQM
(
  Query_Column_sequence_no INTEGER NOT NULL
,Result_Row_Sequence_No  INTEGER NOT NULL
,Query_Column_Result_Cnt INTEGER
,Query_Column_Result_Date DATE FORMAT 'YYYY-MM-DD'
,Query_Column_Result_Amt DECIMAL(18,2)
,Query_Column_Result_Desc VARCHAR(100) CHARACTER SET UNICODE NOT
CASESPECIFIC
,Query_Id INTEGER NOT NULL
,Run_Id INTEGER NOT NULL
,Validation_Rule_Id INTEGER
,Result_Data_Type_Code SMALLINT NOT NULL COMPRESS (1 ,2 ,3 ,4 ,5 )
,Database_Name VARCHAR(30) CHARACTER SET UNICODE NOT CASESPECIFIC
,Table_Name VARCHAR(30) CHARACTER SET UNICODE NOT CASESPECIFIC
,Measure_Ind CHAR(1) CHARACTER SET UNICODE NOT CASESPECIFIC NOT NULL
DEFAULT 'Y' COMPRESS ('N','Y')
,Column_Name VARCHAR(30) CHARACTER SET UNICODE NOT CASESPECIFIC)


;

SET RECORD VARTEXT '|';
DEFINE
 C1 (VARCHAR(1000))
 ,C2 (VARCHAR(1000))
,C3 (VARCHAR(1000))
,C4 (VARCHAR (1000))
,C5 (VARCHAR(1000))
,C6 (VARCHAR(1000))
,C7 (VARCHAR(1000))
,C8 (VARCHAR(1000))
,C9 (VARCHAR(1000))
,C10 (VARCHAR(1000))
,C11 (VARCHAR(1000))
,C12 (VARCHAR(1000))
,C13 (VARCHAR(1000))
,C14 (VARCHAR(1000))

FILE=Y: \DQM\datafile_for_1283085.txt;
RECORD 1;
BEGIN LOADING SA.SSESSION_DQM
    ERRORFILES SA.SSESSION_DQM_ERR1, SA.SSESSION_DQM_ERR2;
```

```
INSERT INTO SA.SSESSION_DQM
(
  Query_Column_sequence_no
 ,Result_Row_Sequence_No
 ,Query_Column_Result_Cnt
,Query_Column_Result_Date
 ,Query_Column_Result_Amt
 ,Query_Column_Result_Desc
,Query_Id
 ,Run_Id
 ,Validation_Rule_Id
 ,Result_Data_Type_Code
 ,Database_Name
 ,Table_Name
 ,Measure_Ind
,Column_Name
)
VALUES (:c1,:c2,:c3,:c4,:c5,:c6,:c7,:c8,:c9,:c10,:c11,:c12,:c13,:c14);
END LOADING;

LOGOFF;
```

b) Saving validation results to file

```
private void writeToFile(String filePath, String[] dataRows) {
   try {
      File file = new File(filePath);
      if (file.createNewFile()) {
         System.out.println("File created at: " + file.getPath());
      } else {
         System.out.println("File already exists");
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
   try {
      BufferedWriter bw = new BufferedWriter(new FileWriter(filePath,
true));
      for (int i = 0; i < dataRows.length; i++) {
         if (i != 0) bw.newLine();
         bw.write(dataRows[i]);
      }
      bw.close();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

c) Executing Fastload to load data from file to Staging area

```
private void executeFastload() {
   String command = "Fastload < \"Y:\\
DQM\\dev\\dqm_validation\\resources\\party_script.fl\"";
   try {
      Runtime rt = Runtime.getRuntime();
      Process process = rt.exec("cmd.exe /c SET GUILOGON=NO && " +
```

```
command);
      BufferedReader stdInput = new BufferedReader(new
InputStreamReader(process.getInputStream()));
      BufferedReader stdError = new BufferedReader(new
InputStreamReader(process.getErrorStream()));
      // Read the output from the command
      System.out.println("Here is the standard output of the
command:\n");
      String s = null;
      while ((s = stdInput.readLine()) != null) {
         System.out.println(s);
      }
      // Read any errors from the attempted command
      System.out.println("Here is the standard error of the command
(if any):\n");
      while ((s = stdError.readLine()) != null) {
         System.out.println(s);
      }
   } catch (IOException e) {
      e.printStackTrace();
   }
```

d) Inserting data from staging area to *DQMResult* schema

```
private void InsertFromOsaToDqmResult(Connection conn, String
sessioonid, String insertStatement) throws Exception {
   String insertQuery = String.format(insertStatement);

   PreparedStatement stmt = null;

   try {
      stmt = conn.prepareStatement(insertQuery);
      //collectLog("Executing InsDetailQueryColumnResult:");
      //collectLog(InsDetailQueryColumnResult);
      stmt.executeUpdate();

   } catch (Exception ex) {
      collectLog("Insert from SA to DQMRESULT failed"
            + ex.getMessage());
      throw new Exception(ex.getMessage());
   } finally {
      if (stmt != null) {
         stmt.close();
      }
   }
}
```