FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Solving Data Quality Problems in Data Warehousing by Means of Data Exchange Techniques

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Jan Jozaf
Matrikelnummer 0226013

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof.Dr. Reinhard Pichler

Wien, 16.02.2015

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Solving Data Quality Problems in Data Warehousing by Means of Data Exchange Techniques

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Jan Jozaf
Registration Number 0226013

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof.Dr. Reinhard Pichler

Vienna, 16.02.2015

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Jan Jozaf
Sedmokraskova 6, 821 01 Bratislava, Slovakia

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Acknowledgements

First of all, I would like to thank my advisor, Prof. Reinhard Pichler, for his guidance with all good suggestions and advices. I need to thank also for his patience and willingness to support me even if the completing of the thesis took longer time.

Next, I offer my sincere appreciation for the learning opportunities to all teachers whom I met during the university study. They showed me the value of the knowledge which currently builds the base in everyday life.

Finalization of this thesis would also not be possible without the continuing support of my family. I am very grateful to them, for giving me all the care and providing their continued encouragement during the study. They gave me every time new impulses to continue when it was necessary for the completing of this thesis.

Abstract

Since the concept of data warehousing started to play an important role for bigger companies in the past few years, consequently data quality has became an essential part in the process of collecting, integrating and delivering trust-worthy data to business people. During the time, the subject of data quality has been examined in many theoretical research papers and many of these findings have been successfully applied in practice. However, new challenges have recently appeared, causing bigger demands for more detailed data quality checks including the processing of constantly increasing amount of data. Taking into account this new demands in the overall data quality process, the data validation has became considerably slower than before. The individual data validation checks have to be optimized, so they can achieve better performance.
This problem area is the main focus of this thesis and could be described more profoundly with the following question: "How can we validate the correctness of data in a high-performance way, to guarantee the on-time processing of daily operations within data warehouses"
Although the currently used techniques and tools for data validation deliver significantly better performance in comparison to previous years, they are still not sufficient to validate the continuously increasing data volume in acceptable time. In our effort to find possible improvements in the currently used data validation algorithms we looked at the recent research results in the field of the data exchange. The algorithmic similarity between data validation and data exchange could serve as an inspiration for examination of research results in the data exchange area, under the consideration that they are put into the role of data validation.
In this thesis, we present and examine possibilities of such an implementation of the data exchange solution with the focus on processing of large amount of data quality checks. The aim is to find an algorithm which can minimize the amount of data checks to be executed but preserves its original declarative definition. The approach that was chosen for this thesis, takes a detailed look at the research that has been done in data exchange and tries to find for them the best possible applicability, which should bring added value in the data quality field. This added value is measured on the performance level with respect to data quality standard implementation. As a result of this work, we present the findings and recommendations, but also the possible restriction by the use of

data exchange in the data quality area. The presented results are based on a concrete implementation of the proposed algorithm in several data validation scenarios. The source code of the implementation is publically available on https://code.google.com/p/data-quality-with-data-exchange/source/browse/.

Kurzfassung

Seitdem das Konzept von Data Warehousing eine wichtige Rolle für größere Unternehmen zu spielen begonnen hat, wurde die Datenqualität zu einem wesentlichen Teil im Prozess der Erfassung, Integration und Bereitstellung von zuverlässigen Daten. Das Thema "Datenqualität" ist schon in vielen theoretischen Forschungsarbeiten und auch praktischen Implementierungen untersucht worden, wobei viele dieser Ansätze erfolgreich in der Praxis eingesetzt worden sind. Allerdings sind in den letzten Jahren mit neuen Anforderungen an immer detaillierte Datenqalitätsverifikationen und das steigende Datenvolumen neue Herausforderungen entstanden. Die meisten davon betreffen die Performanz der "Daten prüfenden" Prozesse. Die einzelnen Datenqualitätsverifikationen müssen optimiert werden, um eine Steigerung der Performance und damit eine Verringerung der Durchlaufzeit zu erreichen. Diese Problematik steht auch im Fokus dieser Diplomarbeit und ist mit folgender Kernfrage detaillierter ausgedrückt: "Wie kann man die Validierung der Korrektheit von Daten in einem hochperformanten Weg für die täglichen Abläufe in einem DataWarehouse gewährleisten?".
Die derzeit verwendeten Techniken und Werkzeuge für die Datenvalidierung liefern zwar eine deutlich bessere Performanz im Vergleich zu früheren Jahren, sie sind aber immer noch nicht ausreichend, um kontinuierlich steigende Datenvolumen zu überprüfen. Beim Suchen nach möglichen Verbesserungen in den jetzigen Datenqualitäts-Algorithmen helfen uns die letzten Forschungsergebnisse aus dem Bereich von Data Exchange. Die algorithmische Ähnlichkeit zwischen Datenvalidierung und Data Exchange könnte uns als Inspiration für die Untersuchung von Forschungsergebnissen in Data Exchange dienen. Vor allem, wenn wir die Data Exchange Algorithmen in der Rolle der Datenvalidierung einsetzen möchten.
In diese Diplomarbeit untersuchen wir die Möglichkeiten der Implementierung einer Data Exchange Lösung für die Datenqualität und die mögliche Performanz-Verbesserungen in Bezug auf die Standard-Implementierung innerhalb eines DataWarehouse Systems. Die Vorgehensweise, die für diese Diplomarbeit gewählt wurde, wirft einen detaillierten Blick auf die Forschung, die in Data Exchange gemacht wurde, und findet Möglichkeiten für ihre Umsetzung mit Mehrwert im Daten-Qualitätsbereich. Als Ergebnis dieser Arbeit präsentieren wir einerseits die Erkenntnisse und Empfehlungen und anderseits die möglichen Einschränkungen bei der Verwendung von Data Exchange im Daten-

Qualitätsbereich. Die präsentierten Ergebnisse basieren auf einer konkreten Umsetzung des vorgeschlagenen Algorithmus in mehreren Datenvalidierungsszenarien. Die Implementierung des Algorithmus ist unter dem Link https://code.google.com/p/data-quality-with-data-exchange/source/browse/ zu finden.

Contents

CHAPTER 1

Introduction

1.1 Data Quality

Data quality is an essential part of data warehouses with the primary aim to clean data from source systems and deliver trust-worthy data to business users. In addition to data cleaning, the process of delivering trust-worthy data might include supplementary processes like consolidation, validation or even unification of processed data. The complexity of the whole process is illustrated by the comparisons of different methodologies used in data quality process e.g. [1].

The core of the process consists of 2 main components: rules definition and regularly (ongoing) data checks based on the defined rules. Rules for data quality purposes could be of different nature and in general could be categorized into 3 categories: database integrity rules, match and merge rules and business rules [2]. The structure of the rules is either composed of isolated data checks or could consist of single isolated checks defined in a workflow or as an algorithm. Because of the high variability in the nature of the rules it is very difficult to find a unified solution for a general implementation of a data quality solution. The most problematic part in the practice implementations is the runtime performance, as the typical implementation usually consists of execution the data checks one by one in the defined order, without any optimization.

Companies which own a data warehouse meet very frequently performance problems of data quality solutions. Companies usually have big data warehouses with hundreds of data quality rules which should be processed in "time windows" on a large amount of data. In a typical data quality implementation is one data quality rule translated into one sql statement and the whole examined data amount has to be scanned once for every defined rule. Similarly every sql statement from a defined set of rules has to be executed using one sql statement. This has negative impact on the performance.

1.2 Data Exchange

Data exchange is the problem of transforming data structured under a schema (called source) into data structured under a different schema (called target) [3] by using schema mappings. Schema mapping defines the relationship between source and target schemas and thus specifies the transformation logic between the schemas [4].

The foundation of this research has been laid by the article: Data Exchange: semantics and query answering [5]. The concept presented in this article proposes an algorithm which could be used for the mapping/transformation of data from the source schema to the target schema under certain dependencies defined by source and target (STD) and target dependencies (TD). The motivation behind this research is the classical data exchange problem. This problem consists of a source schema filled with data; an empty target schema; a translation schema how to map data elements from the source schema to target schema; and possible constraints imposed by the target schema. A solution for this problem is a non-empty result-set which satisfies the above conditions, if such a result-set exists. This solution might be obtained by using various algorithms described in many research papers.

As the complexity of this problem could rise into enormous dimensions e.g. by defining dependencies with inequalities [6, 7] or using special operators like (!=, =>,<=) [3, 8], it is also important to investigate the computation time, in which the result set could be computed. This has been subject for examination in many research papers and still represents the subject for further research, as for now only a limited number of special cases has been investigated. During the research it has been proved that specific subsets of data exchange problems could be solved in polynomial time, nevertheless the research is going forwards.

1.3 Research Subject

Data quality appears to be a suitable field to discuss findings from data exchange, as there exists a belief, that in specific situations it could contribute to performance enhancements by the execution of data quality checks.

This thesis will study and examine the possibility for the implementation of research results from the data exchange area in the data quality area. The rationale behind this research is the fact that the data quality check process could be in a certain way considered as a data exchange problem. The set of data quality checks composes a data exchange setting. Data quality rules could be in data exchange terminology considered as source to target dependencies and target dependencies. Further, the universal solution and the core as the results from a data exchange problem, appear to be relevant by getting the "valid" records from data quality, those records which meet the data quality rules.

In this work we will try to examine these similarities. Firstly, we will put the algorithm for core computation into the role of data quality solution and evaluate its applicability. Next, we propose a small modification of this algorithm and prepare runtime evaluation against the standard used data quality algorithm. The evaluation bases on a practical implementation of this algorithm.

1.4 Method

At the beginning of this work we will theoretically examine typical cleaning and matching operators from the research area and as well from data quality tools used in practice. Following to this, we will introduce a typical data quality example, including its setup and process of sql generating from defined rules. Identified cleaning operators will be decomposed into sql statements and based on this example we will consider data exchange algorithms, which could be suitable for implementation in this example. Well-suited algorithms will be evaluated by different criteria as complexity, possibility of usage, time demand for result set computing and implementation difficulty. A selected subset of these algorithms will be implemented in the data quality field. Based on this implementation and the following evaluation it will be considered under which conditions the usage of data exchange algorithms in the data quality process is still reasonable and which performance improvements can this implementation bring.

1.5 Summary of Results

We will show that the general principles and knowledge from data exchange is also applicable for data quality checks. The core is the ultimate set of “valid“ records and the algorithms for its calculating could be also used in data quality. Similary as in data exchange, the problem of redundancy in universal solutions, leads to performance inefficiency in getting the “valid“ records in the data quality. Fortunately, the data quality checks adapted for the dimensional star resp. snowflake structure of data model in source systems, have only small potential to be a source for generating the redundancy. In the last chapter we will show a practical implementation of such algorithm in data quality area with the performance evaluation against the standard used algorithm.

1.6 Organisation

We will formally introduce general principles of Data Warehousing and Data Quality in Chapters 2 and 3. The foundations of the main research subject - Data Exchange - will be covered in Chapter 4. After exploring the foundations; the main contribution with the results is the topic of Chapters 5 and 6. Chapter 5 shows the similarities between

Data Exchange and Data Quality. These similarities are used in Chapter 6 to define a possible algorithm for Data Quality. Based on the result of Chapter 6 we provide a practical example of the implemented algorithm including timely efficiency in Chapter 7. We conclude this work and give an outlook to future work in Chapter 8.

CHAPTER 2

Principles of Data Warehousing

This chapter briefly describes the purpose of data warehousing, including the terminology and concepts, which are important for an understanding of subsequent chapters of this thesis.

2.1 Introduction to Data Warehouse

The idea of data warehousing was mentioned for the first time in a book by Bill Inmon "Building the Data Warehouse" [9]. In the first chapters, the Data Warehouse is presented as an information system, which has to provide analysis possibilities on the top of companies' data. The result from this analysis has to be primarily used to support decision-making. The idea of Data Warehouse was the first attempt to allocate a separate system used only for analytical tasks. Transactional systems[1] (such as CRM[2], banking systems, billing systems, etc.) which provide elementary data for analysis do not act anymore as analysis querying systems, but transfer data themselves into one common system - Data Warehouse. The foundation of Data Warehouse has brought with it a new distinction of previously not distinguished systems.

OLTP (Online Transaction Processing) is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). The focus of OLTP systems is put on transactions and maintaining of data integrity. Data stored in an OLTP system is mainly intended to support daily operations. An example of an OLTP system is CRM or billing system.

[1] A transaction processing system is a type of information system, which collect, store, modify, and retrieve the transactions of an organization (www.wikipedia.org)

[2] Acronym for system to support Customer Relationship Management

OLAP (Online Analytical Processing) is characterized by allowing users to analyze database information from multiple database systems. Due to its structure, OLAP provides multidimensional data, meaning the information can be presented in many different ways. The purpose of OLAP is analytical activity. An example of OLAP is a Data Warehouse [10].

2.2 Definition and Role of Data Warehouse

The most representative and widely used definition of Data Warehouse comes from Bill Inmon: "It is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions" [9]. From a detailed perspective, a Data Warehouse is a collection of data integrated together from individual transactional systems. In its way, the Data Warehouse is able to provide a single view of data from different systems. These systems usually store company operations and business data. This single view allows more sophisticated analytical approaches such as evaluation of correlations between data e.g. in order to identify customer needs and in response to offer the most suitable product or service to the customer. Data Integration is the core for Data Warehouse. It represents the logic regarding how data from individual transactional systems are merged together. The complexity of this logic can grow into enormous complexity and is highly dependent from the number of source systems and integration rules. This complexity has direct impact on the quality of integrated data. As the value of integrated data has been very soon recognized by many companies, the importance of data quality was considered in order to allow more precise analysis. Companies realized that many data inaccuracies occur as a result of the increasing complexity. This is an indirect result of still higher complexity by intern organization processes. Awareness of this problem brought a new topic to the scene – the area of data quality. Data quality consists of a set of processes and techniques which should increase both technical and logical qualities of integrated data. The data quality plays nowadays an important role in building of confidence to data. It is expected that the area of data quality will play an important role in the future [5].

2.3 Architecture of Data Warehouse

2.3.1 Logical Architecture

A Data Warehouse is neither a tool nor a product; it is rather a concept. Within this concept, it is necessary to support diverse processes for data reading and extracting from source systems, data transforming and integrating from multiple sources and finally providing access to data for reporting and analysis purposes.

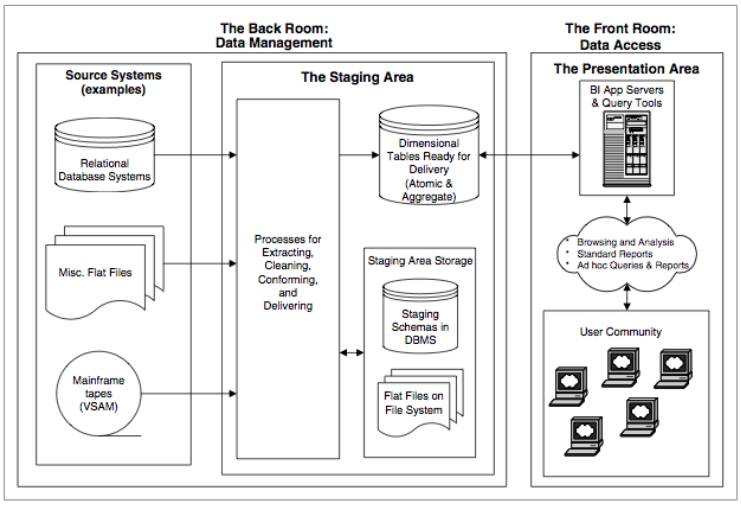

Figure 2.1: Architecture of Data Warehouse components, including relevant processes

A simple logical representation of the architecture supports firstly extracting of source data into an area called “Staging area”. A Staging area is a storage place for data in the form and structure as it was extracted from source systems. Once the data is extracted, transformation processes start transformation, integration and cleaning. These operations are precursors to loading data into “Core”. A Core is a data presentation area modeled by reporting tools for simple data accessing. As the business needs for the structure of data for different perspectives of individual subjects can vary, there might be a need for individual subject representation. Such needs can result in an extension in the form of Data Marts, with data organization according to different business requirements such as invoices, customers, billing, etc. The technical architecture of Data Warehouse consists of one or more databases (depending on the physical location of Stage, Core and Data Marts) and a set of components and tools, which are necessary for data querying, data loading, data transformation and data presentation. This functionality is nowadays mostly delivered by commercial ETL tools (e.g. Informatica, Data Stage, Oracle Warehouse Builder, Oracle Data Integrator) or simplified by SQL and PL/SQL routines. An inevitable part for successful implementation and maintenance of Data Warehouse is the provision of the developing, monitoring and maintaining infrastructure, which is as well mostly delivered by ETL tools.

Simplified view of this architecture is in the following Figure 2.1.

As the data warehouse architecture is very complex, in the following sections only the terminology relevant for this thesis, i.e. Data model, ETL and Data quality is shortly described.

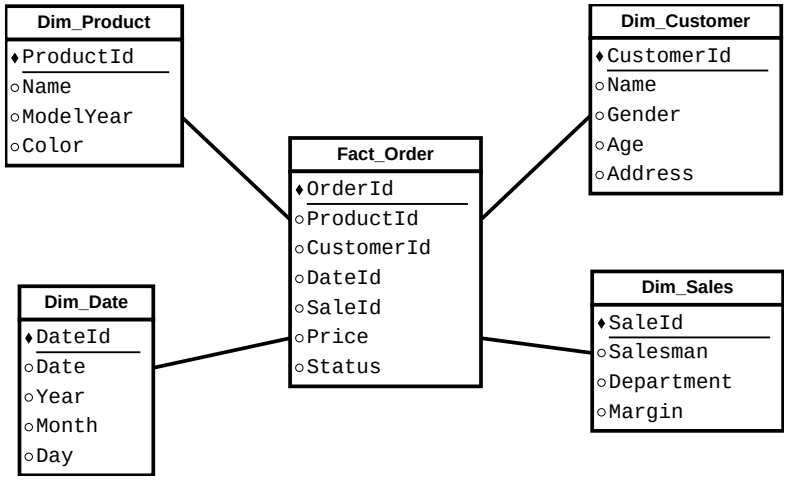

Figure 2.2: Example of star schema model with one fact table and 4 dimensional tables

2.3.2 Data Model

A data model represents a logical structure of tables within a database schema, including rules for data distribution within this structure. As a data model in Data Warehouse is nowadays used widely accepted dimensional model. The idea of adopting other than relational data model in data warehousing has been presented in the first book by Ralph Kimball "The Data Warehouse Toolkit" [11]. Kimball proposed a new data model - called dimensional model, which is currently a fundamental model for data warehousing. This model differs from the 3rd normal form of relational model. The structure of the dimensional model is characterized by its simplicity to an end-user, as the subjects are mapped very similarly to the real world. As data warehouse might contain up to several hundreds of tables, the simplicity of query building for an end-user is preferred to the efficiency of data querying in the traditional relational model. Tables in the dimensional model tend to have relations in form of a star or snowflake schema.

The simplified star schema consists of one central fact table which is connected with multiple dimensions within the relationship 1:N. The fact table stores measures and calculated facts or events. The dimensions store descriptive information about these facts. Direct relationships between dimensions connected to the central fact table are forbidden, as it might lead to loops.

The snowflake schema is very similar to the star schema. It is in fact an extension of the star schema in such a way that the dimension tables are decomposed into more subdimensions. This decomposition illustrates different hierarchies in dimensions data. In practice, data warehouse implementation often includes tables organized in star schema and also tables organized in snowflake schema [11]. Understanding of this structure is necessary for the subsequent chapters of this thesis, as all queries and performance

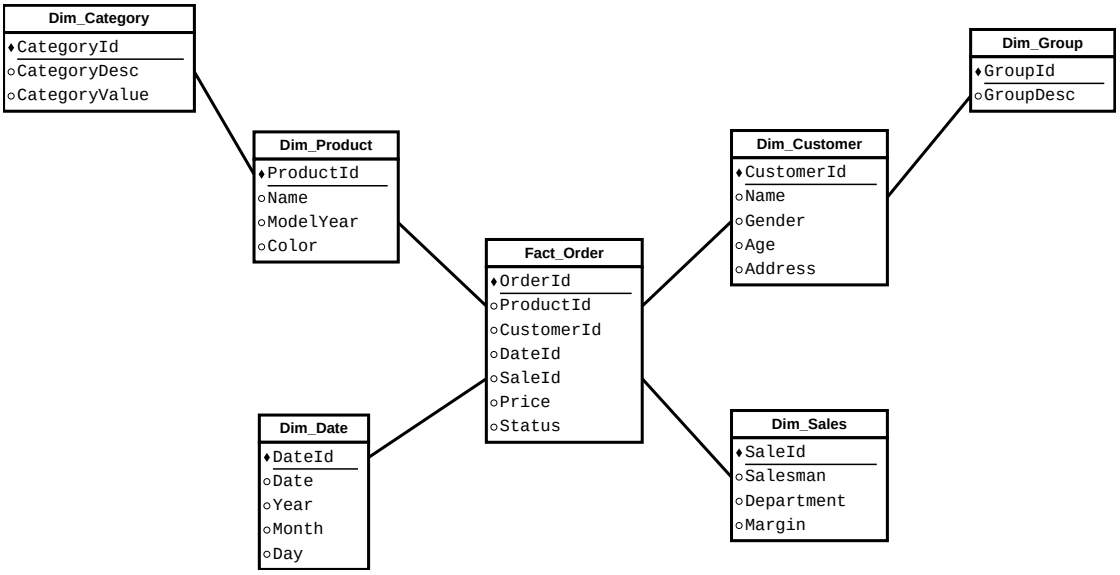

Figure 2.3: Example of snowflake schema model, this example is an extension of 2.2. New tables represents higher hierarchies of related dimensions

evaluations are applied on this schema. As these schemas (star; snowflake) are very similar to each other, only the snowflake schema is treated in the subsequent chapters of this thesis, because the star schema is actually included in the snowflake schema.

2.3.3 ETL

"Extract, Transform, and Load (ETL) processes physically integrate data from multiple, heterogeneous sources into a central repository referred to as data warehouse." [12]

2.3.3.1 ETL Transformations

ETL is the most important part of a Data Warehouse; with some overstatement, we can say it represents its DNA. It designs transformation rules and creates data flows for source data and at the end loads it into the target. This process is performed step by step in the order pre-defined by a set of transformation activities. These transformation activities are similar to operations executed on standard databases through SQL1 statements. Typical transformation activities are atomic operations, such as value assignment, filter, join, sorter, or even more complex operations, such as schema transformation, data cleansing or data validation [13].

The logic of the transformation process is mostly divided into smaller entities called sessions or jobs and in the next step organized into a logical workflow. This structure allows a better understandability and manageability for end-users and also space for de-

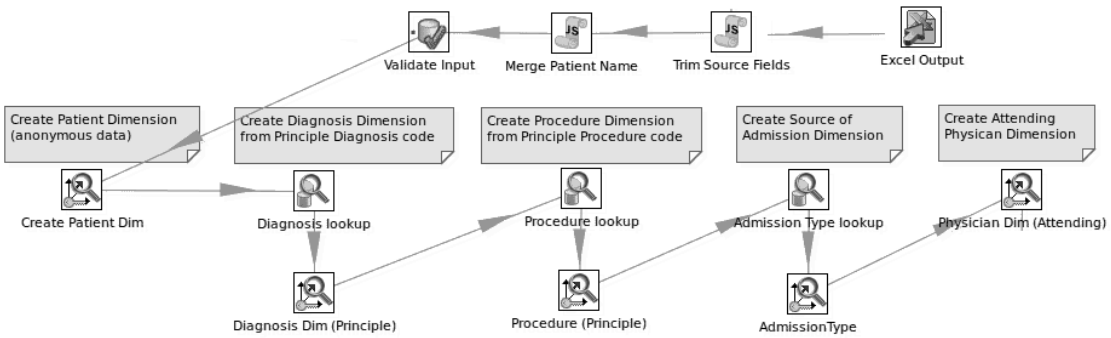

Figure 2.4: Example of ETL workflow in the open source ETL tool Pentaho

signing of dependencies between the individual entities. An example of ETL workflow is illustrated in the next figure.

2.3.3.2 ETL Processing

Transformations can be executed on the database using 2 different strategies:

1. ETL performs separately each activity directly on the database and then pushes the result-set as the source for the next activity. A disadvantage of this approach is performing of separate database querying for each activity and it leads to a decreased performance of the process.

2. ETL loads at first data from all sources defined in a workflow and then processes this data "on the fly" in memory. A disadvantage of this approach is in the fact, that if the sources contain a big amount of data, this entire amount should be loaded into memory, even if it includes data not impacted by processing.

The result of both approaches is equivalent. The preference of the first or second approach is always dependent on the objective and subjective judgment of Data Warehouse architects. Besides the performance aspects, also the implemented database system with its own characteristics plays an important role in making this choice.

2.3.3.3 ETL Components

The data integration is the most dominant role of ETL, but in practice, ETL functionality covers a much bigger area. For the proper logical transformation of data, it is often necessary to create additional processes e.g. for data historization, data updating or data cleansing.

During the past years, focusing on the data cleansing became an unavoidable part of Data Warehouses [14]. With the mission of data quality monitoring and correcting, it has to be a part of an ETL solution. It can be implemented directly into the integration process of ETL or as recommended be decomposed into a stand-alone data quality process. The reason for de-coupling of the data quality part from the integration part of ETL is based on the primary function of each process. The integration process physically integrates and changes data (merge, join, insert update), while the data quality process only checks and evaluates data but does not change it.[3]

[3]The cleaning process could also change data (e.g. data enrichment), but it is not recommended. It could bring inconsistence into data warehouse.

CHAPTER 3

Data Quality

The primary goal of data quality examination in data warehousing is to determine whether the data is suitable for the intended purpose and satisfies defined requirements for end user needs. Manually entered data, inappropriately defined transformation rules in ETL process or unsatisfactorily defined data requirements are often sources of data problems called "data defects" [5]. In order to identify these data defects, data quality examination and evaluation are necessary to distinquish between high quality and low quality data. A decrease in data quality leads to a particular uncertainty while working with the data and causes decreasing confidence in the data. As a result, there might be a necessity to execute additional post-processing data validation checks by end-users (e.g. manual data check). In order to build credible data, the data quality process has to identify data defects and, if possible, undertake corrective actions. The importance of data quality has been recognized in many research papers, including data quality impact on information quality and decision-making process. [2, 15, 16] The most appropriate definition of data quality can be found in [5] "Information and data quality is commensurate to its ability to satisfy its customers and to meet customers' needs".

3.1 Data Quality Process

3.1.1 Scope of Data Quality

Before we discuss detailed characteristics and processes within data quality, it is necessary to understand the meaning of quality within data quality. There are several definitions of quality which might be used also in relation to data quality. Among these are many relevant to the general perception of quality [17], including its subjective and objective characteristics, but in the sense of data warehousing, this might provide only

a one-sided view. In general, data quality in data warehousing is a combination of product-based (source data) and process-based (process of data manipulation) quality.

With respect to this definition, there are 2 dominant factors which influence the data quality:

1. Source data – product-based quality. The quality of data in source systems has significant impact on the overall data quality. If the source data has data defects, these are further spread to all related modules (Data warehouse, Data Mart) and presented to end-users. Source data quality is highly dependent on the quality of manually entered records. Names, addresses or facts are often misspelled, not entered or entered with wrong dummy values. Next, the processing of records (editing, deleting, copying) can lead to inconsistence in business contents (orders in invalid statuses, incorrectly deleted customers, etc.). The so called "orphans" created in this manner are usually not identified by source system application and instead are still stored in databases as normal records.

2. Integration and transformation process – process-based quality. The integration process modifies data according to pre-defined rules. As the transformation process works with source data, it expects a certain structure and content in source tables (good data quality). If this content does not match the expected structure, it may result in unexpected data modifications. Similarly, problems can be caused by incorrectly defined transformation rules, which are often defined without sufficient knowledge of business logic. To mention a simple example, orders are not categorized according to their statuses (open, checked, confirmed, processed, canceled, re-created, etc.) but are processed together and in the process of revenues evaluation canceled orders are incorrectly included among all orders.

3.1.2 Types of Data Defects

Many research papers define a set of data quality dimensions [5, 18, 19]. This set is not fixed and there are several discussions regarding which dimension to include. The most important and commonly agreed dimensions are: accuracy – do we have up to date data ? , completeness - do we have all data ?, conformity - do we have data in correct format ?, consistency - do we have any data conflicts ? A combination of these dimensions serves to determine a set of possible categories of data defects. Thus, each data defect has a negative impact on at least one of these categories, but can also impact more than one.

English (1999) recognizes the following error patterns:

- Domain value redundancy – Non-standardized data values or synonym values, where two or more values or codes mean the same.

Example: A customer is assigned to 2 different groups with different hierarchy levels.

- Missing data values – Data that is supposed to have a value is missing. This includes both required fields and fields not required to be entered upon data capture, but necessary for downstream processing. Example: A customer has not filled in a product price in an order.

- Incorrect data values – These may be caused by transposition of keys, entering data in a wrong place or misunderstanding the meaning of the data captured.

  Example: A customer's first name has been entered into the column for a customer's family name.

- Duplicate occurrences – Multiple records that represent one single real-world entity.

  Example: A customer is recorded in 2 different systems and is not recognized as the same person by the integration process.

- Inconsistent data values – Data stored in relevant databases are updated inconsistently.

  Example: A customer's email address is not changed in general but only for a current order.

- Information quality contamination – This error results from combining accurate data with inaccurate data or when several accurate data sources are combined by using an inaccurate integration process.

  Example: Prices from a current order are assigned to a previous order.

The nature of these categories allows us to predict that some data quality aspects might be identified and managed to a higher degree than the other. E.g., it is quite difficult to increase information quality contamination of data as there are only limited possibilities of validating entered data against "real-world" data or enriching the data with this new information.

3.1.3 Identification and Processing of Data Defects

There are several techniques for identification of data defects, ranging from simple rules for checking allowed values to sophisticated rules for checking the business meaning of delivered data (e.g. by using of different algorithms and knowledge-based activities). [1] A single execution of certain data rules is usually unsatisfactory to guarantee the overall data quality. For this reason a data quality process is introduced and is responsible for

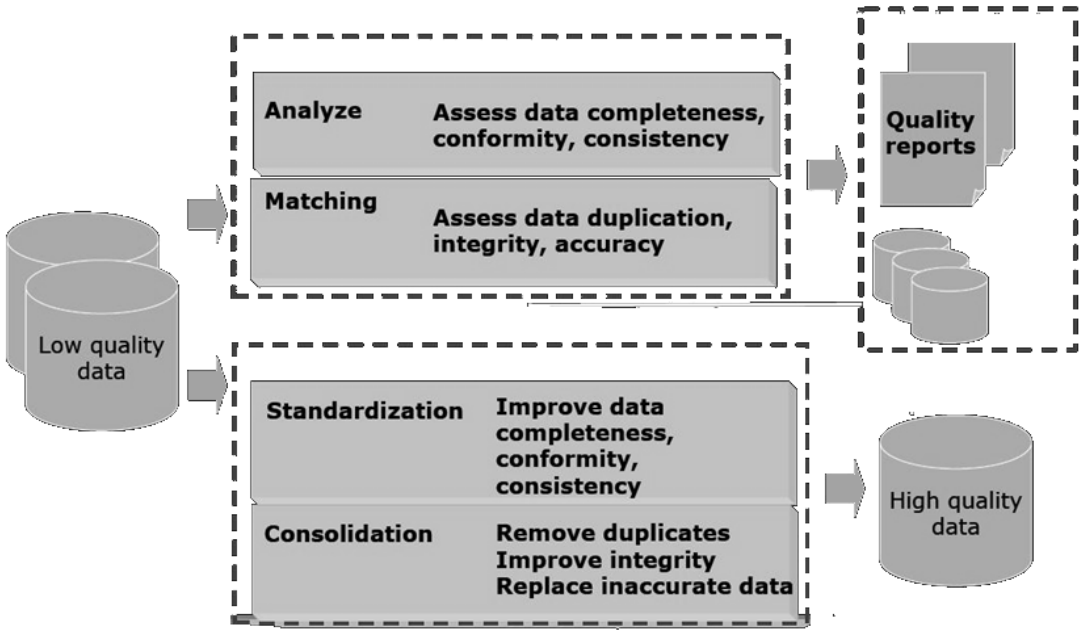

Figure 3.1: Illustration of a complete framework [20] used for the data quality process.

the general elimination of data defects. Several methodologies and frameworks for creating and covering data quality processes have been established in order to ensure the overall data quality. A comprehensive description and comparison of those methodologies is contained in [1]. A simplified form of these frameworks, which is currently very often used in practice, is illustrated in the Figure 3.1. For the purposes of this thesis, we treat in detail only the relevant part Data analysis. This part consists of 4 steps:

1. Definition of data errors types
2. Data analysis in order to identify data errors
3. Correction of errors and data enhancement
4. Documentation of changes

3.1.4 Data Analysis

Data analysis is a process that helps to detect structural and content-related characteristics of data. When narrowing the meaning of data analysis only to data warehouse, it

represents primarily data checks against defined rules. There exists also a more in-depth data analysis, including statistical analysis and detailed data pattern recognition, but it might be present only in additional parts of data warehouse called Data mining.

Data Analysis could be generally performed using 2 different approaches:

- Data-oriented approach – Data inspection and data cleaning
- Process-based approach – Analysis of transformation activities or detection of critical paths

Of these two approaches, data oriented analysis is the preferred choice in case of Data Warehouses. It examines in particular data contents and indicates possible data errors. In comparison with the process-based approach, it provides a more detailed data inspection and data validation. An example of data clensing framework could be found in [21].

3.1.4.1 Data Profiling

The planning of data analysis begins with the understanding of current data contents and data structure. This is the fundamental basis for defining what data characteristics data needs to comply with. These characteristics have to be defined at the beginning, even before the implementation of the data transformation process. Unfortunately, in most cases the value of good data is recognized only after first "wrong data" appears. Even though many researchers showed that it is almost impossible to predict all data quality problems at the beginning, the quality evaluation of source data, which is done before the ETL process is designed, can help to identify first quality defects. The preferred evaluation method of source data quality is called Data profiling. It examines source tables and classifies table records in order to determine frequency distribution of data patterns and data relationships. These findings help later when designing of first data checks. A presumption for data profiling is an expected similar distribution of data patterns during daily loads; so the process covers the data life cycle. The result of this process serves as an input for further analysis and data quality rules preparation. This concept of data quality analysis and monitoring is derived from the TDQM1 methodology, well-known in the field of data quality. TDQM assumes the following parts of quality process: define, measure, analyze, improve. Details of this methodology, including a comprehensive comparison with other suitable data quality methodologies are listed in [22]. A fundamental part of data analysis is built based on data quality rules. Characteristics collected in the process of the first real data examination are subsequently turned into data quality rules.

6 Total Data Quality Management is a research program, which with the aim of data quality definition, analysis, and improvement.

3.1.4.2 Data Quality Rules and Data Checks

Criteria for data quality identified by end-users and data analysts, including patterns and characteristics from data profiling, are decomposed into data quality rules (logical definitions) and transformed into data quality checks (physical implementation of logical definitions) [23]. Data quality rules can be of different nature and there are several classifications and categorizations of the rules [5, 24]. For the purpose of this thesis, we use the classification proposed by [25]. This classification divides these rules very similarly by suggested area for its own usage. This consideration is of great importance from the practical point of view as it represents frequently occurring distribution of data rules within a data warehouse.

- Database integrity rules: Rules based on database constraints and column data types.

  Example: All order details are supposed to have corresponding records in the main order table.

- Match and merge rules: Rules that match records from different data sources and merge them into a single record. These are sometimes called also deduplication rules or record linkage problems.

  Example: Identical customers integrated from two different systems are supposed to have only one instance in the unified data warehouse table.

- Business rules: Rules defined by business users that look for business related suspicious values based on comparison and logical relationships. These include missing values or erroneous data based on a certain business definition.

  Example: If there are more bank accounts assigned to a customer, there must be exactly one of them, which has the priority indicator set to 1.

The logical definition based on data rules is transformed into its physical implementations called data checks. Many types of data quality checks can be implemented in a similar way and by using almost the same components as those used in the ETL process, as the core is created using SQL operations. These include table scan operations, conditions for accessing only selected records and also conditions for validating selected rows against defined rules. Data quality checks are designed to provide only 2 evaluation results: passed or failed. The main difference between ETL logic and data quality checks is the expected result. Unlike in case of ETL process, the result of data quality process is checking whether the data satisfies defined rules, but without a physical change to it. The process of changing records may sometimes be a part of a data quality solution, but to ensure consistence with source systems the process is mostly omitted.[1]

[1] In practice, these are usually bad records, which are intended to be replaced with new "enhanced" information flagged with invalid status "not valid", but the records are not deleted or changed. Conse-

Taking into account this similarity, it is mostly possible to implement data quality checks within the ETL tool by using ETL tool components. In the market, there are several specialized products with the primary focus on data quality (DataFlux, Trillium Software, Informatica) [26], which cover almost the whole data quality process, including data profiling, data rules definition and a sophisticated algorithm for pattern matching.

3.2 Performance Aspects of Data Quality

The performance of data checks plays an important role in a Data Warehouse. Data checks are SQL operations with all their performance characteristics and they are executed against a real database. Bearing in mind the improving data quality, it is often necessary to run tens or even hundreds of checks in a reasonable timeslot. As data checks have to be a part of an ETL solution, the processing time has to be minimum. At present, this is the most limiting factor of integrating parts of the data quality process into data warehouses.

3.2.1 Data Model

There are several data model factors which play a role in the performance of designed rules. The architecture of the data model needs to be considered already during the initial steps when evaluating the performance of data quality checks. The data model represents definitions of table orchestration (structure, composition) and defines the character of data quality checks. E.g.: In the dimensional model, which is created for understandability, there is a high probability that many tables are joined together in order to create a meaningful business context, which can be later validated. As a simplified illustration, we can say that the meaningful concept consists of one or two joined fact tables and more dimension tables.

The kind of data model predicts in a particular way the probability of presence of different kinds of data checks and data errors. Accuracy, conformity or completeness checks can be executed already on the source data (mostly entity relational model), while integrity or business checks even after integration stage (snowflake model).

3.2.2 Atomic SQL Operations

The performance of atomic SQL operations has a big importance to data warehouse architects. We can compare it to the validation relevancy of data check, being the most relevant criterion for data quality end-users. It decides which logical concept of SQL

quently, new records might be created or copies of the old records updated with the new information, but these records have to be similarly marked as they are impacted by the data quality process.

operations are used in data checks. The whole performance of data check is strongly dependent on the performance of individual SQL atomic operations of which it consists. These atomic operations are atomic processes on the database site, such as: table scan, join, sort operation, merge operation and etc. Fortunately, SQL offers a wide range of operators, which allows different strategies to build identical logic with different performance demands. There are several factors which influence the performance of individual SQL operators to a different degree. Some of them are manageable on technical level (indexes, partitions, database performance/load), some of them based on fixed defined characteristics (number of scanned records, records data type) and some of them based on internal database operations executed in relation to supporting the logic of executed SQL. These 3 categories can be of different importance in the course of the real execution and every category can be dominant in specific cases. Regardless of this fact, the last category is of the highest importance to us. In comparison with the first 2 categories, which are mostly fixed predefined, we can influence the occurrence of SQL operators by building of different logical design. Thus, we cannot influence the performance of atomic operators directly, but we can optimize the composition of data checks by using low demanding SQL operations. The calculation of generic performance demand for individual SQL operators is complicated, because it is dependent on the previous two criteria. It is possible that an operation considered in general as low demanding is in specific cases significantly slower than an operation considered as high demanding. With respect to these limitations of defining exact performance cost of SQL operations, we decided to define our own notation and definition of SQL performance. The rationale behind this definition is to define two categories of operations (table scan, table join). The performance measurment of the data check will base on the category and amount of operations which are necessary for its execution. This new definition helps us to compare different compositions of SQL checks.

Definition 3.1. Given are 2 independet not related tables. In the standard environment the scan operation of 2 given tables performs by far better than the table join operation executed on those 2 given tables. The rationale behind this statement is the fact that joining of two tables requires in addition to a table scan for both tables also a sort and match operation for both tables, which requires additional performance demand. △

Definition 3.2. In analogy to the 3.1, we assume that one table scan is by far better performing than table join of this table with another one. △

According to our 3.1 and 4.1, the following frequently used data check SQL operators obtain the following performance complexity:

1. Select from one table – complexity of 1 table scan

2. Join of two tables – complexity of 1 join scan, in case of joining of more tables, the complexity is calculated as a count of joined tables - 1

3. Union – complexity of 1 join scan (this is indicated by the need of sort process during union operation), in case of union of more tables, the complexity is calculated as a count of unioned tables – 1

4. Union All - complexity of 1 table scan (no sort operation needed), in case of union of more tables, the complexity is calculated as a count of unioned tables – 1

5. Subquery - complexity of 1 join scan (this is indicated by the same processing as in case of join operation), in case of more subqueries, the complexity is calculated as a count of individual subqueries

6. Minus – complexity of 1 join scan (this is indicated by the need of sort process during minus operation), in case of minus of more tables, the complexity is calculated as a count of minused tables – 1

7. Merge - complexity of 1 join scan (this is indicated by the need of sort process during merge operation), in case of merging of more tables, the complexity is calculated as a count of merged tables – 1

8. Where condition – does not have any complexity, it is always used within other SQL operations which have the complexity of table scan or join scan

3.2.3 Performance of Individual Types of Data Rules

The type of data rules partially predetermines in a certain way the structure and degree of complexity of SQL statements. This leads to the state, that some sort of rules has generally lower performance demands than others. The next section describes the consideration of data rule types in relation to performance demands, which provides only one-sided view in cases when data rules are from different categories executed separately. In practice, it is very common that rules from different categories are merged and executed together.

3.2.3.1 Database Integrity Rules

Raw data from the source systems are checked using database integrity rules usually "stand alone" (separately table by table), as it is assumed that relationships between tables are maintained by source system applications and they do not need to be validated. These checks can represent potential performance problems in case that there are many similar scan checks which access the same table. In this case, a separate table scan has to be executed for each check. This proceeding leads to redundancy, because each record is accessed by every check, so the total execution time is calculated as the sum of cost for one table scan multiplied by the number of data quality checks. A convenient and not necessarily complicated solution to this kind of data checks is to create an SQL

statement, which scans all affected records in the table and includes ‘case‘ conditions for evaluation of individual checks. This kind of data check covers all check evaluations and acceses the scanned table only once. One of the befenits by this approach is the fact, that the structure of the data model does not negatively imacpt the performance.Every table is scanned separately and the relations between the tables does not play any role. This kind of data check mostly does not feature performance problems and in case it does, then there is only a minimum room for improvements in the logical level of data quality rules constellation. Database integrity rules have usually the complexity of 1 table scan.

3.2.3.2 Match and Merge Rules

This kind of check is a typical representative of checks executed after the integration part on the dimensional data model. A typical data integration scenario includes 2 or more tables which are merged into a single table. A fundamental element of data integration is a unique key (called primary key/business key) which creates an identifier for record identification and aims to merge together data from multiple sources. Designed checks to confirm correctness of the match and merge process use this key as a join condition to validate that all instances with the same key have been correctly merged into one or more records. The match part of this rules has to ensure that the whole set of records from the source has been reflected in the target side. From the performance perspective, the occurrence of joins represents a decrease in performance, but joins are unavoidable in this context. If the validation of merge process includes only minimum joins necessary to verify business contents (i.e. without redundant table scans), there will be only a small room for performance optimization.

Typically, the match and merge process involve only small amount of tables (e.g. 2 or 3). The reason is, that the comapnies tend not to use similar entities distributed in more than 2 - 3 source systems. The complexity in this cases is usually not bigger than 1 or 2 join scans. Match and merge rules can be considered from different perspectives as a special case of business rules. Split of these rules into separate categories is often argumented with the necessity to compare source data (relational model) with target data (dimensional model).

3.2.3.3 Business Rules

Business rules are another example of checks executed on dimensional data after the integration process. Compared to the match and merge rules these rules have a bigger variability. The purpose of these rules is to confirm that data modifications designed for integration processes have been applied and target data is in conformity with the intended purposes. Data quality checks designed for business rules do not have to follow integration process logic and do not concentrate on checking data against the source, but

primarily confirm the validity of business rules for target data. There are several areas suitable for their application: validation that all records have been transferred, validation that relationships from the source have been preserved in target or validation that new relationships have been established/created in target. Business rules have the greatest importance for end-users, because if they are well-designed they approve the entire data transformation process. As the integration process can reach enormous complexity, there is a need for the validation of many business rules. Business rules in their own way control the proper distribution of relational data in a newly designed dimensional data. This sort of checks usually requires table joins to validate the existence, uniqueness or relationship between records coming from more tables. It is not unusual 5 – 6 business rules with similar structure, but validating of different business contents, to be designed in order to validate 2 or 3 target tables. This creates a large room for performance improvement, as it is not reasonable to execute queries which access the same records many times and retrieve the records again and again with slightly different conditions. Business rules have usually the complexity of many join scans.

3.2.4 Composition of Data Checks into Complex Data Checks

The complexity of data checks might range from very simple checks to very complex checks. Even if a check is quite complex, it consists only of basic atomic operations, which are either performed in a defined order or, if possible, merged together. The composition of such checks is performed either with the help of an ETL tool or manually. The kind of composition can significantly influence the performance of the check. Composition made using automated professional Data Quality tools is usually not satisfactory. Data quality tools mostly do not include performance aspects and focus more on providing a visually attractive design environment and extensive controls over administration process.

Example 3.1. The following is a typical example of processing simple data checks with using a typical data quality tool.

We want to validate values of 1 attribute - Title in the table *table_customer* depending on gender. If gender is ‘W’, allowed values are (‘Mrs’,’Ms’), if gender is ‘M’, allowed values are (‘Mr’). The result is supposed to deliver a count of records which do not comply with these rules. This data rule has been implemented through 2 data checks: ▲

1. Rule validating allowed values for gender = ‘W‘

```
Select          count( case
                when title not in ('Mrs', 'Ms') then 'FAILED'
                else NULL end) as qc_title_woman
from            table_customer
where           gender = 'W'
```

2. Rule validating allowed values for gender = 'M'

```
Select          count( case
                when title not in ('Mr') then 'FAILED'
                else NULL end) as qc_title_man
from            table_customer
where           gender = 'M'
```

There are 2 possible processing ways used by data quality tools for these data checks (these ways are very similar to the 2 different approaches by processing of records in ETL tools, see section 2.3.3.2).

1. If the processing is made sequentially, then both data checks are executed sequentially. This involves time complexity of 2 table scans.

2. If the processing routine loads at the beginning all data into memory and then examines data, the complexity is one table scan, but there is still a great redundancy, because all data from table are loaded there, not only those which satisfy the conditions for gender = 'W' or gender = 'M' Database tools do not try to merge more rules together to calculate and minimise the execution time. The reason for this behavior is in most cases the general problem of finding minimum necessary SQL statement (mapping) in case of more data checks, which covers exactly the size of data set necessary for examination.

In the case above, one of possible minimum solutions can be composed using the following SQL statement:

```
Select          count( case
                when title not in ('Mrs', 'Ms') then 'FAILED'
                else NULL end) as qc_title_woman,
                count( case
                when title not in ('Mr') then 'FAILED'
                else NULL end) as qc_title_man
from            table_customer
where           gender = 'W' or gender = 'M'
```

This SQL statement has the complexity of 1 table scan, which is the best possible performance demand. This example was very trivial and the design of the final solution was prepared smoothly without any complex logical examination needed. Unfortunately, the integration process usually requires validating of complex relationships with many dependencies. In that case, it is hardly possible to find a universal method how to merge related data checks together to omit redundant table scans and joins.

In the next section, a typical integration schema from source to target, including existing dependencies is proposed. This schema is used for further examination of data exchange algorithms in order to discuss an automated way how to merge as many related data checks as possible by using data exchanges and schema mapping algorithms/principles.

CHAPTER 4

Data Exchange

The aim of this chapter is to introduce the formal definitions of data exchange. Subsequently, we introduce the universal solutions and algorithms for their computation, core and query answering as the central parts of data exchange. In the last section of this chapter we examine the data quality and data exchange from the aspect of their possible correlations.
Definitions used in this chapter are accompanied with pictures and typical examples to provide better understandability.

4.1 Introduction to Data Exchange

Data exchange deals with translation of structured data which is stored in the source schema, into the target schema by specifying its new target structure. The theoretical foundations of data exchange has been for the first time formalized by Fagin et al. [27] in 2003. It was quite recent formalization under the considering, that the problem area of data exchange arose along with first database principles many years ago e.g. [28]. After its first formalization by Fagin et al., the research in data exchange area has become increasingly important in the course of following years. Nowadays, the research continues mainly with focus on theoretical aspects with consideration of different data exchange settings and scenarios. Nevertheless, the foundations laid by Fagin represent its pillars till today. The prevailing definitions, used in this chapter, refer to these principles.

Before we provide explicit definitions of terms and principles in data exchange, let us shortly explain the topic in a more understandable form.

A simplified picture of data exchange is represented by a given source schema S, given target schema T and by translation instructions how to map data from source schema into target schema $\sum_{st}$. These translation instructions have the form of depen-

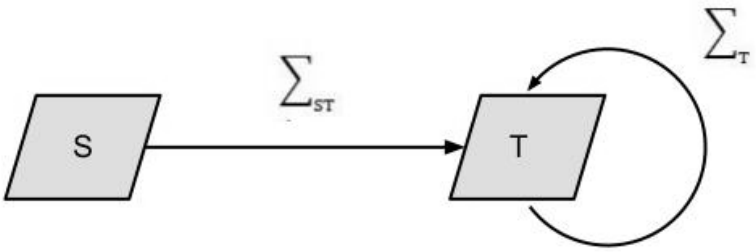

Figure 4.1: Graphical illustration of data exchange setting.

dencies. In addition to these dependencies, the target schema has its own dependencies $\sum_t$, which the data must meet.

The most important part of data exchange is the data exchange problem. Data exchange problem is denoted by materialized source instance and a problem how to compute target instance by using data from source instance under the source-to-target dependencies (translating instructions) and target dependencies. The main task in data exchange focuses on finding an universal algorithm how to efficiently compute the target instance for given data exchange problem.

4.1.1 Data Exchange and Data Integration

Data exchange area has many similarities with data integration used in data warehouses [1]; the most important is the process of transforming data from the source schema to the target schema. This process is very similar but not identical. The dominant aspects of this process are determined by different roles of these both subjects.

Data integration in the meaning of data warehouse integration is understood as integration of data from multiple source schemas into one target schema. The purpose of data integration is to provide one query answering interface, in this meaning - the target. Consequently, the structure of the target is afterwards designed to conform to the structure of source schemas and dimensional model. The target structure is actually the product of a step based process by designing the data integration solution (e.g. ETL process). It is created "on the fly" during the process dependently on the source schema structure(s). The result of data integration process is the materialized target instance loaded through the ETL process.

In contrast to the data integration, in the data exchange there exists only one source schema. Furthermore, the data from this schema should be mapped into the target,

[1] The data integration process used in data warehouses differs from the classical theoretical data integration

which has already defined structure.
The mappings and the target are not the products of data process anymore but are defined in advance. The result is the data set consisting of source data, which meet the mapping rules and target structure.

The knowledge of differences between the data integration and data exchange is important, as it plays an important role by its application in different practical tasks. Detailed information to the subject of theoretical data integration could be found in [29, 30]. On the other hand, the data integration in the meaning of data warehouse process is extensively described in [31].

4.1.2 Theoretical Aspects versus Practical Aspects of Data Exchange

The process how to find a corresponding data for the target schema in data exchange is interesting primarily from its logical perspective. At the beginning, in the research community dominated considerations about algorithms, which calculate the target instance and its computational complexity. Over the time, a wide range of aspects in the problem area opened new interesting topics for the researchers, such as non-trivial data exchange settings, certain answers and working with arithmetic operators, meta-data and lineage. These new research approaches align with the practical use in everyday life, primarily in database community.

When comparing the practical application of classical data integration and data exchange in data warehousing, the data integration clearly dominates over the data exchange. One of the reasons is the general nature of data warehouse, which is the integration of data from more sources into one target. The target schema is pre-defined and is the result of design process based on the structure of source schemas. From this point of view is data exchange intended for more specific tasks, where the target schema is already designed. The typical example could be the merge of data from one operational application into another operational application.

4.2 Fundamentals of Data Exchange

As mentioned above the first considerations about the data exchange problem were presented by Fagin et al. [6, 27]. The authors introduced a basic framework to describe the semantic of the data exchange problem, including its foundational and algorithmic issues.

4.2.1 Data Exchange Setting

The fundamental part of data exchange is denoted by the schema mapping, which represents the data translation instructions.

Definition 4.1. A *schema mapping* is denoted by a triple **S, T,** $\sum$.
S represents a source schema; **T** represents a target schema and $\sum$ represents the sum of all dependencies between the source schema and the target schema.

△

Based on the schema mapping we can introduce a data exchange setting.

Definition 4.2. A *data exchange setting* is a schema mapping determined by **S, T,** $\sum_{st}$**,** $\sum_t$
S represents a source schema; **T** represents a target schema, $\sum_{st}$ is a set of dependencies defined between the source and target, $\sum_t$ is a set of dependencies defined for the target schema.

△

The set of source-to-target dependencies ($\sum_{st}$) is an essential part in data exchange setting, as it actually defines the mapping rules between the source and target.

Dependencies used in data exchange could have in general 2 forms:

- **Tuple**[2] **generating dependency** [3] – A *tuple-generating dependency* (TGD) over a database schema R is a formula of the form $\forall\vec{x}(\varphi(\vec{x}) \rightarrow \exists\vec{y}\mu(\vec{x},\vec{y}))$, where both the antecedent φ, and the conclusion μ are conjunctive queries over the relational symbols from R.

- **Equality generating dependency** [3] – An *equality-generating dependency* (EGD) over a database schema R is a formula of the form $\forall\vec{x}(\varphi(\vec{x}) \rightarrow x_i = x_j$, where the antecedent φ is a conjunctive query over the relational symbols from R and $x_i, x_j \in \vec{x_i}$.

Based on these definitions, we can demonstrate the first example of simplified data exchange setting. As we would like to bring the data exchange world closer to the data warehousing, we introduce a typical data warehousing example adapted to the data exchange world.

Example 4.1. *Remark: This example is the main example, based on which we demonstrate almost all of data exchange characteristics in the further parts of this thesis. Depending on the demonstration requirements, we can extend its initial setting if necessary.*

[2] A tuple in the database terminology means a record in a table

A company stores data about product sales in the company database. After a new sale transaction is finalized, new order record is automatically created in the table Order. Moreover, another new record(s) are simultaneously created in the table *Order_Items*, which holds details about the order, primarily about product specifications. This data is daily transferred through the ETL process to the existing data warehouse tables (*Fact_Order*, *Dim_Product*, *Dim_ProductGroup*)
In the first part of this example we define the initial data exchange settings including structure of source and target schemas, source-to-target dependencies, target dependencies and records stored in source schema.

The next schema represents the table structure in the source application.

Source schema **S**: *Order*(OrderId, Price, Customer, OrderDate)
Order_Item(OrderItemId, OrderId, ProductName, Quantity, Payment)

This data has to be loaded into another schema (in this case the data warehouse), which has typical dimensional structure, as illustrated below:

Target schema **T**: *Fact_Order*(Id, MainOrderId, Price, OrderDate, ProductStatus)
Dim_Product(Id, OrderId, ProductName, ProductGroupId, Sold_Date)
Dim_ProductGroup(Id, ProductGroupName)

Continuing in our example, we have mapping rules with the information how the data is supposed to be loaded into the target schema. These rules are expressed through the source-to-target dependencies.

Logical formalism of these source-to-target dependencies ($\sum_{st}$) has following form:

Dep.1: *Order*(OrderId, Price, Customer, OrderDate) $\wedge$ *Order_Item*(OrderItemId, OrderId, ProductName, Quantity, Payment) $\rightarrow$ ($\exists$ Id, ProductStatus, Id, OrderId, ProductGroupId, Sold_Date) *Fact_Order*(Id,Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name, ProductGroupId, Sold_Date)

As we already have the existing data in data warehouse (target side), we have to accept constraints[3] on the target site.

Target dependencies $\sum_t$:

Dep.2: *Fact_Ordee* (Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name ,ProductGroupId, Sold_Date) $\rightarrow$ *Fact_Order* (Id, Main_OrderId, Price, OrderDate, "COMPLETED")

[3]In the naming convention of data exchange, the terms constraints and dependency are considered equal

Dep.3: *Dim_Product* (Id, OrderId, Product_Name, ProductGroupId, Sold_Date) $\rightarrow$ ($\exists$ ProductGroup_Name) *Dim_ProductGroup* (Id, ProductGroup_Name)

Dep.4: *Fact_Order* (Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name, ProductGroupId, Sold_Date) $\rightarrow$ Sold_Date = OrderDate

The next Figure 4.2 shows all defined dependencies in a graphical form. We can see also the natural existing dependency determined by source system between attributes Order.OrderId and Order_Item.OrderId.

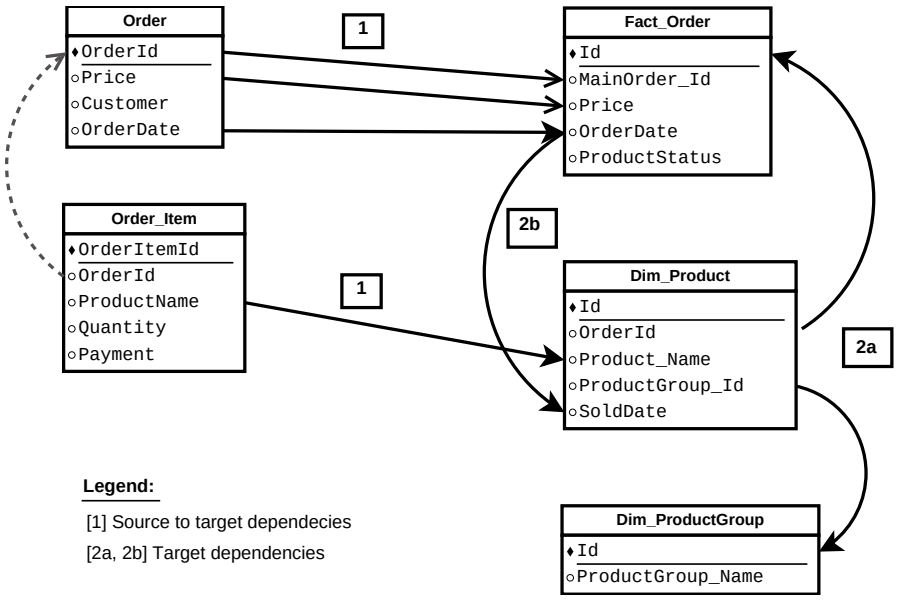

Figure 4.2: Graphical illustration of dependencies.

Defined dependencies correspond to the following business scenarios:

1. Order records including detailed information are supposed to be loaded into the corresponding *Fact_Order* and *Dim_Product* tables. In order to keep the existing Product reference (OrderId $\rightarrow$ Product_Name) from the source tables, the new OrderId foreign key is introduced in the target table *Dim_Product*. Identification of products between the source schema and target schema is specified by MainOrderId. This dependency is TGD.

2. Each record in *Fact_Order* which is referenced from *Dim_Product* should have the ProductStatus = ”COMPLETED” This dependency is TGD.

3. For each product record stored in the table *Dim_Product*, there should exist a corresponding product category record in the *Dim_ProductGroup* table. This dependency is TGD.

4. Each product record stored in the table *Dim_Product* has to have an equal SoldDate with the referenced Fact_Order.OrderDate. in which it was ordered. This dependency is EGD.

The above is a logical description of data exchange setting, which is created by a set of variables including the logical relationships between the schemas. In order to calculate the data exchange we need to have records stored in the source schema. An example of those records could be one record stored in the table *Order* and one corresponding record stored in the table *Order_Item*:

Rec.1: *Order*(OrderId → "1", Price → "100 Eur", Customer → "University", OrderDate → "1.1.2012")

Rec.2: *Order_Item*(OrderItemId → "1", OrderId → "1", Product_Name → "Walkman", Quantity → "2", Payment → "VISA")

▲

Now we have all parts necessary for data exchange and we can look on the actual data exchange problem.

4.2.2 Data Exchange Problem

After we introduced all parts of data exchange we need to answer a natural question: Which data from the target schema will correspond to this source schema data by satisfying the defined dependencies from the data exchange setting. As mentioned earlier this is the core question related to data exchange problem.

Definition 4.3. Data exchange problem: given an instance I over the source schema S, materialize an instance J over the target schema T such that the source-to-target dependencies $\sum_{st}$ are satisfied by I and J together; and the target dependencies $\sum_{t}$ are satisfied by J.

△

An example of a target instance which satisfies dependencies from Example 4.1 could have following representation:

Fact_Order(Id → "1", MainOrderId → "1", Price → "100 Eur", OrderDate → "1.1.2012" , ProductStatus → "COMPLETED")

Dim_Product(Id → “1”, OrderId → “1”, ProductName → “Walkman”, ProductGroupId → “1”, SoldDate → “1.1.2012”)
Dim_ProductGroup(Id → “1”, ProductGroup_Name → “Electronics”)

Based on this definition, each materialized target instance, which satisfies both – source-to-target and target dependencies is a solution for a given data exchange problem. This also means that the number of solutions is not limited and we can assume that there are infinitely many solutions for a given data exchange setting. E.g. in the Example 4.1 we do not have any constraints for the attribute ProductGroup_Name, so we can replace the attribute value by any other value and it is still a valid solution of this data exchange problem. This situation makes it necessary to define formalism for a solution, such that it does not contain any assumptions or data, which cannot be derived from the source. This solution is called universal solution [27].

4.2.3 Universal Solutions

Definition 4.4. Let K1 and K2 be two DB instances of some schema R with values in Const and Var (i.e., constants and variables). A homomorphism h: K1 →K2 maps the constants and variables in K1 to constants and variables in K2, such that

- h(c) = c for every c ∈ Const.
- for every tuple t, if R(t) is a fact in K1 then R(h(t)) is a fact in K2

△

Definition 4.5. Let M = $\langle$ **S**, **T**, $\sum$ $\rangle$ be a data exchange setting and I a source instance. A universal solution for I is a solution J, such that for every solution J’ of I, there exists a homomorphism h: J → J’.

△

Now we can apply these definitions to our Example 4.1 and illustrate an example of universal solution for the example mentioned above.

Example 4.2. This example extends the Example 4.1. In this example we replaced the variables which are not instantiated with the values from the left hand site of the dependency with artificial created variables including its artificial generated identificators. Later we call these variables labeled nulls.

Fact_Order(X_{Id}, MainOrderId → “1”, Price → “100 Eur”, OrderDate → “1.1.2012”, ProductStatus → “COMPLETED”)

Dim_Product(Y_{Id}, X_{id}, ProductName $\rightarrow$ "Walkman", $W_{ProductGroupId}$, SoldDate $\rightarrow$ "1.1.2012")

Dim_ProductGroup($W_{ProductGroupId}$, $Z_{ProductGroup\_Name}$)

This example satisfies requirements for the universal solution. Based on the definition 4.4 the homomorphism h maps all constants from the source schema to constants in the target schema and all variables from the source schema have been mapped to variables or constants in the target schema.

In order to provide the demonstration for the satisfaction of criteria from the second definition 4.5, we should have the whole collection of possible valid solutions. For our purposes, we take only one arbitrary valid solution and show that there exists homomorphism between the universal solution and the arbitrary solution. We materialize the variable $Z_{ProductGroup\_Name}$ with the value "Electronics", so we still have a valid solution.

Dim_ProductGroup(Id $\rightarrow$ "1", ProductGroup_Name $\rightarrow$ "Electronics") ▲

Now, we can see, that there exists a homomorphism $h(Y_{ProductGroup\_Name})$ = "Electronics" from the universal to this arbitrary solution.

4.2.4 Core of the Universal Solutions

Now we have the definition for the most general solution for a given data exchange problem called universal solution. Unfortunately, we have to treat this solution carefully; in fact the homomorphism property can satisfy more than just one solution.

Example 4.3. In order to demonstrate this, we add following additional tuple to the universal solution from the Example 4.2.

Dim_Product(Y_{Id}, X_{id}, $V_{ProductName}$, $W_{ProductGroupId}$, SoldDate $\rightarrow$ "1.1.2012")

We can simply show that this new tupple belongs to the universal solutions, because with the homomorphism function $h(W_{ProductGroupId})$ = "Walkman" there exists a homomorphism h from every valid solution to this solution.

▲

Since we have more than only one universal solution; there raises a natural question, if some of these solutions are "better" than the others. Fagin et. al [6] showed that every universal solution for a given data exchange has a common shared set of tuples. This set is called the core [6].

Definition 4.6. The core of the universal solutions for a source schema I an mapping M $core$(I, M) is the smallest common set of tuples from all J $\in$ UniversalSolution(I, M) and is unique up to isomorphism.

$\triangle$

The core for the above example 4.1 is the universal solution from the Example 4.2. The subject of schema mapping with relation to core is also a topic of other researches [32, 33]

4.3 Computing of Universal Solutions

Based on the introduced definitions and problem qualification, we still do not know, whether there is a way how to compute the arbitrary solutions or even the universal solutions for a given data exchange problem. It appears to be unfeasible to check the existence of a homomorphism from one solution to all other solutions. Fagin et al. [27] proposed a chase algorithm [3,27] which allows the calculation of the universal solution for a given data exchange setting under some specific dependencies [27]. This chase algorithm calculates the universal solution with a polynomial time complexity. It returns the universal solution as a result, under the preconditions that the algorithm terminates and the universal solution exist.

4.3.1 Chase Algorithm

The idea behind the chase algorithm is to start with a source instance and empty target instance $\langle$ I,$\varnothing\rangle$, and then chase by applying of $\sum_{st}$ and $\sum_{t}$ dependencies as long as these are applicable and not all dependencies are satisfied. For the introduction of the chase algorithm, we use its algorithmic form [34], which is easier to understand and logically identical with the definition provided by Fagin [27].

The algorithmic form of the chase algorithm:

Definition 4.7. Chase algorithm for M* = (S, T, $\sum_{st}$): given a source instance I, build a target instance J that satisfies each source to target dependencies TGD in $\sum_{st}$

- by introducing new facts in J as dictated by the right hand side of the source-to-target TGD
- by introducing variables in J each time existential quantifiers need witnesses

Chase the target J with the target TGDs and the target EGDs in $\sum_{t}$ to obtain a target instance J as follows:

- for target TGDs introduce new facts in J as dictated by the right hand side of the source-to-target TGD and introduce variables in J each time existential quantifiers need witnesses. Repeat this step until all dependencies are satisfied.
- for target EGDs $\varphi(x) \rightarrow$ x1 = x2
  - If a variable is equated to a constant, replace the variable by that constant;
  - If one variable is equated to another variable, replace one variable by the other variable.
  - If one constant is equated to a different constant, stop and report "failure".

△

Example 4.4. Assume that we want to compute the universal solution based on data exchange setting from Example 4.2. We provide the sequence of individual chase steps bellow, starting with $\langle$I, $\varnothing\rangle$.

1. $J_1 = \langle I, \varnothing\rangle$
2. J_2 = {Fact_Order(X_{Id}, "1", "100 Eur", "1.1.2012", $Y_{ProductStatus}$), Dim_Product(Q_{Pid}, X_{Id}, "Walkman", $W_{ProductGroupId}$,$V_{SoldDate}$)
3. $J_3 = J_2 \bigcup$ {Fact_Order(X_{Id}, "1", "100 Eur", "1.1.2012" , "COMPLETED")}
4. $J_4 = J_3 \bigcup$ {Dim_ProductGroup($W_{ProductGroup\_Id}$, Z_{Name})}
5. $J_5 = J_4$ [$V_{SoldDate} \rightarrow$ "1.1.2012"]
6. In this step the solution satisfies all dependencies and we get the following universal solution: J_5 = {*Fact_Order*(X_{Id}, "1", "100 Eur", "1.1.2012", $Y_{ProductStatus}$), *Dim_Product*(Q_{Pid}, X_{Id}, "Walkman", $W_{ProductGroup\_Id}$, "1.1.2012"),*Fact_Order*(X_{Id}, "1", "100 Eur", "1.1.2012", "COMPLETED"), *Dim_ProductGroup*($W_{ProductGroup\_Id}$, Z_{Name}) }

▲

The chase algorithm terminated in our case with an end solution – computed universal solution, which means that the solution exists.

However, the algorithm can in general terminate with up to 3 different results:

- It fails – In this case, the solution does not exist
- It succeeds – In this case, the result is the universal solution

- It never terminates – In this case it is not possible to resolve if the solution exists or does not exist

Based on this definition, if the algorithm never terminates it is not possible to determine if the data exchange setting has a solution or not. Fagin et al. [27] recognized a special cases which are responsible for the "infinite" loop of the chase algorithm. The reason of the not terminating of the algorithm is the special arrangement of dependencies, which in each chase step generates new tuples which has to be processed in the next chase step. This results into infinite loop.

We do not provide the explicit definition of the organization of these dependencies, as we do not plan to include this problem area into our considerations. However this definition can be found e.g. in [27]. In the rest of this paper, we consider only the data exchange settings without these special dependencies. However these dependencies might be present in data quality checks (in parent-child hierarchies) we do not treat it in this thesis. These dependencies are related only to target TGDs and the set of TGDs without these cyclic dependencies are called weakly acyclic.

Definition 4.8. Let M = $\langle$**S, T,** $\sum_{st} \bigcup \rangle$ be a data exchange setting, where is a set of source-to-target TGDs and t is the union of a weakly acyclic set of TGDs and a set of EGDs. Then the following tasks can be accomplished in polynomial time (with the respect to the size of the source instance I):

- checking if there exists a solution to I
- if a solution exists, computing a universal solution

△

Later we will see that despite the fact that we can compute the universal solution in a polynomial time, the chase algorithm has one negative side effect. This side effect of the chase algorithm discussed in many papers is a fact, that it brings a certain level of redundancy into the target [35]. This redundancy could be understood as generating of additional tuples which does not belong to the core of universal solution. The redundancy problem is examined later in the Section **??**.

4.3.2 Conclusion

After the introduction of data exchange framework and principles, we can see some similarities and common characteristics between the data exchange and data quality. Even if we provided definitions of both in the previous chapters, we did not approach them side by side. We do this in the next section. outperform traditional SQL algorithms.

CHAPTER 5

Data Exchange towards Data Quality

In general, the data exchange area is much more complex than as described in the previous chapter. The next chapter describes selected parts of this more complex area, with the focus on specific conditions related to the data exchange settings. Before we get to this part, we examine potential synergetic effects between data exchange and data quality.

5.1 Common Characteristics

In order to compare data exchange and data quality side by side, we first decided to decompose both subjects and identify common elements, objectives and tasks.

Data Exchange

Purpose	The rationale behind data exchange deals with a problem how to compute a solution (a materialized target instance) from a source instance under defined constraints
Objective	Find an efficient algorithm how to compute the target instance if exists
Components	Source and target schema, source-to-target dependencies, target dependencies
Initial status	With the records populated source schema, empty target schema, definition of source-to-target and target dependencies

Final status	With the records populated source schema, materialized target schema satisfying all dependencies
Process	The process generates new records in the target schema

Data Quality

Purpose	In the data quality we have to investigate, if the records in the target schema satisfy defined constraints.
Objective	Find an algorithm for data validation which allows efficient validation of many dependencies simultaneously
Components	Schema which has to be validated, schema which serves as etalon with records satisfying dependencies, dependencies between the records
Initial status	Both schemas are populated with data, defined dependencies which the data in the validated schema has to meet
Final status	Both schemas are populated with data, the records in the validated schema have quality flag, if they meet the dependencies or not
Process	The process does not generate new records

From the above descriptions, we see the existence of an almost equal set of components on both sides (schema(s), dependencies), but with a different objective and process. Regarding to this, if we decompose the data quality schema and study it from a different perspective, we can realize interesting findings. We can find out that a simple data quality check is by its nature similar to schema mapping. It consists from two schemas; it is source schema and etalon schema. The etalon schema could be understood as temporary target schema with valid data. The schema with "to be validated" data stays on the side and is taking into account after the etalon schema is materialized with source data. In addition, the process consists from a virtual data translation, which translates source records from the "source" schema into the "etalon" schema with respect to defined constraints. It means that even if the process in data quality generates new tuples, but in contrast to data exchange, these are used only to be compared with the existing target records. Moreover, the similarity can be found between the dependencies too. Dependencies in data quality are not strictly categorized into 2 groups (source-to-target and target) but the dependencies could be similarly divided into 2 corresponding categories: source-to-target dependencies correspond to data completeness checks and target dependencies correspond to business checks or data integrity checks. On the Figure 5.1 we can see the data exchange put in the role of data quality.

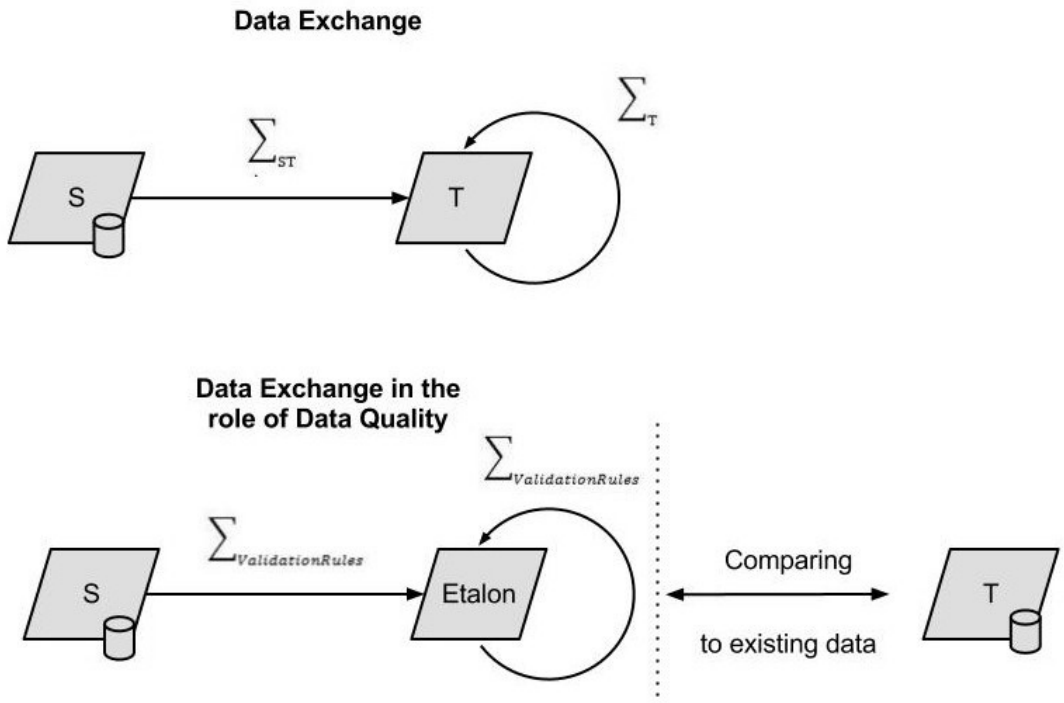

Figure 5.1: Graphical illustration of similarity between data exchange a data quality

5.1.1 Data Quality Constraints in the Role of Data Exchange

Before we discuss the detailed examination of data quality constraints, it is useful to define a new term – *"leading context"*. The leading context is necessary when designing the target dependencies in data quality (see below).

Definition 5.1. The *leading context* determines the main business context, in which the records have to be verified. This leading context might be e.g. an "order placement" context. In this case we have to verify only the record's dependencies related to the "order placement" context. This means to check if corresponding product-, customer- and date records exist, which the "order" record was created with. Based on the leading context, we can simply divide the examined data into logical subsets and approach them in a repeated process on an individual basis.

△

A practical use of leading context is the restriction of constraints into logical entities, e.g. in case of leading record "order placement", we would not validate constraints related to other contexts like product hierarchies, customer addresses which does not directly relate to the order context. We use the concept of leading context in the next

sections in order to put structure into data quality examinations.

Now we can return back to the examination of constraints. If going deeper in the perspective discussed above, the constraints behind the data quality checks could be of two natures. For a better illustration let us use the Example 4.1 from data exchange and put it into the role of data quality check. This check can be according to the type of constraints divided into 3 categories – completeness checks, data integrity checks and business context checks. In the Example 4.1 we have representatives for all of these categories:

- *Completeness checks as source-to-target dependencie*
  We would like to validate that all records from the source exist in the target in the corresponding tables defined by schema mapping. In business terminology, we want to check if all orders, including order details, which are in the source system, exist in the data warehouse. These types of dependencies are in the fact source-to-target dependencies known from the data exchange.

- *Data integrity checks as target dependencies*
  We would like to validate that in relation to the leading record there are relationships to other tables defined by the constraints. In business terminology, we want to check if there is a relevant product description in the table *DIM_PRODUCT* to every order record form the data warehouse table *FACT_ORDER*; and similarly a relevant product category record in the table *DIM_PRODUCTCATEGORY*. These types of dependencies are in fact target dependencies known from the data exchange.

- *Business context checks as target dependencies*
  We would like to validate that all sold products have sold date which corresponds to order date in which they were ordered. These types of dependencies belong also to target dependencies known from the data exchange.

A set of data quality checks does not necessarily need to cover all categories of data quality checks in every case. A simple data check (or set of checks) can validate e.g. source-to-target dependencies only or one kind of target dependencies only. But from the logical point of view, if we can merge two contexts into one data quality check and benefit from it i.e. there is gain in performance, it is natural to consider doing it in this way.

The similarity of data checks to data exchange is illustrated on the Figure 5.2 below.

The graphical illustration in Figure 5.2 shows that the dependencies in data quality can be put into the role of data exchange dependencies. This is an important result, but still does not determine if we can use the knowledge from data exchange in data quality.

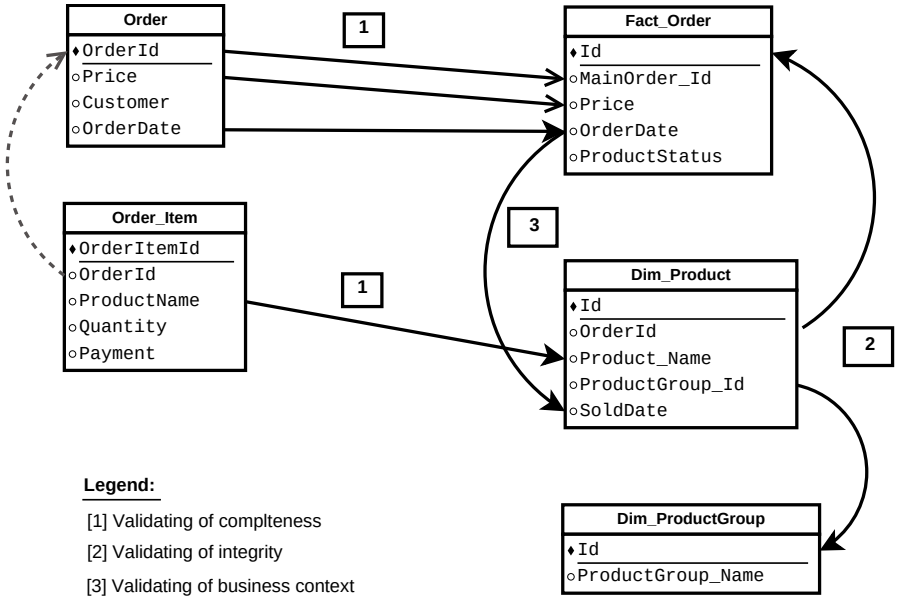

Figure 5.2: Graphical illustration of different nature of data quality checks in the context of data exchange scenario.

5.2 Data Quality Validation Process in the Role of Data Exchange

Now let us to take a closer look to data quality validation process and to the result side of both processes. The result of data exchange is a possible redundant in polynomial time computable universal solution; or usually with post processing [6, 36, 37] computable smallest among the solutions for the given problem - the core. In this situation, an interesting question arises: Can we use this result - universal solution or core in data quality? From the perspective of data quality mentioned above, we realize that the core consists of the ultimate and minimal set of record(s) validated against data quality constraints. This represents a valid data set, so actually the goal, which we want to achieve with our data checks. Unfortunately the situation with the core is not so obvious, as the process of getting the core from the universal solution has its own complexity. For this reason we study it in more detail in the later chapter.

Another interesting question is what happens if we use the classic chase algorithm for the data exchange when looking for good quality records. We can illustrate this scenario by extending the Example 4.1 and putting them into the role of data quality. For a better demonstration, we use in this example only one leading context and we split the chase algorithm to process first the $\sum_{st}$ and than $\sum_{t}$.

Example 5.1. We extend the Example 4.1 by adding a new record into the table *Order_*-

Items and also the $\sum_{st}$ by adding of one more condition. This helps us better understand the characteristic behavior of the chase algorithm.

The new source record for table *Order_Items*:

Rec.3: *Order_Items*(OrderItemId →"2", OrderId →"1", Product_Name→"MP3 Player", Quantity →"1", Payment →"CASH")

The new source-to-target dependency:

Dep.2: *Order*(OrderId, Price, Customer, OrderDate) → (∃ Id, ProductStatus) *Fact_Order*(Id,Main_OrderId, Price, OrderDate, ProductStatus)

Tables below show the records stored in the source schema.

OrderId	Price	Customer	OrderDate
1	100 EUR	University	1.1.2012

Table 5.1: Table Order

OrderItemId	OrderId	ProductName	Quantity	Payment
1	1	Walkman	2	VISA
2	1	MP3 Player	1	CASH

Table 5.2: Table Order_Items

We use the chase algorithm to compute the universal solution under the assumption that we apply the dependency nr. 5 in the first order and then subsequently dependencies nr. 1 - 4. Under these settings we get the following result, which is identical with the universal solution in the perspective of data exchange.

Id	Main_OrderId	Price	OrderDate	ProductStatus
X_{id_1}	1	100 EUR	1.1.2012	$Z_{ZProductStatus\_1}$
X_{id_2}	1	100 EUR	1.1.2012	$Z_{ZProductStatus\_2}$
X_{id_2}	1	100 EUR	1.1.2012	"COMPLETED"

Table 5.3: Table Fact_Order

▲

Id	OrderId	Product_Name	ProductGroupId	Sold_Date
X_{Yd_1}	X_{id_2}	Walkman	$W_{ProductGroupId\_1}$	1.1.2012
X_{Yd_2}	X_{id_2}	MP3 Player	$W_{ProductGroupId\_2}$	1.1.2012

Table 5.4: Table Dim_Product

Id	ProductGroup_Name
$W_{ProductGroupId\_1}$	$Z_{ProductGroup\_Name\_1}$
$W_{ProductGroupId\_2}$	$Z_{ProductGroup\_Name\_2}$

Table 5.5: Table Dim_ProductGroup

Before we discuss our results into details, we can see that the variables in the result are labeled with numbers. This is necessary for differentiating of variables which have to be instantiated with identical values. These variables are called labeled nulls.

After getting to know the labeled nulls, we can return to the analysis of computed result. It satisfies all dependencies, but as we can see it is not perfectly suitable for the data quality purposes. On the one hand, it contains records which are apparently validated records (good quality data set – record 3 in *Fact_Order*), but on the other hand there are additional records – (record 1 resp. 2 in *Fact_Order*), which are actually a subset of the 3. record. If we use these records for the validation we can get misinterpreting results. As we want to use this result to compare it 1:1 with existing target data, we have to eliminate these records. If we go further in the investigations and divide the good quality data set records from the rest of records we realize that the set of good records is identical with the core. In this case, the core can be used for validating of data records. The separation process of records which belong to the core from the other "redundant" records has been made in our case manually. This is possible only until we have small data amount. As in data quality we usually work with big data sets we need to find another approach. The possible way how to do this with the help of knowledge from data exchange is described in the next topic.Before we discuss our results into details, we can see that the variables in the result are labeled with numbers. This is necessary for differentiating of variables which have to be instantiated with identical values. These variables are called labeled nulls.

5.2.1 The Universal Solution Redundancy Problem

One of the differences between the data exchange and data quality is the amount of data intended for processing. The data exchange usually involves working with a smaller data amount. In the data quality, as opposed to data exchange, the data amount is much bigger. The previous demonstration showed that the computing of the universal solution when using the chase algorithm generates a certain amount of redundant records.

This amount is highly dependent on the nature of source-to-target dependencies [38] and the amount of source records [35]. Redundant tuples resulted from the generating of universal solution are not suitable for data quality evaluation. These tuples might be incomplete (e.g. constants are replaced with NULL), which causes that they are insufficient for validating of the data quality records. A detailed example of this behavior is described e.g. in [38].

Based on these findings, if we exclude redundant records from the universal solution, we get the core, the ultimate set of good valid records suitable for data quality purposes. In data exchange exist already algorithms which compute the core from the universal solution. The question, if these could be applied similarly in data quality, is the topic of the next chapter. Before we start with this, we mention shortly the intended final data validation process between the etalon and existing target data and then close and recapitulate this chapter.

5.2.2 How to use the Core in Data Quality Checks

The data validation process takes as an input the core computed from the universal solution. The core represents the etalon - the complete and unique set of valid records, as they should be present in the target schema which should be validated. This set could be easily used e.g. with a database minus operation in order to confirm the existence of these records in the target schema.

The core as the end result brings also very positive side effect. The core has already the target schema structure, so we can compare its data against the validating target schema on an individual table basis. Taking into account this, we can avoid the post processing, which is typical in traditional data quality checks. This post processing consists from the need to make complex join operations in the target schema to join all target tables by using of the foreign keys. The product of these joins is a one big target "view" and this is then the only subject intended for comparison. It is necessary to say, that the existence and definition of leading context play an important role by this comparisons. The approach with comparing the tables on individual basis can be used only in the case that we check the whole data amount in the target table related to the leading context. In other cases, we have to exclude the remaining redundant data not related to the currently checked leading context.

5.3 Summary

In this chapter, we provided definitions of the data exchange components and principles. Furthermore, based on these definitions, we compared a typical data quality example with a data exchange example. The result from the comparison is a general finding that a data quality check is with its structure almost identical to a data exchange

setting. Moreover, the core from data exchange represents the set of "valid" records for data quality check and we can use it to compare to be validated target schema with it. Naturally, the usage of the core in the perspective of data quality is reasonable only if we can benefit from it. In case of the practical database world, it means to increase the efficiency by data validating. In this context, one of the potential efficiency problems appears to be the existence of redundant records in the universal solution and subsequently the determining of the core from the universal solution.

In the next chapter, we analyze the performance efficiency of computing the core from the universal solution. Based on many research articles, there are many different parameters which influence the number of generated redundant records and also the computing performance efficiency itself. We use these results to compare them with the results from the traditional SQL validating process. In this case, our focus stays on the way to identify specific data quality situations in which the data exchange algorithm outperforms traditional SQL algorithms.

CHAPTER 6

Data Exchange Algorithms in the Data Quality Field

In the previous chapter we discussed the principles of data exchange and the way how they can be used for data quality purposes. In this chapter we primarily reference these findings and we continue with the comparison of data exchange and data quality. We focus on the algorithms used in data quality and data exchange from the practical point of view. Consequently, we extend this topic with the deeper discussion about factors and conditions, which impact the efficiency of these algorithms. At the end we show the potential of the chase algorithm in the data quality field. We propose a possible algorithmical approach how we can use the techniques for core computing to compute the set of valid records, in the way that appears to be more efficient than the traditional data quality check approach.

6.1 Data Quality Algorithm(s)

6.1.1 Traditional Data Check Algorithm

In the data quality field there does not exist any standard defined algorithm for data validation which is preferred over any other. There exist some usual techniques which form the concept of an algorithm. These techniques are represented by atomic operations which are frequently used in practice projects. The set of these techniques builds then the process of data validation.

The sources for these techniques are the definitions of dependencies, which are usually defined in the descriptive form in the business terminology language. As the business terminology is only a formal definition of requirements, it should be described in its logical form – consisting from the set of constraints and dependencies. This logical form

is then used as a base for its technical implementation – rewriting into database understandable statements. This "translation" of dependencies into the database executable pieces is usually done by its direct rewriting. The dependencies could be rewritten into relational algebra or direct into logical SQL statement(s).

The realization of the validation process itself starts with the execution of the SQL statements in the database. Every SQL statement is executed separately and the database engine returns the corresponding result set. The final result set of valid data is than calculated as the intersection of the results from all executed SQL statements. As the SQL statements are executed in the form as they were defined at the beginning, this process is not impacted by any non-deterministic actions (except rewriting from business terminology into SQL statements, but this is done under exactly defined rules), we can be sure that the final result set contains expected valid records.

This method is frequently used in data warehouses because of its simple implementation when translating dependencies into executable SQL script. Unfortunately, despite this advantage of its simplicity, the final SQL(s) are in common not very efficient. The execution complexity of this process grows rapidly with the number of dependencies. The inefficiency is caused by the fact, that every dependency is rewritten into an individual SQL statement and all these statements are at the end executed separately in the database.

The demonstration of such individual generated SQL statements is illustrated on the example below.

Example 6.1. (Continuing of the Example 4.1)

SQL statement, which validates source to target dependencies (data completeness) can have following form:

```
Select          t1.Id,
                t1.Price,
                t1.OrderDate,
                t2.Product_Name,
from            Order t1
left join       Order_Items t2 on (t1.OrderId = t2.OrderId)
```

SQL statement, which validates target dependencies (data integrity and business rules) can have following form[1]:

[1] We merged both checks into one statement, as the complexity of these dependencies allowed it. But in practice this is more often exception than the rule.

```
Select          t1.Id,
from            Fact_Order t1
join            Dim_Product t2 on (t1.Id = t2.OrderId)
and             t1.OrderDate = t2.SoldDate
```

▲

The problem area of efficiency / inefficiency of final generated SQLs is very complex and depends on many factors. As this traditional data check algorithm does not use any optimization technique; it is driven only by the form and design of the dependencies. This is not very practical; as two different dependencies sets can have the same logical definition but different execution complexity. Gottlob et al. [39] presented recently several criteria, which could be used for simplifying of dependecies but preserving its logical equivalence. But these should be used allready by design of dependecies.

6.1.2 Optimization of the Traditional Data Check Algorithm

Before we start with the optimization of the standard used algorithm, we should recall the reasons of its inefficiency as were outlined in the Chapter 3.2. We realized that the problem is very similar to the chase algorithm. It is the redundancy; in our case the "table scan" redundancy. In the frequently occurred cases, in which we have more dependencies related to one table, we have to redundantly scan the table n – times, whereby n is equal to the number of dependencies related to this table. Referring to the previous section, our optimization aim is to design a query with minimum execution cost[2] while preserving its logical definition. Applied for this case, if we can keep the redundancy of table scans on the minimal level and preserve the dependencies logical meaning, it will improve the query efficiency. A quite simple optimization approach appears to be the manual recognition of these "redundancy generating" dependencies and rewriting them into the more complex statements which cover more than one dependency. If we look in detail on this we can apply this for the target dependencies.

6.1.2.1 Optimization of Target Dependencies

If we consider target dependencies only; the way of rewriting them into one more complex statement appears to be significantly easier than by source-to-target dependencies. The reason is, that the dimensional structure (star resp. snowflake) on the target site does not require loops and cross joins for its validation. In this case we can apply following rewriting principle, which appears to be the efficient one. For the cases with one leading context e.g. "order placement" with the main table *Fact_Order*, we join to this table all referenced tables with left outer join (products, customers, order details, date, etc.) and

[2]Minimal execution cost is in a practice not explicitly determined, because it depends also on the characteristics of queried data

we create the complete data set suitable for validation. After the join is executed we can evaluate each referenced record even from different tables through the one table scan operation (e.g. implementation with *case when ... end*).

This case is illustrated on the next example which is the continuing of the Example 6.1. The example shows the merge of 2 target data quality checks into one, which counts the number of not existing references in the dimensional table *Dim_Product*.

Example 6.2. (Continiuing of the Example 6.1)

The individual selects have the following structure:

```
Select          count(*) as WrongRecordsProduct
from            Fact_Order t1
left join       Dim_Product t2 on (t1.OrderDate = t2.SoldDate)
where           t2.SoldDate IS NULL

Select          count(*) as WrongRecordsProduct
from            Fact_Order t1
left join       Dim_Product t2 on (t1.Id = t2.OrderId)
where           t2.OrderId IS NULL
```

After we use the proposed optimization technique, we can rewrite these 2 checks into the new form by merging of both previous selects:

```
Select          sum(case
                when t2.SoldDate IS NULL then
                1 else 0 end) as WrongRecordsDate,
                sum(case
                when t2.OrderId IS NULL then
                1 else 0 end) as WrongRecordsProduct,
from            Fact_Order t1
left join       Dim_Product t2 on (t1.Id = t3.OrderId and t1.OrderDate =
                t2.SoldDate)
```

▲

This select has the same logical meaning as the previous 2 selects, but needs only one table scan for the *Fact_Order* table. The merge technique used in this example can be also extended to other referenced tables.

Unfortunately, the demonstrated merge technique cannot be used for all kinds of data checks. We demonstrated a possible approach for target dependencies only. But considering the source-to-target dependencies we do not have any efficient approach

how to rewrite them into logical equivalent check(s) with smaller number of redundant table scans. An essential prerequisite for the applying of this approach is the existence of the star or snowflake schema. In data warehousing, this is mostly the case of target dependencies only.

6.1.3 Summary of Data Quality Algorithm(s)

The conclusion from this section bases on the fact that the traditional data check process is not very effective, especially for cases with many dependencies. The reason is the enormous number of redundant table scans caused by the direct rewriting of dependencies into executable SQL statement(s) without any optimization. We showed one optimization approach, but this is applicable only for selected dependency types and has to be applied manually. The search for a general applicable optimization approach complicates the wide scale of different data quality checks nature. Different natures bring their own specifics and make it more difficult to find a generally applicable principle. With respect to this, we can only argue that the optimization effort should focus on avoiding of redundant table scans with the redesigning of dependencies. The technique how to do it is missing. If we look at this problem from a wider perspective, we can see it also as a problem of effectively computing the valid result set based on defined dependencies.

Hopefully, the exploration of data exchange algorithm in the next chapter and consequently the similarity of data quality checks with data exchange help us to transform some data exchange principles into generally applicable effective data validation algorithm.

6.2 Data Exchange Algorithm

By the examination of data exchange approach we have to take much deeper involvement. In the previous chapter we identified the core as the final set of valid records which are suitable for data quality purposes but we didn't examine the core algorithms for its computing complexity. The computing process itself and its complexity belong nowadays to one of the challenges in the data exchange area. Several algorithms have been developed, which compute the core with the polynomial time complexity under very general conditions [40]. Question whether this polynomial time complexity is good enough to be more efficient as the traditional data check process, stays now in the center of our examination. Before we can do this, we need to select an appropriate core computing algorithm.

In general we identified 2 computing algorithms based on different approaches which can be used for the core computing. The first one removes redundant tuples from univer-

sal solution, the second one tries to avoid the existence of redundant tuples in universal solution.

6.2.1 Approach with Removing Redundant Tuples

The first examined approach expressed through its algorithmic form belongs to the most cited and popularized. It is presented in [40] respectively the extended version of the algorithm in [37]. The main idea behind these algorithms can be described with these steps:

1. Compute the redundant universal solution
2. Identify endomorphism of tuples to other tuples in the computed solution
3. Remove identified redundant tuples from the universal solution (shrink the universal solution to core)

It has been proved, that this is possible in polynomial time. Unfortunately despite this optimistic fact, the number of operations needed for the identification of homomorphism (even by using of optimized steps) is not that optimal. The practical implementation of this solution requires many database fetch operations to the computed universal solution for every tuple. The computational complexity refers to the polynomial time but with considering all database fetch operations the runtime grows rapidly with the number of evaluated records.

Decomposition of this algorithm can help us to identify the real life performance through the step by step process.

6.2.1.1 Decomposition

By the decomposition we use the algorithmic implementation form from [37], as it shows a feasible way how to practically implement the core computation for data exchange systems.

1. The first step of the algorithm computes the universal solution by using of the chase algorithm. This step already contains certain degree of redundancy depending on the source data and the nature of the dependencies. But in practical implementation this step does not require special time or memory demands. At the beginning of this algorithm TGD's generate new tuples what is for the computer engine a smooth process. Following, EGD's requires replacement of variables with constants or another variable; what is also by using of proper optimization similarly effective process.

2. The second important step checks the universal solution for possible existing homomorphism between the tuples. For this purpose it was implemented a complex table structure to support the storing of additional record relevant information. For every record with existing variable it contains variable name incl. its IDs (actually label identifier for labeled nulls). Another table holds information of all relevant values which this variable can have, when looking for homomorphism. The search process is optimized by dividing the variables into field partitions, relevant smaller groups of related variables (actually in the same column) which are considerable to be replaced. If we put this step in the context of data quality, we realize following findings which negatively impacts the performance.
   By the identification of a homomorphism between the tuples, we have to search for endomorphism[3] after replacing all variables with all relevant possible combinations of terms. The search for endomorphism for every record is performed by an individual select statement over the target table. In practice, in data warehousing we can count with up to many million records. This search can than lead to enormous number of selects in the target table. Moreover, we have to search for endomorphism of the whole tuples, which are usually created by records from more than one table. In this case we have to join these tables to effectively search for the homomorphism. In data warehouse with dimensional table structure we have usually only tables related together. This results into database queries, consisting of complex join operations, which may be very ineffective.

3. The last step is the removing of identified redundant tuples. As this step is only a one time activity, it does not negatively impact the performance.

6.2.2 Approach with Removing Redundant Tuples

Another approach has been proposed in [38]. This approach is different from the previous one. It does not focus on the problem from the algorithm view but tries to manage it primarily from the perspective of existing logical relationships between the source-to-target dependencies.

The main idea is based on the fact, that the existence of homomorphism between the dependencies in the data exchange setting (particular on the right-hand sides of source-to-target TGDs) causes producing of redundant records by the chase algorithm. The approach proposed in this paper identifies common cases with the occurrence of homomorphism and provides "manual" how it is possible to rewrite these dependencies into another form of dependencies with the same logical equivalence.

The rewriting process focuses on the TGDs in the source-to-target dependencies (it omits target dependencies) and rewrites them into new form but with the same logical equivalence. This is made with the help of negations. Moreover, this paper shows

[3] Endomorphism is a homomorphism which maps to an identical mathematical object resp. to itself.

a “translation” of dependencies into SQL executable pieces in the next steps. In the first step, the redesigned dependencies are converted into expressions in relational algebra. Negations became difference operators and mappings (“dependencies”) are merged with the union operators. This algebraic expression can be implemented with the SQL statement and executed in the database. The result set represents the core.

The process of redesigning of dependencies follows strict rules and can be implemented easily. The time complexity of this process is low.

We can demonstrate this principle on our example; according to this algorithm we have to rewrite our source-to-target dependencies Dep.1 and Dep.5. The reason is the existing homomorphism between these dependencies on their right hand-side.

Example 6.3. The new dependencies will have the following form:

Dep.1: *Order*(OrderId, Price, Customer, OrderDate) $\wedge$ *Order_Item*(OrderItemId, OrderId, ProductName, Quantity, Payment) $\rightarrow$ ($\exists$ Id, ProductStatus, OrderId, ProductGroupId, Sold_Date) *Fact_Order*(Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name, ProductGroupId, Sold_Date)

Dep.5: *Order*(OrderId, Price, Customer, OrderDate) $\neg$ (*Order*(OrderId, Price, Customer, OrderDate) $\wedge$ *Order_Item*(OrderItemId, OrderId, ProductName, Quantity, Payment)) $\rightarrow$ ($\exists$ Id, ProductStatus) *Fact_Order*(Id, Main_OrderId, Price, OrderDate, ProductStatus)

De facto, the dependency Dep.5, has been modified to omit records which come from the dependency Dep.1. These dependencies could be transformed into the following SQL statement:

```
Select          t1.Id,
                t1.Price,
                t1.OrderDate,
                t2.Product_Name,
from            Order t1
join            Order_Items t2 on (t1.OrderId = t2.OrderId)
```

```
UNION ALL

(
(Select          t1.Id,
                 t1.Price,
                 t1.OrderDate,"
from             Fact_Order
                 t1
MINUS
                 (Select     t1.Id, t1.Price, t1.OrderDate,"
                 from        Order t1
                 join        Order_Items t2 on (t1.Id = t2.OrderId))
)
```

▲

This generated script is better performing than the chase algorithm, but in our case for 2 dependencies it needs to perform 2 join operations, 1 minus operation, 1 union all and 1 table scan operation.

According to our definition from 4.1 the complexity is 3 join operations and 2 table scan operations. This complexity is quite big if we consider the amount of dependencies - 2.

6.3 The Redundancy in the Algorithms

6.3.1 What is Generating Redundancy?

As we presented above, redundancy is the main problem in searching for an efficient way how to generate "core" as a paragon to be later compared with existing data. All algorithms which we analyzed, worked with a certain amount of generated redundancy. The redundancy in our case is a product created by the computation algorithms. Furthermore, we could see that despite this redundancy, there are algorithms for the core computing in a polynomial time but it appears that their implementation doesn't allow efficient time processing in reality. The reason is the necessity for execution of many table scans in order to identify homomorphisms to other records. This fact is very important from the data quality checks perspective. It means that if we can prevent or minimize the generation of redundant records, we can get a more efficient algorithm for data quality checks. In order to do this, we have to understand the main problem – what actually generates redundant records.

By the decomposition of the problem, we analyze two existing kinds of dependencies: TGDs and EGDs.

6.3.1.1 EGDs

We begin with the simpler one – the EGD. The algorithmic implementation of EGDs dependencies in the chase algorithm is the replacement of existing variables with another variables or constants. Based on this, EGDs do not generate any new content (tuples), so they cannot be the source of the redundancy.

6.3.1.2 TGDs

Now let's take a closer look on TGDs dependencies. Its algorithmic implementation in the chase algorithm generates new tuples in order to satisfy its dependencies. If we retrospectively look on our Example 4.2, the TGDs generated new target tuples. Furthermore, in this example we could see an interesting behavior, that if we have only one TGD in a data exchange setting, we will not have redundant tuples in the universal solution. Next, after we add new dependency, as in the Example 6.2, we can see that this results in producing of redundant tuple in the universal solution. This behavior results from the similarity of the new dependency to the existing one. The need to satisfy the logical expression of the new dependency results in generating of new tuple. As the chase algorithm doesn't reflect the similarities between the dependencies, the existence of logical similarities between dependencies can result in producing of similar and also redundant tuples. According to [38] the redundancy is generated for the cases where a homomorphism exists among formulas in the right-hand side inside source-to-target TGD dependencies. It means that if homomorphism exists between the formulas, it generates redundant tuples in the target schema. From the practical point of view, this scenario occurs usually by dependencies, which cover relations to more than one table e.g.: We can have first dependency with relation to one table only; and another dependency having relation to the same table as first dependency and also relation to another table. Both dependencies use the same target table. This can result into the homomorphism between the right-hand sides of these dependencies. This is illustrated in the next example.

Example 6.4. If we carefully analyze the target side of our dependencies Dep. 1 and Dep. 5 (which produced redundant tuples), we can see that there exists a homomorphism *h* from Dep. 5 to Dep.1 which maps variables from Dep. 5 to variables in Dep. 1.

h: Fact_Order.Id5 $\rightarrow$ Fact_Order.Id1
Fact_Order.Main_OrderId5 $\rightarrow$ Fact_Order.Main_OrderId1
Fact_Order.Price5 $\rightarrow$ Fact_Order.Price1
Fact_Order.OrderDate5 $\rightarrow$ Fact_Order.OrderDate1

▲

According to this example, if we have 2 or more dependencies which define constraints for the same target table(s) and their target tuple(s) represent a subset or equal set of at least one other target tuple, there is a likelihood that its processing with the chase algorithm can lead to the redundant tuples.

6.3.2 Existence of Redundancy in a Typical Data Quality Example

How big is the probability that the redundancy occurs in a data quality example? As we already know, the existence of redundancy depends on the nature of dependencies. The biggest probability for its occurrence is in a scenario when 2 or more checks are related together and a homomorphism exists between them. Does this represent a real scenario which can occur in data warehouses?

The most important role in the answer for this question is played by the source data coming from the source applications (source systems). They usually have strict relational form and consist from many business domains usually covered by 1 main table and 2 or 3 side-tables, which are directly related to this main table. The side-tables usually contain attributes related to the main table e.g. order and related order item table(s) or order shipping information. Moreover, we have other tables, which we have to count with. Main table and side-tables usually contain references to other entities, usually called "list of values" like list of categories or dictionaries. In this context we can call these tables "reference tables". An example of such schema is shown on the Figure 6.1

Data, which are structured in this form in source systems, have to be loaded into data warehouse, specifically into the star or snowflake model. Records from the main table are usually merged with records from the side-tables and loaded into the fact table. In addition, records from reference tables are loaded into the dimensional tables and re-mapped to new keys corresponding to actual artificial keys in dimensional tables.

In this setting, we can identify 2 general scenarios, which can be a source of redundancy:

1. In this first scenario we have 2 TGD dependencies. Both dependencies have the same table on the source side and the second dependency has an additional table on the source side. Both dependencies use at least one common target table(s) and imply the existence of the same variables and constants. This scenario is presented also in our Example 4.1, built with dependencies Dep.1 and Dep.5. In this case both dependencies refer to the table *Order*. In our example has the dependency Dep.1 also an additional relationship with the table *Order_Items*. Both dependencies have the same common target table *Fact_Order*.

   This kind of dependencies has usually homomorphism on the right hand side and leads to generating of redundant records. The reason is the very similar structure

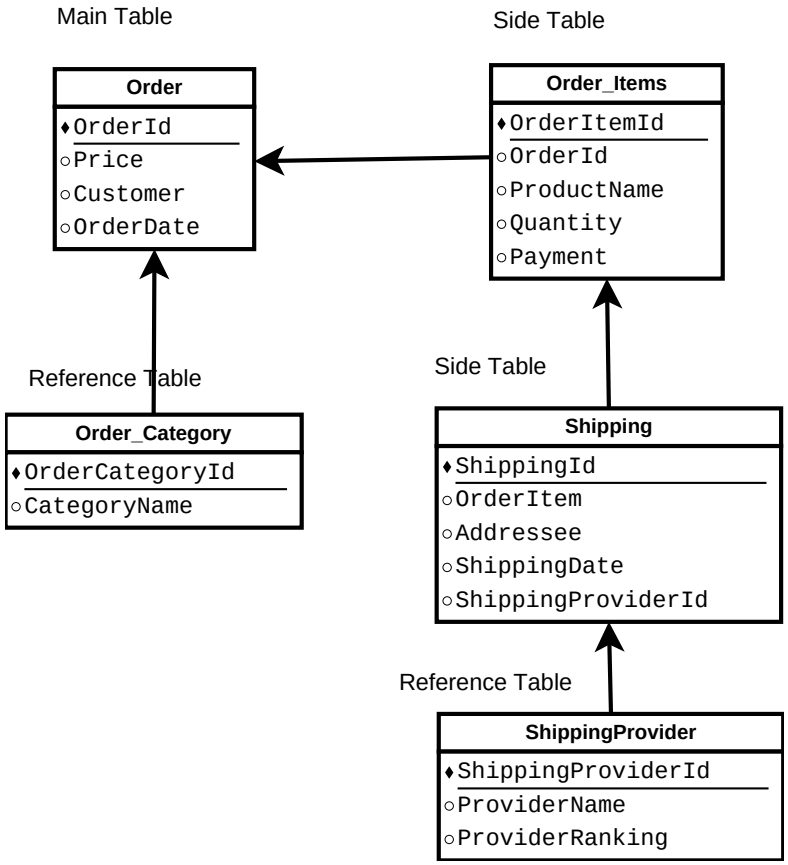

Figure 6.1: Example of source system data structure

of the dependency to the [38]. Fortunately, this kind of dependencies doesn't occur very frequently in practice. The dependency which contains homomorphism to the other dependency does not have any practical applicability. The logical validation which is defined by this dependency is already covered with the previous dependency and this dependency becomes redundant. This scenario can occur only by mistake due to inaccurate dependencies definition.

However, we have to consider such cases but they should not occur very frequently. Based on this, we have to count with a certain amount of redundancy in the universal solution.

2. In the second scenario we have 2 very similar cases:

   a) We have 2 TGD(s), concretely one source-to-target and one target dependency. The first source-to-target dependency uses as a target two or

more tables (T_1..T_n). The second target dependency uses as source one or more tables (T_1..T_n) and uses also at least one table (T_1..T_n) as the target.

Example 6.5. The following dependencies are an example of such discussed scenario.

The dependency on the source schema defined for the table *Order* assumes corresponding records in the target schema in the table *Fact_Order* and *Dim_Product*. This dependency also forces a relationship between these both tables.

Dep.1: *Order*(OrderId, Price, Customer, OrderDate) $\wedge$ *Order_Item*(OrderItemId, OrderId, ProductName, Quantity, Payment) $\rightarrow$ ($\exists$ Id, ProductStatus, Id, OrderId, ProductGroupId, Sold_Date) *Fact_Order*(Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name, ProductGroupId, Sold_Date)

In addition, the target dependency assumes an existence of record in *Fact_Order* table which refers to the product record in *Dim_Product* with the value of "ProductStatus" = "COMPLETED"

Dep.2: *Fact_Order* (Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name ,ProductGroupId, Sold_Date) $\rightarrow$ *Fact_Order* (Id, Main_OrderId, Price, OrderDate, "COMPLETED")

▲

b) We have 2 target TGDs, such that the first dependency uses the source table S and as target the table(s) (T_1..T_n). The second dependency uses at least one of the table(s) T as a source table and the S table as the target table. This scenario is based on the fact, that both tables were used as a source and target table. In this scenario we need to take into account that both dependencies should not be weakly acyclic, otherwise it is not possible to use the chase algorithm. In practice, this scenario can occur, if the source table S is a prerequisite to the existence of the target table T and in addition the source table should include some information which is implied by the existence of T table. In this case we cannot prevent the redundancy coming from this scenario. Fortunately, this scenario does not occur very often in practice. Based on the frequency of possible occurence of this scenario, we exclude this scenario from our considerations. Its non existence becomes one of the presumptions by looking for the algorithm.

6.3.3 Summary for Redundancy in the Algorithms

To summarize the previous considerations, we have to deal with possible redundancy in the universal solution in a typical data quality scenario. But based on the nature of data warehouse load characteristics/settings this amount is likely to be small. Even if there is only a small amount of redundancy it might be not very effective to use any of the algorithms discussed above, namely [37, 40]. Even a small amount of redundancy can lead to a decrease of performance of these algorithms, as we need to run SELECT in the target table for every record to validate the non-existence of a homomorphism.

6.4 Algorithm for the Practical Implementation

The presented algorithms for core computation, which were analysed in Section 6.2, might not be very efficient in the practical implementation for data quality purposes, but in our case they could serve as an inspiration for a new idea how to improve their practical efficiency. This efficiency is related to decreasing the run time in case of their practical database implementation in the data quality environment by using the concepts from presented algorithms [37,40]. With some knowledge from practical database world we can see that to build an efficient algorithm, we have to minimize the count of table join and table scan operations.

In this place we would like to present an idea for a practical implementation of an algorithm suitable for data quality purposes, which is based on the modified chase algorithm. The algorithm expects typical characteristics for data quality – dependencies which primarily define the snowflake data model (e.g. dependencies between fact table and dimensional tables), only small amount of homomorphism between dependencies based on [38]; non existence of standalone tables (every table has to be directly or indirectly connected to the main fact table) and non-existence of self- or circle joins between the dependencies.

The algorithm consists of 2 major steps:

1. The first step of the new algorithm computes the redundant universal solution using the chase algorithm.

2. The second step deals with removing of redundant records from the universal solution. As the data quality validation requires the ultimate minimal set of valid records which satisfies the data quality checks (dependencies), we need to remove all redundant records which are a subgroup of other records or are identical with other records i.e. with existing homomorphism or endomorphism to other records. In order to achieve this, we try to find a simplified solution satisfactory

for the data quality purposes. In our considerations, we would like to avoid the replacing of labeled nulls with variables which occurred on this place in other records. This process requires not only the very time consuming operation for the comparing of one record to other records, but this is multiplied by the amount of the "replaceable" variables. We were not successful to identify an algorithm in which we can avoid the comparing of the records but as an improvement of the existing algorithm we skip the step with the replacing of labeled nulls with the variables.

Related to this, we cannot forget that if a labeled null with the same label is used on more than one place it expresses the relationship, which cannot be validated in our proposed way. Fortunately, in typical data quality scenarios these relationships are dedicated only to define the relationships between the table(s) primary and foreign key(s). In this case if the target schema of the universal solution contains more than one target table, we propose a solution that we simply join all target tables according to primary and foreign keys (according to our definitions related to data quality scenario with the star schema, this should be possible) and then the algorithm searches for the homomorphism resp. endomorphism in this one-table structure. This join operation guarantees that all identical labeled nulls are used in the join conditions and these records are associated to the same records from the second join table. The join path will be always initialized from the centre i.e. the main fact table will be joined by using left outer join to all referenced tables. Based on our preconditions for this algorithm, it should be always possible to create this one-table structure. As it is not possible to identify the main snowflake fact table from the nature of the dependencies, it needs to be defined manually at the beginning.

Definition 6.1. The final one-table structure is called "valid master table". It consists of the final set of validated data including the redundant data, which has to be removed in post-processing. △

The proposed algorithm has the following form:

Input
S – represents the source schema
T – represents the target schema
t_f – represents the main fact table from the T schema

Output
U – represents the universal solution and at the end it represents the minimal set of valid data which satisfies the dependencies

Algorithm

(1) Chase (S,$\oslash$) with $\sum_{st}$ to obtain (S,T):=(S,$\oslash$)$^{\sum_{st}}$;

(2) Chase T with $\sum_{t}$ to obtain U:=T$^{\sum_{t}}$;

(3) Join all tables $t_{1..n} \in$ U with left outer join coming from the fact table tf to obtain U, such that it consists of one target table t

(4) Take the first record r_n with n=1 and obtain X_{core}:=$\sum_{x} \in r_1$

(5) Iterate with n=n+1 until $r_n \in$ U

(5.1) Mark r_n as redundant for all X_n:=$\sum_{x} \in r_n$ which satisfies $X_n \subseteq X_{core}$

(5.2) GoTo (5)

(6) Remove all records marked as redundant from U

Example 6.6. To see the process of removing the redundant tuples, we demonstrate it on the following example.

Source-to-target dependencies:

Dep.1: *Order*(OrderId, Price, Customer, OrderDate) $\wedge$ *Order_Item*(OrderItemId, OrderId, ProductName, Quantity, Payment) $\rightarrow$ ($\exists$ Id, ProductStatus, Id, OrderId, ProductGroupId, Sold_Date) *Fact_Order*(Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name, ProductGroupId, Sold_Date)

Dep.5: *Order*(OrderId, Price, Customer, OrderDate) $\rightarrow$ ($\exists$ Id, ProductStatus) *Fact_Order*(Id,Main_OrderId, Price, OrderDate, ProductStatus)

Target dependencies:

Dep.2: *Fact_Order* (Id, Main_OrderId, Price, OrderDate, ProductStatus) $\wedge$ *Dim_Product* (Id, OrderId, Product_Name ,ProductGroupId, Sold_Date) $\rightarrow$ *Fact_Order*(Id, Main_OrderId, Price, OrderDate, "COMPLETED")

Source instance:

OrderId	Price	Customer	OrderDate
1	100 EUR	University	1.1.2012

Table 6.1: Table Order

Universal solution:

OrderItemId	OrderId	ProductName	Quantity	Payment
1	1	Walkman	2	VISA
2	1	MP3 Player	1	CASH

Table 6.2: Table Order_Items

Id	Main_OrderId	Price	OrderDate	ProductStatus
X_{id_1}	1	100 EUR	1.1.2012	$Z_{ProductStatus\_1}$
X_{id_2}	1	100 EUR	1.1.2012	$Z_{ProductStatus\_2}$
X_{id_2}	1	100 EUR	1.1.2012	"COMPLETED"

Table 6.3: Table Fact_Order

Id	ProductGroup_Name
$W_{ProductGroupId\_1}$	$Z_{ProductGroup\_Name\_1}$
$W_{ProductGroupId\_2}$	$Z_{ProductGroup\_Name\_2}$

Table 6.4: Table Dim_ProductGroup

We can see that we have identical labeled nulls in the *Fact_Order.Id* and *Dim_Product.Order_Id*. These columns build relationship between the tables and in our case are parts of the join condition. After we join these tables into the "valid master table" structure, the relation of the records *Dim_Product* with OrderId = X_{id_2} to the records in *Fact_Order.Id* = X_{id_2} is guaranteed by the fact that in the valid master table structure they have the same combination(s) of *Fact_Order* values.

The result of valid master table is in the Table 6.5

▲

The last step is the identification of redundant records. In our case we need to compare the records based on the instantiated variable values and we can omit the labeled null in the comparison. As a result we get the records nr. 4 and 5 which represent the set of valid records intended to be compared with existing target schema.

We assume that this algorithm can work more effective than the standard used chase algorithm with the presumaptions which we defined in Section 6.4. The reason is, the more effecient table scan for redundant records, as we have only one target table, which needs to be scaned. The execution complexity is determined by the necessity to compare every record with other records to identify if it is contained in other record. Nevertheless the computational complexity of this algorithm appears to be lower than the standardly used algorithms in data quality; there are many factors which can influence it in negative way. We assume that the effectiveness of this algorithm in the data quality environment

F_Id	F_Main_OrderId	F_Price	F_OrderDate	F_ProductStatus	P_Id	P_OrderId	P_Product_Name	P_ProductGroupId	P_Sold_Date
X_{id_1}	1	100 EUR	1.1.2012	$Z_{ProductStatus\_1}$					
X_{id_2}	1	100 EUR	1.1.2012	$Z_{ProductStatus\_2}$	X_{Yd_1}	X_{id_2}	Walkman	$W_{ProductGroupId\_1}$	1.1.2012
X_{id_2}	1	100 EUR	1.1.2012	$Z_{ProductStatus\_2}$	X_{Yd_2}	X_{id_2}	MP3 Player	$W_{ProductGroupId\_2}$	1.1.2012
X_{id_2}	1	100 EUR	1.1.2012	“COMPLETED”	X_{Yd_1}	X_{id_2}	Walkman	$W_{ProductGroupId\_1}$	1.1.2012
X_{id_2}	1	100 EUR	1.1.2012	“COMPLETED”	X_{Yd_2}	X_{id_2}	MP3 Player	$W_{ProductGroupId\_2}$	1.1.2012

Table 6.5: The valid master table

will be proved in scenarios with many dependencies. The small disadvantage of this algorithm is its applicability, which is possible for the star schema only.

In the next chapter we plan to propose a practical implementation of this algorithm in the form of a prototype in order to confirm or discard our presumptions.

6.5 Conclusion

In this chapter we have revisited the existing algorithms for core computing and we have introduced a new version of computing algorithm for data quality purposes. This version computes the set of validated data from the universal solution in the post processing and its value should be recognized primarily in data quality area. Our expectation for the efficiency of this algorithm is based on our experience from typical database processing and possibly doesn't have to correspond with the real implementation and practice. In the next chapter we consider the implementation aspects of the algorithm and try to find the best implementation strategy. Moreover we provide an implemented form of the algorithm suitable for practice. Based on this implemented version we will run experimental tests to prove the efficiency / inefficiency of the algorithm in the last chapter.

CHAPTER 7

Implementation & Evaluation

7.1 Practical Implementation

In the previous chapter we introduced a modified version of the chase algorithm which can be used for data validation in data quality. In this chapter we focus on the implementation details of this algorithm and evaluate which implementation strategies could lead to the solution with a good run-time performance. The algorithm itself is implemented as a part of executable program and this chapter discusses the algorithm in the context of this program. In order to provide better understandability, we split its implementation into more logically related subparts to manage them separately. A necessary part of the implementation is formed by the set of "supporting functions" which are needed to provide the basis services (e.g. load data, prepare / clean environment). As these functions serve only to support the core algorithm we omit them in our description and our main focus stays on the implementation of the algorithm itself.

We use in this implementation part the same dependencies and values as we defined them in the theoretical examinations, specifically in Section 4.2. The source code of the implementation is publically available on https://code.google.com/p/data-quality-with-data-exchange/source/browse/.

7.1.1 Implementation Considerations

Before we begin with the implementation of the algorithm, we need to answer the question, which environment will be used for the implementation. Both, the ETL (as is standard used in the Data Warehouses) or one of the database languages are good candidates. Our first consideration leads to comparison of functionality, in which we can argue the ETL capabilities could be limited. Nowadays, the most reputable ETL tools

provide the support of a native database language - this can provide equivalent functionality as the native database languages. In addition, the benefit from using the ETL tool is the framework which deals with detailed technical implementations on the lowest level. Unfortunately, this is in our case also the main argument against the ETL tool. In our case we lose the flexibility in the control of data processing. The data manipulation strategies are for specific cases not available in the user interface and the ETL can non-deterministically chose between them. Consequently, our complexity evaluation could by negatively impacted by this. Therefore, we decided to use the native database language. In our case, one of the most popular databases Oracle and the programming language PL SQL.

7.1.2 Functional Design

The core functionality of the program is data validation. The input data are validated against the validation rule and the result is the set of data, which meet the validation criteria. The validation rules are defined in the form of dependencies. The algorithm used in this program has been inspired by the chase algorithm including the enhancements which are possible with respect to the target snowflake structure. The functionality of the program should guarantee the following objectives:

1. Read and build the meta structure for the dependencies
2. Process of source-to-target TGD dependencies
3. Process of target TGD dependencies
4. Identification of homomorphisms and removing redundant data
5. Comparison of the computed result set with the existing data and filtering out invalid data

7.1.3 Program Objective

7.1.3.1 Read and Build the Meta Structure for the Dependencies

Our program holds the dependencies stored in the database relational tables. The logical concept of this relational data model has generic flexible structure which allows the building of different kinds of dependencies, including storing of dependency constants. Besides the storing of dependencies, the data model stores operational data which are necessary for monitoring of the computing process itself. Temporary table structures, which depend on the dependencies, are created later after the execution of the program "on the fly" and therefore are not a part of the data model. The temporary part of the data model consists from the copies of target tables and one master table, which stores

the final dataset. We call this table “valid master table” and it corresponds to the “final one-table” from the Section 6.4. The schema of the data-model is given in Figure 7.1.

From the technical point of view, the model was designed as simple as possible. It does not contain any Oracle specific elements so that it would be compatible with a wide range of database languages. Only standard DDL commands have been used to describe the table structure, primary and foreign keys.

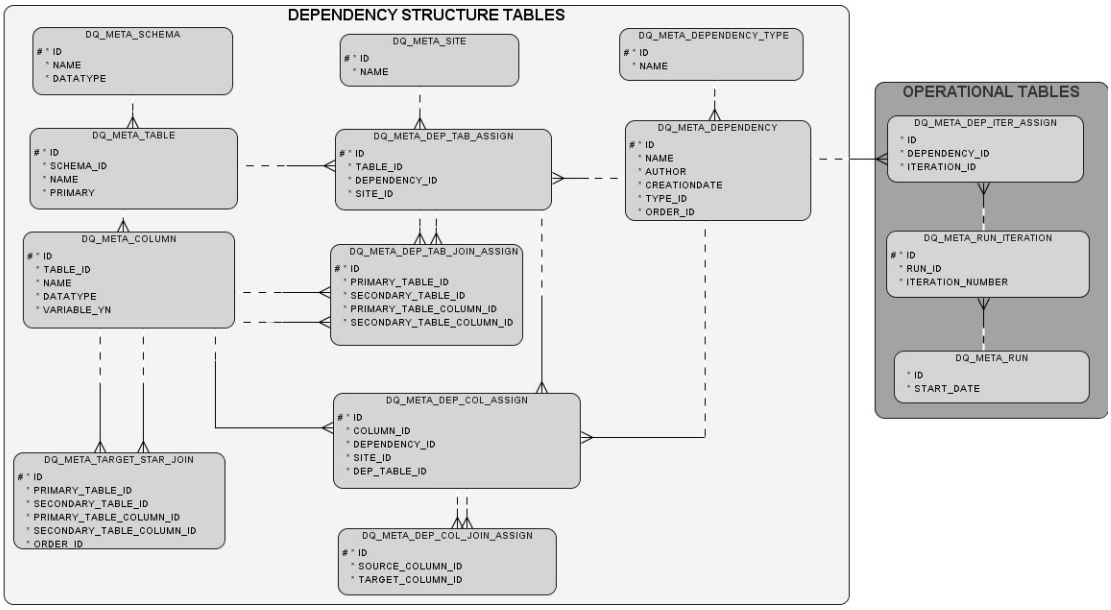

Figure 7.1: Schema of the existing data-model and the temporary data-model

7.1.3.2 Processing of Source-to-Target TGD Dependencies

The processing of the dependencies starts after the program creates all temporary target structures. The program begins with the source-to-target dependencies. The processing algorithm must guarantee that at the end of this step all source-to-target dependencies are satisfied. The logical concept of the algorithm uses the same principle as the chase algorithm. The algorithm takes in the first step all source-to-target dependencies and identifies related target tables. Accordingly, it continues with a loop which iterates per every dependency and further per every identified target table in this dependency and computes an “insert statement”, which should be used to populate this target table with relevant source data. In the case that the data for a specific target table comes from more source tables, the insert statement prepares a join operation to join all relevant source

tables, in order to get the data for the target table. An example of such insert statements generated for the Dependency nr.1 from the Example 4.1 is illustrated in Figure 7.2.

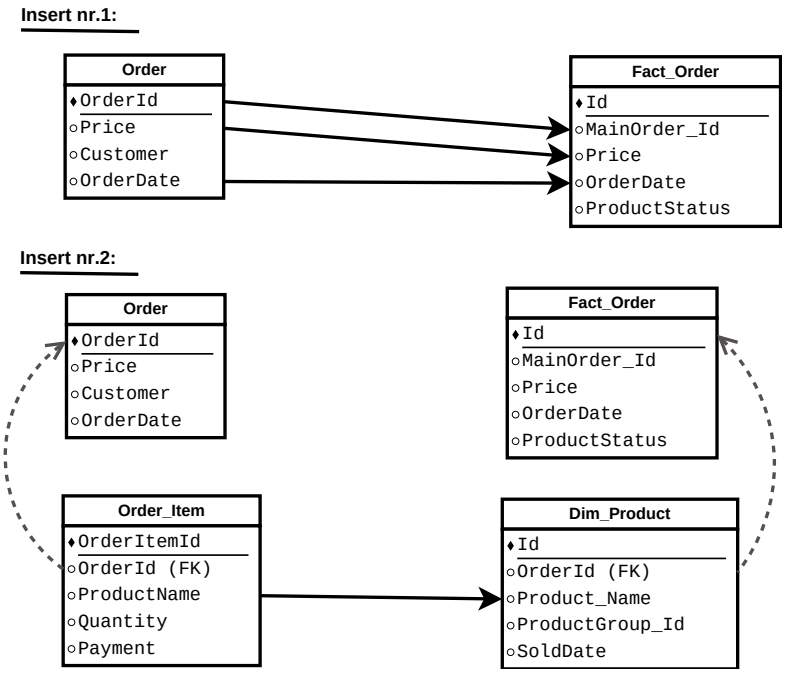

Figure 7.2: Example of inserts generated for the dependency nr. 1

In this step, the variables which are specified through the existential quantifier must be replaced with the labeled nulls. The "label" for these nulls is generated by a specific mechanism. This mechanism generates a unique "label ID" based on the source data which are used to generate the target records. It means that it takes all records from the source table(s), which are relevant to this target record. The value of all members from these source records is used to generate a unique hash key. This ensures that the labeled nulls get always a unique label for unique source values.

Example 7.1. This example shows the concept of generating "label ID" for the labeled nulls.

If we consider the dependency number 2 from our dependency definitions, the label for the target tables *Fact_Order* and *Dim_Product* consists of the concatenation of the following values:

1. General label identifier: "##" + table_ID + "_" + variable_name + "##"

2. Hash value calculation: hash(Order.OrderId || Order.ProductCode || Order.Date || Order.Price || Order_Details.OrderId || OrderDetails.ProductName || OrderDetails.ProductColor).

   The result might be e.g. ##H3_STATUS##57381

▲

The result from this first computing process is the temporary target structure filled with records and labeled nulls, in case of existential quantified variables. All source-to-target dependencies are satisfied.

7.1.3.3 Processing of Target TGDs

The procedure in the processing of the TGD dependencies is very similar to the processing of source-to-target dependencies. The first step is the computation of the target records which have to satisfy the target dependencies. This step requires additional complexity comparing to the source-to-target dependencies. As the target dependencies can have on the right hand side target tables, which are used by other dependencies on the left hand side, it could happen that after the first iteration, the new records imply that another dependency is not satisfied. Based on this, the algorithm might require more than one iteration to compute the target records so that all target dependencies are satisfied.

In order to handle this situation, we considered two possibilities how to do it:

1. The first approach would be the validation of all dependencies against the target records at the end of every iteration, in order to confirm that all dependencies are satisfied. But this approach is algorithmically very ineffective as it requires the scanning of all records for every dependency.

2. The second possible approach would be based on determination of those relationships between the dependencies, which might be invalidated by loading of new target data. If such dependencies exist, then the next iteration is executed with these dependency(ies) only. This is a repeatable process until there are no affected dependencies. In this case all dependencies should be satisfied. A rule to identify the affected dependencies bases on the analysis of the dependencies; if a variable on the right hand side is loaded through the dependencies from the left hand side and this variable is used in an other dependency on the left hand side then it is necessary to run the next iteration with this dependency. The identification mechanism requires also the handling of some specific cases which could occur, but these scenarios could be quantified and handled within the algorithm. At the end of this computing step, all dependencies are satisfied.

7.1.3.4 Identification of Homomorphisms and Removing Redundant Data

The purpose of this step is to compute the final valid data set. At the beginning, the program creates and loads the "valid master table". The "valid master table" is built by left join(s) of all tables which surround the central target table in a snow flake constellation. This table becomes our base table in order to start with the searching for homomorphisms. The join criteria are defined in a separate meta-table and are usually identical with the relationships between target tables on the right hand side.

After this step is accomplished, we can proceed with a master table which contains the following kind of values/variables.

1. Regular record values – Instantiated with values
2. Labeled nulls – Identified by the pattern "##TableId_ColumnName##HashValue"
3. Null values – Database null values

An Example of the master table is illustrated in the Figure 7.3.

H3 ID	H3 MAIN ORDERID	H3 PRICE	H3 ORDERDATE	H3 STATUS	H4 ID	H4 ORDER ID
##-H3 ID-##57381	1	100 EUR	4.3.13	##-H3 STATUS-##57381		
##-H3 ID-##138208	1	100 EUR	4.3.13	##-H3 STATUS-##116668	##-H4 ID-##85387	##-H3 ID-##138208
##-H3 ID-##106789	1	100 EUR	4.3.13	##-H3 STATUS-##94490	##-H4 ID-##111680	##-H3 ID-##106789
##-H3 ID-##138208	1	100 EUR	4.3.13	COMPLETED	##-H4 ID-##85387	##-H3 ID-##138208
##-H3 ID-##106789	1	100 EUR	4.3.13	COMPLETED	##-H4 ID-##111680	##-H3 ID-##106789

Figure 7.3: Example of computed "valid master table"

Now, the "valid master table" contains data which have been succesfully validated. However, it contains also redundant data and these have to be eliminated during the next steps. It requires that for every record in the table the rest of records are compared to it. If there exists at least one identical record then the original record is redundant. The base definition for the "identical record" is specified by the rule that two records have the same values independent from the labeled nulls 'labels'. In general, our definition must not be satisfactory for all cases, as the labeled nulls have two different meanings:

1. A labeled null represents a variable which could be replaced with any value, if there does not exist any other labeled null with the same label.
2. All labeled nulls with the same label has to be replaced with the same variable. In the case of the dimensional data model this situation can occur only in the scenario that variables with same label define the relationship between tables.

In the first case we do not need to consider the labeled nulls in our algorithm as these could be replaced with any variables and it will not have any impact.

In the second case we need to take them into account, because if one labeled null is instantiated with a specific value then all labeled nulls with the same label need to be replaced with the same value. By the handling with these labeled nulls helps us the assumption that this case can occur in the real data warehouse scenario only by foreign key(s). This assumption is based on the dimensional model of data warehus and the data model of source systems. As the value of foreign keys is used as a join condition by the joining of all tables to create the master table, we do not need to consider this case. The join of these tables guarantees that identical labeled nulls in different records are paired together.

The latest problem could appear in the case that we have dependencies which are a subset of other dependencies. In this case we similarly do not need to consider this scenario as we build a final and distinct set of data which we expect on the target side. If the record is a subset of other record(s) it should be eliminated from the data set. The definition for a subset means that the original record has standard "null" values on the positions where the scanned record has labeled nulls or values.

One of the disadvantages by using our implementation is the fact, that we are not able to define any "or" logical evaluations. In a typical data quality scenario we can use for the definitions of rules also the "or" conditions. In our case we strictly calculate an exact data set which represents the valid target data and this could be defined with "and" conditions only.

Example 7.2. This example illustrates the removal of redundant records from the "valid master table".
The Figure 7.4 shows a part of the computed "valid master table".

H3 ID	H3 MAIN ORDERID	H3 PRICE	H3 ORDERDATE	H3 STATUS	H4 ID	H4 ORDER ID
##H3 ID##57381	1	100 EUR	4.3.13	##H3 STATUS##57381		
##H3 ID##138208	1	100 EUR	4.3.13	##H3 STATUS##116668	##H4 ID##85387	##H3 ID##138208
##H3 ID##106789	1	100 EUR	4.3.13	##H3 STATUS##94490	##H4 ID##111680	##H3 ID##106789
##H3 ID##138208	1	100 EUR	4.3.13	COMPLETED	##H4 ID##85387	##H3 ID##138208
##H3 ID##106789	1	100 EUR	4.3.13	COMPLETED	##H4 ID##111680	##H3 ID##106789

Figure 7.4: Example of computed "valid master table"

If we use the input from the table in Figure 7.4 for redundancy elimination, we get the following results:

▲

Record Nr		Redundancy evaluation		Result
1	→	Record is subset of the record 2	→	Remove the record 1
2	→	Record is identical with the record 3	→	Remove the record 2
3	→	Record is subset of the record 4	→	Remove the record 3
4	→	Record is identical with the record 5	→	Remove the record 4
5	→	Record is unique	→	Keep the record 5

7.1.3.5 Data Comparison

The data validation procedure takes in the first step all relevant target tables with real target data and builds from them one table constellation in order to allow 1:1 comparison to the computed result set in the valid master table. In this comparison, we need on the one hand to compare the variables and on the other hand the labeled nulls which have a relationship to other labeled nulls (identical label). During the comparison of labeled nulls we need to check if the real target data have on the places with identical labeled nulls also identical values. The algorithmic implementation of this comparison is made by the one to one comparison of the "real master table" records and valid master table records. The result from the validation check is the set of real target data which has been successfully validated and could be marked as "good".

7.1.4 Implementation Details

7.1.4.1 Technical Considerations

By approaching the technical design we should first consider, that in a typical data warehouse scenario our program can expect a large amount of data which must be processed. The processing of such an amount of data e.g. table joins between large tables could be technically implemented either inside the PL SQL block by simulating the database join operation or could be executed as a database command. The first option might be more flexible, as the data are once loaded into the memory and then they are available also for other operations, if needed. The second option offers better performance by standard atomic operations (e.g. table join) but for every atomic operation the data have to be accessed in the tables. Having contemplated these facts, we decided to prefer the second option, mostly due to the fact that with large data it is better to let the database decide about the strategies how to handle such data. In this case all data manipulation operations are executed directly with database DML (data manipulating language) commands. The program builds the database statements and lets the database execute them.

7.1.4.2 Structure

The program itself consists of several procedures which are encapsulated inside a package. The main function "START_DATA_QUALITY_VALIDATION" executes the program and calls subsequently next procedures which process the data.

The procedures have the following meaning:

1. DQ_META_BUILD_INFRASTRUCTURE - This procedure creates the temporary target tables (1:1 copy of the real target tables which should be validated). These tables are used in the next procedures.

2. DQ_META_SOURCE_TARGET_DEP – This procedure processes the source-to-target dependencies and loads the temporary target tables with relevant data to satisfy source-to-target dependencies.

3. DQ_META_TARGET_DEP – This procedure processes the target dependencies and loads the temporary target tables with relevant data. If the target dependencies are not satisfied within the first run of this procedure, then the computing mechanism starts a next iteration.

4. DQ_META_BUILD_MASTER_TABLE – This procedure builds the valid master table from the temporary target tables. Furthermore the procedure creates another master table consisting of the real target data. We call this table real master table. This table will be used in the last step when both master tables will be compared.

5. DQ_META_REMOVE_REDUNDANT_DATA – This procedure scans the master table and identifies records with existing homomorphism or endomorphism to other record(s). This ensures the elimination of identical records and also records which consist of a subset of other record(s). It produces the final set of data used to validate the existing target data.

6. DQ_META_VALIDATE_TARGET – This procedure compares the records from both master tables and based on the equivalence of data identifies valid records.

7.1.5 Summary

If we look on the implemented solution from the helicopter view, we can realize that it represents only the core of the algorithmic part. Putting this solution into real data-warehouse architecture still needs the validation of our prerequisites and general prerequisites for core computation (e.g. weakly acyclic dependencies). Even if this implementation is not directly ready to be used in a practical implementation it could be used as a starting point for building such a system.

7.2 Evaluation

7.2.1 Procedure for the Performance Evaluation

In the performance evaluation we focus on the comparison of our program against the standard used data quality algorithm as described in 6.1. We approach the comparison with the idea to compare the count of table scan and table join operations needed for its successful completion. The definition for the assigning of complexity (table scan resp. table join) is given in section 3.2. This kind of comparison brings the exact performance calculation for a particular case. As these database operations usually consume most time by accesing the database storage drive, we see the amount of table scans and table joins as one of the most reliable metrics to define the complexity in our case. In our case, as the variability in the dependency types is a significant factor which impacts the performance, it might be not enough to define the performance calculation for a particular case(s). Consequently, we would like to define a new measure - "complexity increase" as the additional complexity which is needed if we add to the scenario one new dependency. In the next section, we first calculate the complexity for a typical data quality example, processed with our program and then processed with the typical data quality procedures. Next, we calculate the complexity increase for a typical additional dependency.

7.2.2 Complexity Estimation

The complexity estimation uses the variable n as the count of target tables to be validated.

The implementation of our algorithm completes in following steps with following complexity:

1. **Processing of source-to-target dependencies**
   It requires the performing of at least one table scan operation and the number of join operations which is equivalent to the number of remaining tables on the left hand side.

2. **Processing of target dependencies**
   It requires the performing of at least one table scan operation and the number of join operations which is equivalent to the number of remaining tables on the left hand side. This step can run in more iterations, but in the typical data warehouse scenarios such situations do not occur very frequently. The reason is the data constellation in the source system which usually does not have deep or cyclic relationships. Nevertheless if they occur, then they require maximum 1-2 addi-

tional iterations. Based on this, we decided to marginalize this factor from our comparison.

3. **Creation and load of the master table for the temporary target tables**
   We join all temporary tables together to create the star schema. It requires the complexity which is equivalent to n-1 table joins.

4. **Identification and deletion of the redundant records**
   All records in the master table have to be compared with the rest of records, to identify the redundant records. The complexity behind this step is 1 table join.

5. **Creation and load of the master table for the real target tables**
   This step requires the join of all tables together similarly as in the point nr. 3. It has the same complexity n-1 table joins.

6. **Comparing of results**
   The comparison requires that every record from the real master table is compared to every record in the temporary master table. By appropriate implementation of this step we can reach the complexity of 1 table join. Unfortunately this step cannot be performed by the database MINUS command (because of validating of identical labeled nulls), so it requires complex PLSQL implementation to reach this complexity. Our implementation does not implement this complex logic, but we use in the next section the realistically achievable complexity of 1 table join.

Example 7.3. This example describes the complexity of the operations.

1. Processing of source-to-target dependencies

Dependecy Nr.	Table scans	Table Joins
1	1	0
2	0	2

2. Processing of target dependencies

Dependecy Nr.	Table scans	Table Joins
3	1	0
4	0	2

3. Creation and load of the master table for the temporary target tables

Temporary Target tables	Table scans	Table Joins
Fact_Order vs. Dim_Product	0	1
Dim_Product vs. Dim_PorductGroup	0	1

Temporary Target tables	Table scans	Table Joins
Valid master table	0	1

Target tables	Table scans	Table Joins
Fact_Order vs. Dim_Product	0	1
Dim_Product vs. Dim_PorductGroup	0	1

4. Identification and deletion of the redundant records
5. Creation and load of the master table for the temporary target tables
6. Comparing of results

Relevant tables	Table scans	Table Joins
Temporary maste table vs. real master table	0	1

If we summarize the complexities from the previous calculated steps, we get the complexity of: **2 table scans and 10 join operations**.

▲

7.2.2.1 The Complexity of the Standard Algorithm

The standard used data quality algorithm approaches this problem from another perspective. It goes through every dependency and builds the temporary left hand resp. right hand data view. After this it uses the database MINUS operation to compare the results.
In our case it means:

Dependency Nr.	Table scans	Table Joins	Table Minus
1	0	1	0
2	0	3	0
3	0	2	0
4	0	1	0

Table 7.1: Complexity of the classical data quality algorithm

The standard algorithm has the complexity 7 join operations and it is actually better performing than our program. This result is unfortunately negatively influenced by the

“one-time“ steps in the algorithm, which are independent of the number of participating source and target tables. In order to bring a more detailed evaluation, we approach to the evaluation of a hypothetical example and a scenario that one new dependency has been added to this example.

7.2.3 Complexity Evaluation for an Additional Dependency

Objective: Adding one new dependency consisting of *m* table(s) on left side and *n* table(s) on the right side.

Operation	Table scans	Table Joins	Comment
Processing of source-to-target or target dependencies (depends on the nature of the dependency)	0	(m-1)*(n-1)	(m – 1) table joins to prepare the left side multiplied by the count of target tables (n-1)
Preparing the valid master table	0	n-1	
Preparing the real master table	0	n-1	
Removing of redundant records	0	1	
Comparing of both master tables	0	1	

Table 7.2: Complexity of our implementation of the algorithm

Operation	Table scans	Table Joins	Comment
Processing of the left side to prepare the left side data view	0	m-1	
Processing of the right side to prepare the right side data view	0	n-1	
Comparing of both results	0	1	

Table 7.3: Complexity of the classical algorithm

We can see that also for this operation the complexity is better in the classical algorithms. The highest complexity ranking has in our case the first operation of our implemented program. The reasons are the target iterations, in which we handle every target table in a separate iteration. Due to this fact the number of left side joins is multiplied by the count of target tables. Technically there should not be a problem to change

the implementation that all target tables within one dependency are loaded within one step. The only limitation here is the PL/SQL capability. If we consider that we can improve this, we get the complexity of (m-1) * 1 = m-1. Even in this case we do not have the complexity of the standard algorithm.

7.2.4 Complexity Evaluation Discussion

The performance and complexity evaluation of the algorithm showed that the performance of the algorithm does not achieve the performance of the traditional algorithm. The main reason is the fact that even if we can avoid the joins on the target side during the initial data load into these tables we have to count with additional joins which we need to process the data into the "valid master table". Based on our findings we can expect the best performance against the traditional data check in the cases where we have many dependencies which work with small amount of different target tables.

7.2.5 Practical Evaluation

Based on the complexity evaluation results, the next step is to know, how the complexity impacts the execution time of the algorithm. We are going to make several practical executions of the algorithm, in order to see the correlation between the complexity and the performance (execution time). Presumably the performance of the new algorithm might be comparable or slightly slower than the traditional algorithm.
As a next step, we would like to see the applicability of the algorithm in a practical scenario, in the meaning of results correctness and processing of large data amount.

7.2.6 Test Case for Practical Evaluation

During the preparation of the scenarios for the practical execution, we realized that our first intended aim could not be accomplished in the scope of this examination. More concretely, the relationship between the complexity and performance could not be directly computed. By the definition of the test case, we found out that the factors like structure of the data model, the nature of dependencies and the structure of the records have high impact on the complexity and also on the performance results. Every small change of these factors can lead to completely different results in the particular areas. These factors have to be always taken into consideration by the evaluation of the complexity. As our complexity results do not differentiate the number and relations between the records, it is not possible to provide a universal statement valid for comparison of complexity versus performance.
The variability of these factors is almost unlimited and even in data warehouses, there does not exist something like a typical data warehouse example, which could be used in a universal case.

Due to the fact that we are not able to examine the relationship between the complexity and performance, we decided to prepare a test case with which we focus on findings from our complexity examinations. After detailed look on the results, it showed us an interesting fact, which we would like to use for this demonstrative test. The algorithm appeared to be not efficient with large amount of tables, but if we consider a small amount of target tables and we use high complex dependencies, we hope to see more comparable results.

We consider a hypothetical test case, with many dependencies which works with small amount of different target tables. Concretely a test case where every source-to-target dependency includes all target tables. This constellation may not be a typical picture of the data warehouse scenario, but could be used as scenario, where we can expect better results than in the previous test case.

We expect, that the complexity of our algorithm increases slightly, but the complexity (in the meaning of "join" and "table scan" operations) of the traditional algorithm increases significantly. The reason is the necessity of join of all tables together by every validation check.

In this example, we make several executions with different amount of records, in order to see the correlation between the data amount and execution time.

7.2.7 Test Case Definition

We decided to take the following kind of dependencies:

On the source side, we have exactly one table (*Source_1*). The rationale behind this, is the fact that the amount of tables on the source site is not the differentiating factor. Both algorithms have to make the same join operations on the source side, so we use only one table.
On the target side, we took 5 target tables (*Target_1*, ... , *Target_5*) which have to be loaded with records from source side. The target structure has the snowflake form.
Next, we have source-to-target dependencies where all 5 target tables are involved. We decided to use this kind of dependency 5 times, whereby there are always different target columns which are loaded.

Dep.7: *Source_1*(ID, FK_KEY) $\rightarrow$ ($\exists$ T2, T3, T4, T5) *Target_1*(S1) $\wedge$ *Target_2*(S2) $\wedge$ *Target_3*(S3) $\wedge$ *Target_4*(S4) $\wedge$ *Target_5*(S5)

We did the tests with 3 different counts of data, concretely: 1.000 records, 10.000 records, 100.000 records. Every execution has been made 3 times and the average execution time has been calculated.

7.2.8 Execution Results

The tables below show the execution results for the new algorithm:

The execution results are in the Tables 7.4 resp. 7.5

Table 7.4: Execution with 1.000 records

Validation package	Execution time in sec.
DQ_META_BUILD_INFRASTRUCTURE	0.105
DQ_META_SOURCE_TARGET_DEP	0.500
DQ_META_TARGET_DEP	0.001
DQ_META_BUILD_MASTER_TABLE	0.040
DQ_META_REMOVE_REDUNDANT_DATA	4.435
DQ_META_VALIDATE_TARGET	0.061
Total execution	**5.142**

Table 7.5: Execution with 10.000 records

Validation package	Execution time in sec.
DQ_META_BUILD_INFRASTRUCTURE	0.301
DQ_META_SOURCE_TARGET_DEP	3.565
DQ_META_TARGET_DEP	0.001
DQ_META_BUILD_MASTER_TABLE	0.156
DQ_META_REMOVE_REDUNDANT_DATA	165.500
DQ_META_VALIDATE_TARGET	0.354
Total execution	**169.877**

The execution times are significantly higher than we expected. With the bigger amount of data, the new algorithm appears to be slow. From the first look, its performance is highly dependent on the amount of records and its increase appears to be linear. The deeper analysis showed that the implementation of the step which looks for redundant records is the root cause of the slow performance. It looks for the same records between the other records, but very inefficiently, because it uses a separate query for every search. This is a potential improvement place, to replace the logic with a database "DISTINCT" operation. It requires an implementation of a mechanism which generates "DISTINCT" queries for every dependency in comparison to generate a query for every one record. After the implementation of this improvement, we gathered better results.

The repeated execution results are in the Tables 7.6 resp. 7.7 resp. 7.8

Table 7.6: Repeatedly execution with 1.000 records

Validation package	Execution time in sec.
DQ_META_BUILD_INFRASTRUCTURE	0.105
DQ_META_SOURCE_TARGET_DEP	0.500
DQ_META_TARGET_DEP	0.001
DQ_META_BUILD_MASTER_TABLE	0.040
DQ_META_REMOVE_REDUNDANT_DATA	0.137
DQ_META_VALIDATE_TARGET	0.061
Total execution	**0.844**

Table 7.7: Repeatedly execution with 10.000 records

Validation package	Execution time in sec.
DQ_META_BUILD_INFRASTRUCTURE	0.301
DQ_META_SOURCE_TARGET_DEP	3.565
DQ_META_TARGET_DEP	0.001
DQ_META_BUILD_MASTER_TABLE	0.156
DQ_META_REMOVE_REDUNDANT_DATA	1.452
DQ_META_VALIDATE_TARGET	0.354
Total execution	**5.829**

Table 7.8: Repeatedly execution with 100.000 records

Validation package	Execution time in sec.
DQ_META_BUILD_INFRASTRUCTURE	0.240
DQ_META_SOURCE_TARGET_DEP	72.630
DQ_META_TARGET_DEP	0.014
DQ_META_BUILD_MASTER_TABLE	3.825
DQ_META_REMOVE_REDUNDANT_DATA	5.452
DQ_META_VALIDATE_TARGET	24.104
Total execution	**106.265**

For the execution of the traditional algorithm we took the dependencies and rewrote it manually into SQL statements. The statements have the form, in which all tables have been joined together, including the WHERE conditions to validate the data.

An example of such statement could be:

```
Select          count( * )
from            s_1
                join t_1 on (s_1.s1_1 = t_1.t1_1)
                join t_2 on (t_1.t1_1 = t_2.fk_key)
                join t_3 on (t_1.t1_1 = t_3.fk_key)
                join t_4 on (t_2.t2_1 = t_4.fk_key)
                join t_5 on (t_3.t3_1 = t_5.fk_key)
where            t_3.t3_3 = s_1.s1_3
```

The execution results of the traditional algorithm are shown in the table 7.9

Table 7.9: Execution of traditional algorithm

Number of records	Total execution time in sec.
1.000	**0.360**
10.000	**0.850**
100.000	**1.870**

7.2.9 Test Evaluation

Our tests confirmed that the traditional algorithm is better performing than the new one. Surprisingly, we did not find the performance of the new algorithm as slow as expected. What we did not expect, is the high performance of the traditional algorithm. There are many factors, which could cause the unusually good performance. We think that the Oracle database optimization techniques played a big role during the execution. As an example, we used a data generator with almost identical records, we used simple join conditions and almost identical validation queries. The Oracle database appeared to generate rich table statistics and deliver the results with the help of them. While using this, for some cases it is not necessary to make a full table scan; resp. it can speed up the join operations.

7.2.10 Summary

The execution showed, that for this particular test, the new algorithm does not outperform the traditional one. However, we see the overall results as promising. Even if the execution times in our test case did not achieve the performance of the traditional algorithm, it showed that it is a good candidate for further examination. The detailed analysis will surely reveal a space for next implementation improvements; resp. executions with different data settings can also show better execution results.

As a second test result we proved the possibility of using the algorithm for data validation, whereby the algorithm always generated the data which we expected.

CHAPTER 8

Conclusion

In this thesis we presented a possible approach how to use the knowledge from data exchange in the data quality area. We confirmed that the principles and several algorithms from data exchange could be used to validate data in the data quality area. The core, which is the main product in data exchange, represents also the ultimate set of data which satisfies the dependencies. In data quality it represents the set of valid data which can serve as an etalon by validating existing data.

We put a modified version of the core computing algorithm into the role of data quality and on its base we prepared an algorithm which can compute the set of valid data for the target with snowflake structure. This algorithm supports source-to-target and target TGDs. Its applicability has some limitations (e.g. no weakly acyclic dependencies, "and" conditions only), but in general shows a feasible way how to algorithmically compute the valid data set for selected types of data quality checks. Furthermore, it can serve as a base for later examinations how to extend them to cover more data quality check types.

The practical implementation of the algorithm showed a way how to automate the computation of a minimal valid data set. Afterwards practical complexity evaluation did not confirm our optimistic expectations that the algorithm can outperform the standard data quality algorithm. With the aim to limit the disk accessing operations we tried to minimize the count of table scan / table join operations to avoid multiple reading of the same data rows. The elimination of such data rows in our tested scenarios was the most complicated implementation challenge. The final implemented algorithm creates one big master table and processes further operations directly over this structure. This allows to start complete data quality assessment for the current accessed data rows within one database statement. Consequently, we had to dynamically build such database statements to put all validation logic into them. Despite the performance improvements of this approach, the necessity of accessing the data multiple times still remained.

If we summarize our findings into one core message, it might sound: The algorithms from data exchange could be successfully used with several restrictions for data validation. Our practice implementation with several scenarios shows an automated way how to do it but did not show performance benefits against traditional data validation algorithms.

8.1 Future Work

Towards a practical usage it would be interesting to implement the algorithm with the ETL tool and see whether a strategy consisting of a combination of more processing steps into one logic unit with no need to repeatedly access the same table data, can be applied. This can significantly reduce the time needed for reading disk data.

Furthermore, if we consider the implementation with the ETL we realize another interesting fact. We get the complete set of valid data based on the formal description of dependencies. With some modifications, we can use this concept also for the standard data load. For this purpose we need only a small extension of the program and few more supporting functions. Under these circumstances we will have a robust framework which is used to load the data and also could be called on demand to validate this data.

Bibliography

[1] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, July 2009.

[2] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, September 1984.

[4] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 61–75, New York, NY, USA, 2005. ACM.

[5] Larry P. English. *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[6] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: Getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, March 2005.

[7] Phokion G. Kolaitis, Jonathan Panttaja, and Wang-Chiew Tan. The complexity of data exchange. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 30–39, New York, NY, USA, 2006. ACM.

[8] Aleksander Mądry. Data exchange: On the complexity of answering queries with inequalities. *Inf. Process. Lett.*, 94(6):253–257, June 2005.

[9] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[10] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, March 1997.

[11] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[12] Thomas Jorg and Stefan Dessloch. Towards generating ETL processes for incremental loading. In *Proceedings of the 2008 International Symposium on Database Engineering & Applications*, IDEAS '08, pages 101–110, New York, NY, USA, 2008. ACM.

[13] Panos Vassiliadis, Alkis Simitsis, and Eftychia Baikousi. A taxonomy of etl activities. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, DOLAP '09, pages 25–32, New York, NY, USA, 2009. ACM.

[14] Thomas C. Redman. *Data Quality for the Information Age*. Artech House, Inc., Norwood, MA, USA, 1st edition, 1997.

[15] InduShobha N. Chengalur-Smith, Donald P. Ballou, and Harold L. Pazer. The impact of data quality information on decision making: An exploratory analysis. *IEEE Trans. on Knowl. and Data Eng.*, 11(6):853–864, November 1999.

[16] Yang W. Lee and Diane M. Strong. Knowing-why about data processes and data quality. *J. Manage. Inf. Syst.*, 20(3):13–39, December 2003.

[17] David Garvin. What does product quality really mean? *Sloan Management Review*, 26:25–45, 1984.

[18] Thomas C. Redman. *Data quality : the field guide*. Digital Pr. [u.a.], Boston, 2001.

[19] Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. Data quality assessment. *Commun. ACM*, 45(4):211–218, April 2002.

[20] Informatica Inc. Data quality process. Informatica Corp. Figure published in 2009.

[21] Daniela Florescuand. An extensible framework for data cleaning. In *Proceedings of the 16th International Conference on Data Engineering*, ICDE '00, pages 312–, Washington, DC, USA, 2000. IEEE Computer Society.

[22] Stuart E. Madnick, Richard Y. Wang, Yang W. Lee, and Hongwei Zhu. Overview and framework for data and information quality research. *J. Data and Information Quality*, 1(1):2:1–2:22, June 2009.

[23] David Loshin. Rule-based data quality. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 614–616, New York, NY, USA, 2002. ACM.

[24] Richard Y. Wang and Diane M. Strong. Beyond accuracy: What data quality means to data consumers. *J. Manage. Inf. Syst.*, 12(4):5–33, March 1996.

[25] Jasna Rodic and Mirta Baranovic. Generating data quality rules and integration into etl process. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, DOLAP '09, pages 65–72, New York, NY, USA, 2009. ACM.

[26] Gartner Inc. Gartner magic quadrant for data quality. Report published in 2010.

[27] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, May 2005.

[28] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. Express: A data extraction, processing, and restructuring system. *ACM Trans. Database Syst.*, 2(2):134–174, June 1977.

[29] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: The teenage years. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 9–16. VLDB Endowment, 2006.

[30] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 233–246, New York, NY, USA, 2002. ACM.

[31] Umeshwar Dayal, Malu Castellanos, Alkis Simitsis, and Kevin Wilkinson. Data integration flows for business intelligence. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 1–11, New York, NY, USA, 2009. ACM.

[32] Balder Ten Cate, Laura Chiticariu, Phokion Kolaitis, and Wang-Chiew Tan. Laconic schema mappings: Computing the core with SQL queries. *Proc. VLDB Endow.*, 2(1):1006–1017, August 2009.

[33] Laura Chiticariu, Phokion G. Kolaitis, and Lucian Popa. Interactive generation of integrated schemas. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 833–846, New York, NY, USA, 2008. ACM.

[34] Phokion G. Kolaitis. *A Tutorial on Schema Mappings & Data Exchange*. University of California Santa Cruz, IBM Research Center Almaden, 2010.

[35] Ariel Fuxman, Mauricio A. Hernandez, Howard Ho, Renee J. Miller, Paolo Papotti, and Lucian Popa. Nested mappings: Schema mapping reloaded. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 67–78. VLDB Endowment, 2006.

[36] Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2):9:1–9:49, May 2008.

[37] Reinhard Pichler and Vadim Savenkov. Towards practical feasibility of core computation in data exchange. *Theor. Comput. Sci.*, 411(7-9):935–957, 2010.

[38] Giansalvatore Mecca, Paolo Papotti, and Salvatore Raunich. Core schema mappings. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 655–668, New York, NY, USA, 2009. ACM.

[39] Alan Nash, Philip A. Bernstein, and Sergey Melnik. Composition of mappings given by embedded dependencies. *ACM Trans. Database Syst.*, 32(1), March 2007.

[40] Georg Gottlob and Alan Nash. Data exchange: Computing cores in polynomial time. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 40–49, New York, NY, USA, 2006. ACM.