Managing Evolution of Heterogeneous Data Sources of a Data Warehouse

Darja Solodovnikova[a], Laila Niedrite[b] and Lauma Svilpe
Faculty of Computing, University of Latvia, Raina blvd. 19, Riga, Latvia

Keywords: Data Warehouse, Evolution, Change Propagation, Metadata, Big Data.

Abstract: The evolution of heterogeneous integrated data sources has become a topical issue as data nowadays is very diverse and dynamic. For this reason, novel methods are necessary to collect, store and analyze data from various data sources as efficiently as possible, while also handling changes in data structures that have occurred as a result of evolution. In this paper, we propose a solution to problems caused by evolution of integrated data sources and information requirements for the data analysis system. Our solution incorporates an architecture that allows to perform OLAP operations and other kinds of analysis on integrated big data and a mechanism for detecting and automatically or semi-automatically propagating changes occured in heterogeneous data sources to a data warehouse.

1 INTRODUCTION

Over the last few years, due to the increase in volume of data stored in data warehouses and emergence of various new data sources, such as unstructured and semi-structured data types, new challanges related to data storage, maintenance and usage arise. Therefore, novel efficient algorithms for the integration and analysis of heterogeneous data are necessary for the development of data warehouses. Besides, despite the fact that in most cases data sources are already integrated into the data warehouse system, there is a need for automatized solutions that are able to deal with data variability and evolution problems.

Traditional methods applied in relational databases may not be laveraged for new types of data, therefore, tools, technologies, and frameworks have been introduced to support large-scale data analytics, such as Apache Hadoop file system, Apache HBase database management solution, Hive data warehouse solution and others. These tools are mainly aimed at managing the data growth, leaving the problems caused by evolution of data and their structure unresolved. Hence, handling various types of changes in data structure still requires a large amount of manual work on the part of the developer as existing solutions do not support automatic or semi-automatic propagation of changes in data sources to a data warehouse.

[a] https://orcid.org/0000-0002-5585-2118
[b] https://orcid.org/0000-0002-8173-6081

To address the evolution problem, we propose a solution to handle changes caused by the evolution of heterogeneous integrated data sources. Our solution includes a data warehouse architecture that on one hand supports various types of data analysis of data integrated in a data warehouse and on the other hand is able to discover changes in structured and semi-structured data sources and automatically or semi-automatically propagate them in the system to maintain continuous system operation.

The rest of this paper is organized as follows. In Section 2 the recent studies related to the topic are discussed. In Section 3 we outline the proposed data warehouse architecture. Section 4 is dedicated to the description of the case study system and the running example we use throughout the paper to illustrate our solution. We overview the metadata employed in our solution to handle evolution in Section 5. The main contribution of this paper is presented in Section 6, where we discuss the adaptation scenarios for changes in data sources and present the mechanism for change handling. Finally, we conclude with directions for future work in Section 7.

2 RELATED WORK

The problem of data warehouse evolution has been studied extensively in relational database environments. There are in general two approaches to solving evolution problems. One approach is to adapt just

Solodovnikova, D., Niedrite, L. and Svilpe, L.
Managing Evolution of Heterogeneous Data Sources of a Data Warehouse.
DOI: 10.5220/0010496601050117
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1*, pages 105-117
ISBN: 978-989-758-509-8

the existing data warehouse schema (Bentayeb et al., 2008) or ETL processes (Wojciechowski, 2018) without keeping the history of changes and another approach (Ahmed et al., 2014), (Golfarelli et al., 2006), (Malinowski and Zimányi, 2008) is to maintain multiple versions of schema that are valid during some period of time.

The topicality of evolution problems in big data environments is discussed in the several recent review papers. The authors in (Kaisler et al., 2013) mention dynamic design challenges for big data applications, which include data expansion that occurs when data becomes more detailed. A review paper (Cuzzocrea et al., 2013) indicates research directions in the field of data warehousing and OLAP. Among others, the authors mention the problem of designing OLAP cubes according to user requirements. Another recent vision paper (Holubová et al., 2019) discusses the variety of big data stored in the multi-model polystore architectures and suggests that efficient management of schema evolution and propagation of schema changes to affected parts of the system is a complex task and one of the topical issues.

We have also found several studies that deal with evolution problems in big data context. A solution to handling data source evolution in the integration

field was presented in the paper (Nadal et al., 2019). The authors propose the big data integration ontology for the de

nition of integrated schema, source schemata, their versions and local-as-view mappings between them. When a change at a data source occurs, the ontology is supplemented with a new release that reflects the change. Our approach differs in that the proposed architecture is OLAP-oriented and is capable of handling not only changes in data sources, but also requirements.

Another study that considers evolution is presented in the paper (Chen, 2010). The author proposes a data warehouse solution for big data analysis that is implemented using MapReduce paradigm. The system supports two kinds of changes: slowly changing dimensions are managed with methods proposed in (Kimball and Ross, 2019) and fact table changes are handled by schema versions in metadata. Unlike our proposal, the system does not process changes in big data sources.

An architecture that exploits big data technologies for large-scale OLAP analytics is presented in the paper (Sumbaly et al., 2013). The architecture supports source data evolution by means of maintaining a schema registry and enforcing the schema to remain the same or compatible with the desired structure.

There is also the latest study presented in (Wang et al., 2020) dedicated to evolution problems in heterogeneous integrated data sources. The authors propose to use deep learning to automatically deal with schema changes in such sources.

For our solution, we adapted the metadata model proposed in (Quix et al., 2016) to describe data sources of a data lake. The authors distinguish three types of metadata: structure metadata that describe schemata of data sources, metadata properties and semantic metadata that contain annotations of source elements. In our approach, we extended the model with metadata necessary for evolution support.

3 DATA WAREHOUSE ARCHITECTURE

To solve problems caused by the evolution, we propose a data warehouse architecture for the analysis of big data, which supports OLAP operations and other types of analysis on integrated data sources, as well as includes algorithms for detecting and handling various changes in structured, semi-structured, and unstructured data sources of a data warehouse and information requirements. The detailed description of the architecture is given in the paper (Solodovnikova and Niedrite, 2018). The architecture consists of various components that provide data flow and processing from the source level to the data stored in the data warehouse. The interaction of these components is shown in the Figure 1.

The basic components of the data warehouse architecture are data sources, data highway, metastore and adaptation component. At the source level, data is obtained from various heterogeneous sources (including big data sources) and loaded into the first level of the data highway (data processing pipeline which may be considered as a data lake) in its original format for further processing. Because big data includes different types of data, the system supports structured sources (database tables), semi-structured sources (such as XML, JSON or CSV format), and unstructured data (log files, photos, videos).

The data highway consists of several levels. The idea and the concept of the data highway was first presented in (Kimball and Ross, 2019). At the first level, raw data is stored. The data for each subsequent data highway level is obtained from the previous level by performing transformations, aggregations and integrating separate data sets. Usually, data at the later levels is updated less frequently. The number of levels, their contents and the frequency of their updating are determined by the requirements of the particular system. The final level of the highway is

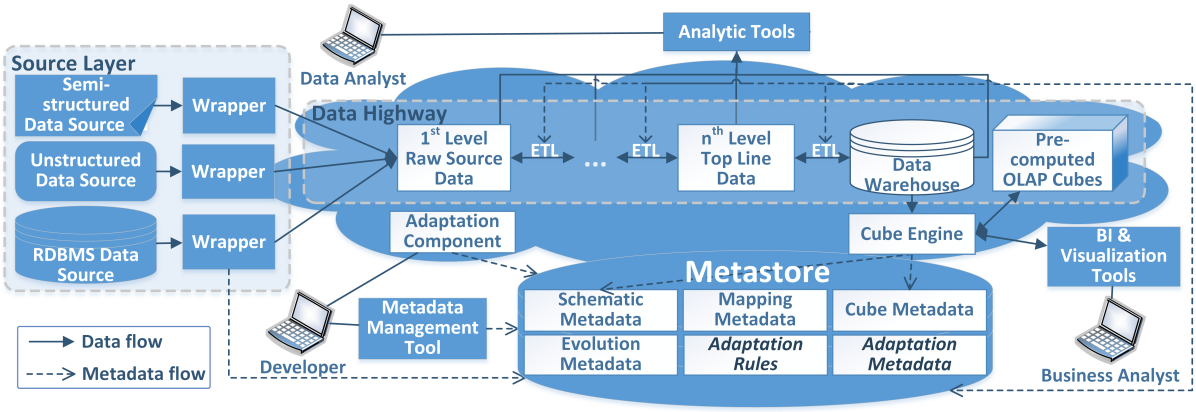

Figure 1: Data warehouse architecture.

a data warehouse which stores structured aggregated multidimensional data.

Business analysts can access already processed data from pre-calculated OLAP cubes introduced to improve query performance, but data analysts can analyze data at various levels of the data highway and are involved in the data retrieval and transformation process. Developers access metadata at the metastore via the metadata management tool.

The unique feature of the architecture is the adaptation component that is aimed at handling changes in data sources or other levels of the data highway. The metadata management tool is intended for users, but the adaptation component is integrated with that tool.

4 CASE STUDY

As a proof of concept, we have applied the solution presented in this paper to the data warehouse that integrates data on research publications authored by the faculty of our university. In this section, we will briefly describe the case study system and present the running example of the change.

4.1 Publication Data Warehouse

Data about publications are integrated from four data sources. Structured data about faculty (authors of articles) are acquired from the university information system. Semi-structured data are gathered in XML and JSON formats from the library information system and indexing databases Scopus and Web of Science. The system architecture consists of three levels of the data highway, the last of them being a data warehouse storing data from all four sources fully integrated within the first and second levels of the highway.

During the operation of the publication data warehouse, several changes in data sources and data highway levels were detected by the adaptation component or introduced manually via the metadata management tool. Such real-world changes include an addition of new data items and removal of existing data items in data sources, addition of a new data source and change in a value of a data set property. We have also emulated other types of changes described in Section 6 in order to practically verify our solution.

4.2 Running Example

In order to demonstrate our approach in the next sections of this paper, we will use a running example of a change that occured in the publication data warehouse. A new XML element *citeScoreYearInfoList* was added to the XML document *Scopus metrics* obtained from SCOPUS. It was composed of several subelements that were also absent in the previously gathered documents. Before the change, data about other Scopus metrics were used in the data warehouse to evaluate publications. The new metric defined by SCOPUS should have been considered too in this evaluation.

5 METADATA

The operation of the data warehouse architecture is mainly based on the data in the metadata repository. Using a metadata management tool and defining different metadata, the developer determines how the system will work. The metadata repository stores six types of interconnected metadata: Schematic metadata describe schemata of data sets stored at different levels of the highway. Mapping metadata define the logic of ETL processes. Information about changes in

data sources and data highway levels is accumulated in the evolution metadata. Cube metadata describe schemata of precomputed cubes. Adaptation metadata accumulate proposed changes in the data warehouse schema. Finally, adaptation rules store additional information provided by the developer required for change propagation.

5.1 Schematic, Mapping and Evolution Metadata

To describe schemata of data sources and data highway levels necessary for the analysis along with changes in structure and other properties of involved data sets we developed the metadata shown in Figure 2. In this section, we briefly describe the physical implementation of the schematic, mapping and evolution metadata. The detailed description these types of metadata is given in the paper (Solodovnikova et al., 2019).

5.1.1 Schematic Metadata and Mappings

The table *DataSet* represents a collection of *DataItems* that are individual pieces of data. Data sets are assigned *FormatTypes* that are stored in the table *Type* and grouped into parent types. Parent types and corresponding format types currently supported in the system are defined in the Table 1. Data items are also assigned types, such as table columns, XML elements or attributes, objects or arrays in JSON, keywords or tags that describe unstructured data sets, and others.

Table 1: Format types of data sets.

Parent Type	Format Types
Structured data set	Table
Semi-structured data set	XML, JSON, CSV, RDF, HTML, Key-Value
Unstructured data set	Text, image, other multimedia

A data set can be obtained from a *DataSource* or it can be part of a *DataHighwayLevel*. In addition to data set formats, information on the *Frequency* and *Velocity* of a data set retrieval, being batch, near real-time, real-time and stream is also stored. If a data set is a part of a data warehouse, a *Role* in the multidimensional model is assigned to such data set (dimension or fact) and corresponding data items (attribute or measure).

If there is a link between different data items in the same or across different data sets, the tables *Relationship* and *RelationshipElement* are used to connect the respective child and parent objects. The types of relationships supported include composition, foreign key and predicate.

Data sets in the system are either extracted from data sources or obtained from other levels of the data highway, thus the information about provenance of data sets within the data highway must be maintained to make it possible to follow their lineage and implement change handling process. For this purpose, mappings were introduced in the metadata. A record in the table *Mapping* indicates a transformation (saved in the column *Operation*) that is used to obtain a data item at the next data highway level from data items at the previous levels associated via the table *MappingOrigin*.

5.1.2 Metadata Properties

Even though several common properties of data sets and items, such as types, velocity, frequency, are explicitly reflected in our metadata model, there still might be other characteristics of data that are necessary to be stored. Examples of such properties include file names, sizes, character sets, check constraints, data types, mechanism used to retrieve data from a data source (for instance, API request), etc. To this end, we included the table *MetadataProperty*. Any name:value pair relevant to schema elements present in the model can be added to the metadata. Considering that various elements may possess different properties, such approach allows for some flexibility. Furthermore, it allows to store user-defined properties or so-called conversational metadata associated with the *Author* who recorded the property.

5.1.3 Evolution Metadata

Thus far, we have discussed the schematic and mapping metadata, however, the infomation on evolution of data sets must also be maintained in the metastore. Hence, we identified a set of change types that are discussed in more detail in the section 6.1, as well as we included the table *Change* intended for recording changes that have been automatically discovered or introduced manually. Every record of this table stores a date and time of the change, its *Type*, *Status* (new, propagated or in process of propagation) and is associated with a schema element affected by the change. The columns *AttrName*, *OldAttrValue* and *NewAttrValue* are filled with a name, previous and updated values of a property if a value has been changed. If the change was produced manually, we associate it with the corresponding *Author*.

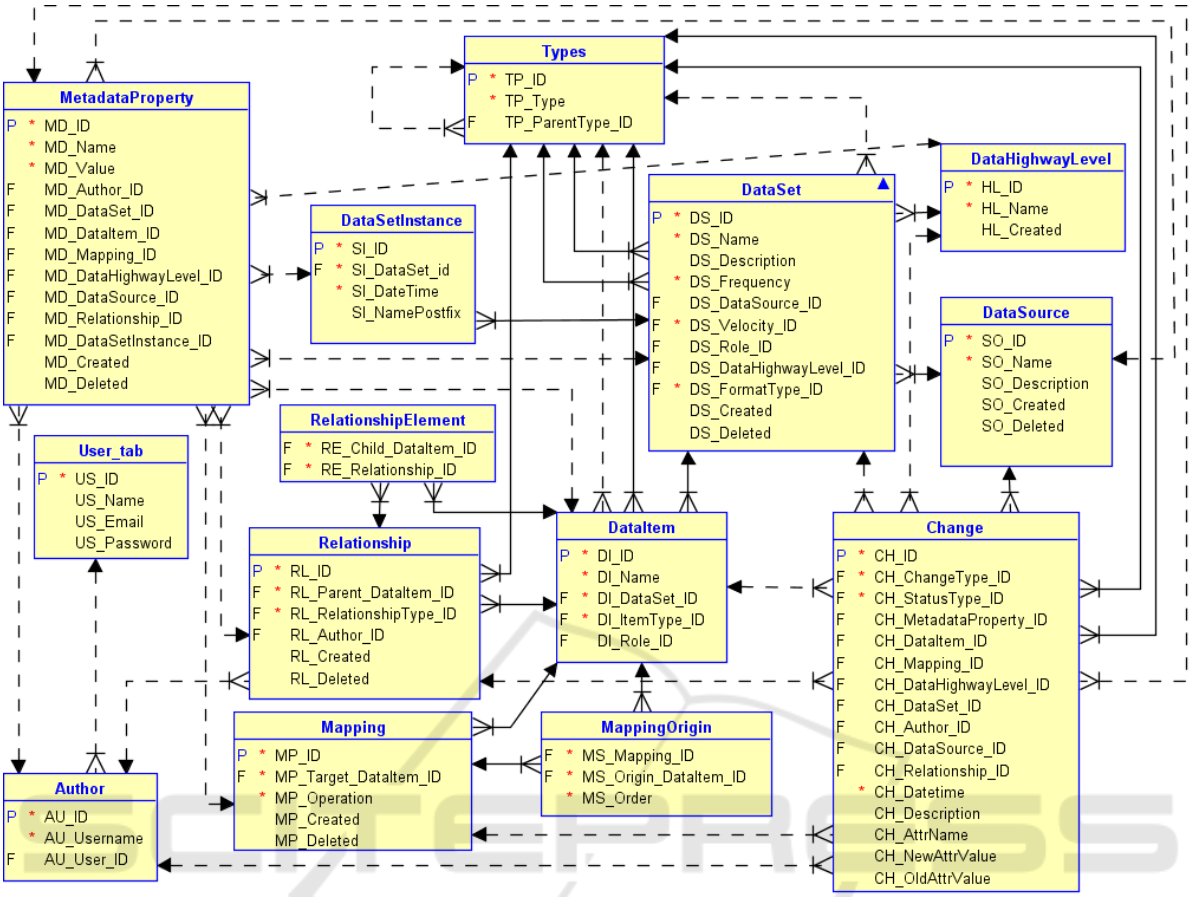

Figure 2: Schematic and evolution metadata model.

5.1.4 Running Example

The schematic metadata of the publication data warehouse were gathered automatically and supplemented by additional information manually. The information about Scopus metrics and the addition of a new element as indicated in Section 4.2 was represented in metadata as shown in the table 2. The table shows an example of a relationship between the parent element citeScoreYearInfoList and the child element citeScoreCurrentMetric in XML document. The new element citeScoreYearInfoList included a more complicated hierarchy of child elements that are not diplayed in the table due to space limitations.

Table 2: Metadata describing the addition of a new element.

Table	Data	References
DataSource	SCOPUS	
DataSet	Scopus metrics	DataSource: SCOPUS
DataItem	citeScore-YearInfoList	DataSet: Scopus metrics
DataItem	citeScore-CurrentMetric	DataSet: Scopus metrics
Relationship	Type: Composition	Parent: citeScore-YearInfoList
Relationship-Element		Child: citeScore-CurrentMetric
Change	Type: Addition	DataItem: citeScoreYear-InfoList

5.2 Adaptation Metadata

After changes have been recorded in the metadata by the change discovery algorithm, the change handling mechanism of the adaptation component determines possible scenarios for each change propagation. The information about possible scenarios for each change type is stored in the adaptation metadata. Scenarios that may be applied to handle changes are provided to the data warehouse developer who chooses the most appropriate ones that are to be implemented. Since certain scenarios may require additional data from the developer, such data are entered by the developer via

the metadata management tool and stored in the adaptation rules metadata after the respective scenario has been chosen.

The Figure 3 demonstrates the adaptation metadata model utilized in the change handling mechanism. The model incorporates three tables from the evolution and schema metadata (Figure 2): *Change*, *Types* and *Author*. The table *Change* stores information about identified changes. It is possible to determine which type of change has occurred by processing entries in this table. The overall change handling process is based on the records in this table.

The table *Author* stores information about system users. This table is used to identify which user performed adaptation after a certain change in the system. The table *Type* is used as a classifier, which determines the types of various elements used in the system. To implement the evolution mechanism, several types and subtypes have been created for storing different statuses and data types in the adaptation metadata.

5.2.1 Change Adaptation Scenarios and Operations

The tables *ChangeAdaptationOperation* and *ChangeAdaptationScenario* are intended for storing information about change adaptation scenarios and operations to be performed to implement each scenario. These tables are filled with data manually before any changes occur.

Change adaptation operations are steps that must be taken to handle a change in the system. They are defined as short and universal as possible. Each of the operations is stored as a record in the table *ChangeAdaptationOperation*. An operation is assigned a type that indicates whether it can be performed manually or automatically. In the former case, the column *Operation* stores a textual description of what the developer must do to perform the operation. In the latter case, the column *Operation* stores the name of the procedure to be executed.

A change adaptation scenario is a series of sequential operations that are performed to successfully handle a change in the system. Multiple adaptation scenarios correspond to each change type. The steps of each scenario are stored in the table *ChangeAdaptationScenario*. Each step of a scenario contains a reference to the adaptation operation, a change type, as well as a parent record identifier from the same table. Storing the parent record identifier maintains the sequence of operations. First steps of each scenario do not have parent records. Each next step contains a reference to the previous step. Such structure facilitates adjustments. For example, adding an operation in the middle of a scenario requires only two table records to be edited.

Several scenarios may overlap in terms of their operations, therefore operations are stored in the separate table to avoid duplication of information in the metadata.

In order to follow the change handling process and store information about the execution of each operation, we introduced the table *ChangeAdaptationProcess* which stores the adaptation scenario corresponding to each actual change. This table is populated automatically during the change handling process. When a change has taken place, the adaptation component inserts potential alternative adaptation scenarios into the table *ChangeAdaptationProcess* for that change. Each record in the table *ChangeAdaptationProcess* stores the link to the operation of the adaptation scenario (defined in the table *ChangeAdaptationScenario*). We also store a reference to a record of the table *Change* to identify which of the changes is being handled. In order to keep track of the change handling process, the status of the operation to be performed (whether the operation has been performed or not) is also stored, as well as the date, time and the user who executed or launched the operation.

5.2.2 Branching Conditions of Change Adaptation Scenarios

To store the branching conditions of adaptation scenarios and to manage the fulfillment of the conditions, the tables *ChangeAdaptationCondition* and *CaConditionMapping* have been introduced. Although the type of change can be determined at the time of its occurrence, it does not guarantee the existence of an unambiguous change adaptation scenario. There are various conditions under which a scenario can branch out. They can be evaluated automatically or manually. In the former case, the adaptation component determines which of the scenario branches to follow. In the latter case, the developer needs to evaluate the specific situation and make a decision.

Just like operations, the conditions for handling different types of changes overlap, so they are stored in a separate table *ChangeAdaptationCondition*. Each record of this table stores the type of the condition (whether the condition is evaluated manually or automatically) and the condition definition. If the condition is executable manually, a textual description that must be evaluated by the developer is stored in the column *Condition*. If the condition is automatically evaluable, the name of the function to be executed is stored there. The conditions are manually described for each change type in advance and are entered in the table before any changes occur.

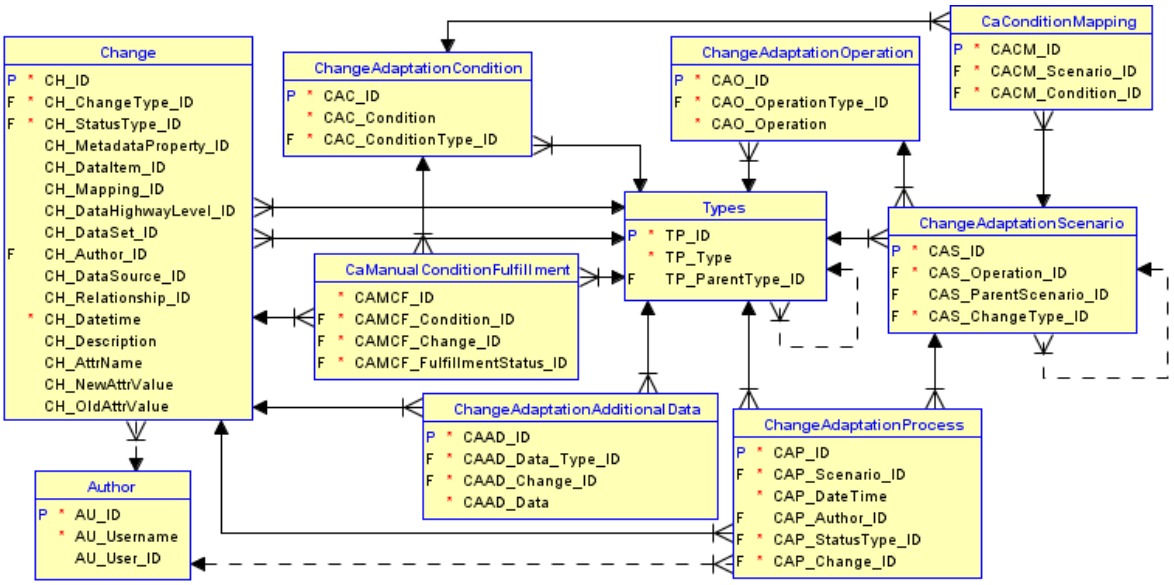

Figure 3: Adaptation metadata model.

Because each operation that is executed within a scenario can have multiple conditions, the table *CaConditionMapping* is used to provide the N:N relationship between the tables *ChangeAdaptationCondition* and *ChangeAdaptationScenario*.

Conditions that can be evaluated automatically can be checked before each operation since they do not require the intervention of the developer. Thus, it is possible to automatically move along one of the branches defined in the scenario. However, when performing a manual condition evaluation (the developer decides on the future course of the scenario), it is necessary to keep information about the decision the developer has made in order for this decision to be taken into account throughout the whole change handling process.

The table *CaManualConditionFulfillment* is used to store decisions of the developer. Records in this table are linked to the particular change and condition as well as determine the condition fulfillment status (whether the condition is fulfilled or not). The table *CaManualConditionFulfillment* is filled in automatically during the specific change handilng process.

5.2.3 Additional Information for the Change Handling Process

During the change handling process, various additional data may be required to be provided by the developer. Such data might be needed to perform operations included in the scenario as well as to evaluate conditions.

If any additional information from the developer is required during the change handling, it is saved in the table *ChangeAdaptationAdditionalData*, which stores the reference to the table Change, the information on the purpose for which the additional data is used, as well as the data itself. The format of the data depends on the type of data. For example, storing only the identifier will be a simple number. However, there might be situations when data has another format (JSON, XML, CSV, etc.), therefore, a textual data type with a large capacity is used for the column *Data*.

6 EVOLUTION SUPPORT

The main element of the proposed architecture that is responsible for handling changes in data sources and information requirements is the adaptation component. Its main task is to detect changes that have taken place and to generate change adaptation scenarios for each change. To ensure the desired functionality, the adaptation component uses data in the metadata repository, as well as additional data that cannot be automatically identified, in which case the developer provides this data manually using the metadata management tool. When the developer has chosen the specific scenarios to execute and provided all necessary data or scenacio choice has been made automatically based on the evaluation of automatic conditions, the adaptation component manages the execution of scenarios. Since data from data sources are loaded into the data highway initially in their original format, the data acquisition may continue even during

the change handling process. In this section, we concentrate on changes supported in our proposed solution as well as their propagation mechanism.

6.1 Atomic Changes

Various kinds of changes to data sets employed in each level of the data highway must be handled by the adaptation component. The list of atomic changes supported in our proposed solution classified according to the part of the metadata model they affect follows:

- *Schematic Changes*: addition of a data source, deletion of a data source, addition of a data highway level, deletion of a data highway level, addition of a data set, deletion of a data set, change of data set format, renaming a data set, addition of a data item, change of a data item type, renaming a data item, deletion of a data item from a data set, addition of a relationship, deletion of a relationship, addition of a mapping, deletion of a mapping;
- *Changes in Metadata Properties*: addition of a metadata property, deletion of a metadata property, update of an attribute value.

Most of these changes may be identified automatically by the change discovery algorithm implemented as a part of the adaptation component and described in detail in the paper (Solodovnikova and Niedrite, 2020). Manually introduced changes are processed by the metadata management tool, however, automatic change discovery is triggered when any new data are loaded from data sources into the data highway by wrappers or during ETL processes.

Initially, the change detection algorithm gathers schema metadata and properties of existing data sources and data highway levels in temporary metadata. For metadata collection, special procedures are used depending on the format of data sets. Structured, semi-structured and unstructured data sets are handled by different procedures. To identify changes in metadata, a change discovery algorithm first processes data sources and data highway levels, then data sets and data items, and finally mappings and relationships. For each processed element, the algorithm compares metadata describing the element available in the metastore with collected temporary metadata and identifies differences that determine types of atomic changes occured. The identified changes are then saved in the table *Change*.

6.2 Change Adaptation Scenarios

After any changes have been detected, the adaptation component of our proposed architecture must first generate potential adaptation scenarios for each change and then execute scenarios according to branching conditions. We have predefined adaptation scenarios for each change type and operations necessary for each scenario. In total, 34 different change adaptation scenarios were defined for 19 atomic changes. The average proportion of automatic operations and conditions within each change adaptation scenario is almost 47%. In the following subsections, scenarios for each atomic change are described.

6.2.1 Addition of a Data Source

If a new data source is required for decision making, examples of data sets from the new source must be added manually so that the metadata collection procedures can generate the necessary metadata for the new source structure. The adaptation component must then create the data structures according to the data sets of the new source at the first level of the data highway (see the change 6.2.5). The developer must then define schemas of other new data highway levels, ETL processes, and create the corresponding metadata.

6.2.2 Deletion of a Data Source

If a data source from which data was previously downloaded is no longer available, then three manual adaptation scenarios are possible:

- *Replacement of a Missing Data Source with Data from Other Sources*. The objective of this scenario is to make it possible to continue data loading to data items that depend on the deleted data source. To implement the replacement of the deleted data source, the developer must provide information on alternative data sources. For each data item from the missing source, an alternative data item from another source must be specified or a formula that calculates data items of the missing source from data items of other sources must be provided. For this adaptation scenario, it may be necessary to add metadata describing the data structure and properties of the new data source (see the change 6.2.1).
- *Data Source Skipping*. If replacement of a missing data source is not possible, data items that depend on the missing data source can not be further updated. The adaptation scenario in such a case includes the modification of ETL processes (not

filling dependent data items) and modification of the mapping metadata.

- *Hybrid Scenario*. If it is possible to replace part of data items that were obtained from the missing source, for such data items new ETL processes, data structure of new sources (if needed) and other properties must be defined. Other data items that can not be obtained any more are left blank.

6.2.3 Addition of a Data Highway Level

If the developer has added a new data highway level, for this change the adaptation scenario consists of describing the data structure of the new level in the metadata, creating the data structure according to the definition and defining ETL processes in the mapping metadata. If any new data source is required to populate the new data highway level, the metadata describing the data structure of the source must be obtained automatically. In this case, the developer must provide examples of data sets from the new data source to detect the data structure.

6.2.4 Deletion of a Data Highway Level

If the developer has deleted a data highway level, it must be examined whether data sets of other data highway levels depend on the deleted level. If such data sets exist, ETL processes must be modified to replace the missing data highway level. This can be done automatically if a data source that was used to retrieve the deleted data is available. If the data source is not available, this change should be treated as the change 6.2.2.

6.2.5 Addition of a Data Set

If the developer has added a data set to a data highway level, then this change is implemented manually. The developer must define the structure (data items) of the new data set, the data highway level to which the new data set is added, the mapping metadata and metadata properties. If an additional data source is required to obtain data for the new data set, it must be added by implementing the change 6.2.1.

If a new data set is added to a data source, the data set metadata should be automatically collected and the new data set must be added to the first level of the data highway.

6.2.6 Deletion of a Data Set

The following adaptation scenarios are possible for this change type, depending on the origin of the deleted data set.

If a data set is deleted from a data source, three manual adaptation scenarios are possible:

- *Replacement of a Deleted Data Set with Data from Other Data Sets*. This scenario is possible only if the developer can provide information on alternative data sets. For each data item in the deleted data set, an alternative data item from another data set must be specified or a formula that calculates data items of the deleted data set from data items of other data sets must be provided. For this adaptation scenario, it may be necessary to add metadata describing the data structure and properties of the new data set (see the change 6.2.5).
- *Data Set Skipping*. If replacement of a deleted data set is not possible, data items that depend on the deleted data set are not further updated. The adaptation scenario includes the modification of ETL processes (not filling dependent data items) and mapping metadata adjustment.
- *Hybrid Scenario*. If it is possible to replace part of a data items belonging to the deleted data set, then for such data items new ETL processes, data structure of new sources (if needed) and other properties must be defined.

If the developer has deleted a data set from a data highway level, it must be determined whether other data sets depend on the deleted set. If such datasets exist, ETL processes must be modified to replace the deleted data set. This can be done automatically if the data source that was used to retrieve the deleted data set is still available. If the data source is not available, this change must be handled as one of the scenarios described in this subsection for the case when a data set is deleted from a data source.

6.2.7 Change of Data Set Format

If only a data set format has changed and its structure (data items) has remained the same, such change can be propagated automatically if it is possible to redefine related ETL processes to use a different data set format. This implies that mapping metadata must be modified to use a different format for data load. Such a solution may also require additional information from the developer on the modification of ETL processes.

If there have been considerable changes in the format, for example from unstructured data to structured data or vice versa, then they would probably result in the change of the structure of the data set. Hence, such change must be processed as deletion and addition of data items (see the changes 6.2.9 and 6.2.12).

6.2.8 Renaming a Data Set

This change is handled automatically as it only affects metadata and possibly ETL processes. The adaptation scenario consists of renaming the data set in the table *DataSet* and transforming ETL processes to reflect the new data set name.

6.2.9 Addition of a Data Item

If a data item has been added to an existing data set which is a part of a data highway level, the metadata of the new data item must be created. If the developer has made this change, he or she must specify a name, type, and (if applicable) data warehouse role of the new data item, the data set that contains the new data item, mapping metadata, and metadata properties. If an additional data source which was not previously used in the system is required for data loading, it must be added by implementing the change 6.2.1.

If a new data item has been added to a source data set, such change must be processed automatically. Metadata describing the new data item must be collected by the adaptation component, and the new data item must be added to the data set at the first level of the data highway that corresponds to the source data set.

6.2.10 Renaming a Data Item

This change only affects metadata and possibly ETL processes, thus it is handled automatically. Similarly as in the case of data set renaming, the adaptation scenario consists of renaming the data item in the table *DataItem* and transforming ETL processes to reflect the new data item name.

6.2.11 Change of a Data Item Type

The change of a data item type can be handled automatically if it is possible to redefine ETL processes affected by the change to use another data item type. In such as case, the mapping metadata are updated to reflect the changed data type. To implement such adaptation scenario, an additional information from the developer on the modification of ETL processes might be required.

6.2.12 Deletion of a Data Item from a Data Set

Several adaptation scenarios are possible for this change type, depending on the origin of the deleted data item.

If the deleted data item belonged to a source data set, two adaptation scenarios can be applied:

- *Replacement of a Deleted Data Item with Data from Other Sources or Data Sets*. In order to implement this adaptation scenario, the developer must provide additional information on an alternative data item or a formula that calculates the deleted data item from other data items. Since the alternative data item of other data items used in the formula might not be present in the system, it may be necessary to add metadata about the structure and properties of the new data items (see the change 6.2.9).
- *Data Item Skipping*. If the deleted data item can not be replaced by others or calculated by any formula, it is necessary to determine data items of the data highway affected by the change and modify ETL processes along with the mapping metadata to skip these affected data item.

If the developer has deleted a data item from a data set that is a part of a data highway level, any other data items that have been obtained from the deleted data item must be identified by analysing the mapping metadata. If such data items exist, the deleted data item should be replaced in the mapping metadata and ETL processes. Such replacement is performed automatically if a data source from which the deleted data item was extracted is still available. If the data source is not available any more, the change must be proocessed by one of the above described scenarios.

6.2.13 Addition of a Relationship

This change may be required for processing other changes, such as addition of a new data set (see subsection 6.2.5). This change may also affect ETL processes. If a new relationship between data items is defined in the data source, this information should be recorded in the metadata. The developer must be informed of the change so that he or she decides on the further use of this information.

6.2.14 Deletion of a Relationship

If a relationship between data items has been deleted, the adaptation scenarios of this change depend on the type of the relationship.

- If the deleted relationship is a foreign key, it must be determined which ETL processes use that foreign key to connect data. The developer must be notified of the change. The information on the mapping metadata where the foreign key is used is included in the notification. Further change handling must be performed manually.
- There is a special case of a relationship with the type *Composition* that is handled differently.

Such relationships may exist between data items in XML or JSON documents. Deleting a relationship with the type *Composition* means that the structure of XML or JSON elements has changed and one of the data items has been moved elsewhere in the document structure. Thus, such information must be determined and, in order to handle the deletion of a composition relationship, a new relationship with the type Composition needs to be established according to the new document structure.

6.2.15 Addition of a Mapping

A new mapping may only be added manually by the developer using the metadata management tool. To handle this change type, a transformation function that calculates a data item from other data items must be specified. Data items must be previously created in the data highway and metadata describing them must already be available.

6.2.16 Deletion of a Mapping

If a mapping has been deleted, there are two possible adaptation scenarios:

- *Replacement of a Transformation Function.* A data item obtained using a transformation function included in the deleted mapping should be replaced by another function, if it is possible. In this case, the developer adds information about the mapping replacement and the adaptation component adds a new mapping by performing the change 6.2.15. If necessary, new data items or data sets are also added.
- *Deletion.* If it is not possible to replace the mapping by another mapping, the adaptation component marks the mapping as deleted and removes the deleted mapping from ETL processes.

6.2.17 Addition of a Metadata Property

If a new property has been added to an element in the schematic metadata, the developer must be notified about the change and must make a decision regarding further usage of the new property. Hence, this change is processed manually.

6.2.18 Deletion of a Metadata Property

If a change discovery algorithm detects a deletion of a metadata property, it is necessary to determine whether any ETL procedures use the missing property. In this case, it must be assessed whether the ETL processes can be automatically adapted or whether additional information from the developer is required.

6.2.19 Update of an Attribute Value

If a value of an attribute has been updated and that modification has not been recorded as another change type (for example, as a change of data set format or data item type), it must be checked whether the changed attribute has been used in ETL procedures. In this case, all dependent ETL procedures must be adapted to utilize the new value of the attribute.

6.3 Change Propagation

In order to successfully propagate any change to the data highway, the change handling mechanism analyzes schematic and mapping metadata as well as adaptation scenarios and operations predefined for each change type. Based on the analysis results, the mechanism populates tables *ChangeAdaptationProcess* and *CaManualConditionFulfillment* included in the adaptation metadata and tries to apply automatic conditions and operations. There are two stages of the change handling mechanism described in detail in the following sections.

6.3.1 Initial Change Processing

The goal of this stage is to determine potential change adaptation scenarios and create initial records in the tables *ChangeAdaptationProcess* and *CaManualConditionFulfillment*.

The high-level pseudocode of the initial change processing stage implemented as a procedure *CreateChangeAdaptProcess* is presented as Algorithm 1. First, the procedure selects data on changes with the status *New*. Then for each new change, the change type is determined by the function *GetChangeType* by analyzing data in the table *Change*. After that, all scenario steps predefined for the change type are selected in the correct order and the function *InsertChangeAdaptationProcess* inserts records that correspond to each scenario step into the table *ChangeAdaptationProcess*. Then, manual conditions for the current scenario step are selected from the table *ChangeAdaptationCondition*, linked with the currently processed change and saved in the table *CaManualConditionFulfillment* with the status *Not executed*. Finally, the status of the change is updated to *In progress* so that handling of this change can be considered as initiated.

Algorithm 1: Initial change processing.

```
Procedure CreateChangeAdaptProcess()
    while exists Change C where C.status =
     'New' do
        vProcessCreated ← false;
        vChangeType ← GetChangeType(C);
        if vChangeType is not null then
            foreach scenario step S that exists
             for vChangeType do
                P ← InsertChangeAdaptation-
                  Process(S,C);
                InsertManualCondition-
                  Fulfillment(P,S,C);
                vProcessCreated ← true;
            end
            if vProcessCreated then
                UpdateChangeInProgress(C);
            end
        end
    end
end
```

Algorithm 2: Scenario execution.

```
Procedure
 RunChangeAdaptationScenario(C: Change)
    Steps ←
     GetChangeAdaptationProcessSteps(C);
    foreach process step S in Steps do
        if S.StatusType = Not adapted then
            O ← GetProcStepOperation(S);
            if O.OperationType = Automatic
              and ConditionsFulfilled(C, S)
              then
                ExecuteAdaptationProc(C,O);
                SetProcessStepAdapted(S);
            end
            exit;
        end
    end
end
```

6.3.2 Scenario Execution

After initial change processing, the second stage of the change handling mechanism - scenario execution is run. Scenario execution is based on condition checks and execution of operations. Change handling steps can be both automatic and manual, and the developer can make decisions about the change handling process. If the execution of the scenario requires the intervention of the developer, the algorithm is stopped and resumed only when the developer has made his or her decision regarding manual conditions or performed the specified operation.

The Algorithm 2 demonstrates the pseudocode of the procedure *RunChangeAdaptationScenario* that executes an adaptation scenario for the specific change. First, the function *GetChangeAdaptationProcessSteps* retrieves the adaptation process steps defined during the initial change processing and stored in the table *ChangeAdaptationProcess*. Then for each step that has not been previously executed and has a status *Not adapted*, the adaptation process is continued only if it is necessary to perform an automatic operation, as well as the corresponding conditions are met. Manual conditions are checked using the table *CaManualConditionFulfillment*. For automatically executable conditions, a procedure name is obtained from the column *Condition* of the table *ChangeAdaptationCondition*. By running the corresponding procedure it is possible to evaluate the condition fulfillment. Following the same principle, the procedures for performing the steps of the change adaptation process are also executed. Their names are stored in the column *Operation* of the table *ChangeAdaptationOperation*.

6.4 Running Example

The change described in Section 4.2 corresponds to the atomic change type *Addition of a Data Item* to an existing source data set. For the running example, the change occured was discovered automatically by the change discovery algorithm during the execution of ETL process. The information about the change was registered in metadata as shown in the table 2.

According to the scenario described in Section 6.2.9 for the case of a data source extension, this change is propagated automatically. The metadata describing the structure of a new element was already gathered by the change discovery algorithm, so the change handling mechanism must just copy the metadata describing structure of the new element *citeScoreYearInfoList* from the source metadata to the metadata describing the corresponding data set at the first level of the data highway and inform the developer about new data became available. Such change does not affect the operation of the system, so the developer may anytime decide how to use new data.

7 CONCLUSIONS

In this paper, we presented our approach to dealing with changes in heterogeneous data sources caused by their evolution as well as development of infor-

mation requirements. We proposed a data warehouse architecture that includes data acquisition from various sources, as well as ETL processes for transforming data into an integrated structure so that it can be loaded into a data warehouse. The operation of the system is based on metadata that describe schema of all data sets involved in the system as well as all changes identified by the change discovery algorithm.

The main contribution of this paper is the mechanism for processing of discovered changes and changes performed manually. As a proof of concept, the proposed solution has been successfully applied to the publication data warehouse.

There are several benefits of the proposed approach comparing to manual processing of changes in data sources and information requirements. Changes of certain types are discoverable automatically and comprehensive information about changes occurred is available to the developer. Management of evolution is ensured with less human participation. Change processing is transparent as all operations performed and conditions verified are available to the developer. The proposed approach is flexible and may be extended by defining additional operations and conditions in the corresponding metadata tables, then building new change adaptation scenarios from them and assigning these scenarios to change types.

Possible directions of future work include definition of preferences regarding adaptation scenarios for various change types that are expressed by the developer to be used to choose scenarios automatically.

ACKNOWLEDGEMENTS

This work has been supported by the European Regional Development Fund (ERDF) project No. 1.1.1.2./VIAA/1/16/057.

REFERENCES

Ahmed, W., Zimányi, E., and Wrembel, R. (2014). A logical model for multiversion data warehouses. In *Data Warehousing and Knowledge Discovery*, pages 23–34, Cham. Springer International Publishing.

Bentayeb, F., Favre, C., and Boussaid, O. (2008). A user-driven data warehouse evolution approach for concurrent personalized analysis needs. *Integrated Computer-Aided Engineering*, 15(1):21–36.

Chen, S. (2010). Cheetah: A high performance, custom data warehouse on top of mapreduce. *Proc. VLDB Endow.*, 3(1–2):1459–1468.

Cuzzocrea, A., Bellatreche, L., and Song, I.-Y. (2013). Data warehousing and olap over big data: Current challenges and future research directions. In *Proceedings of the 16th International Workshop on Data Warehousing and OLAP*, DOLAP '13, page 67–70, New York, NY, USA. ACM.

Golfarelli, M., Lechtenbörger, J., Rizzi, S., and Vossen, G. (2006). Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. *Data & Knowledge Engineering*, 59(2):435 – 459.

Holubová, I., Klettke, M., and Störl, U. (2019). Evolution management of multi-model data. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 139–153, Cham. Springer International Publishing.

Kaisler, S., Armour, F., Espinosa, J. A., and Money, W. (2013). Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004.

Kimball, R. and Ross, M. (2019). The data warehouse toolkit: The definitive guide to dimensional modeling, ed. wiley.

Malinowski, E. and Zimányi, E. (2008). A conceptual model for temporal data warehouses and its transformation to the er and the object-relational models. *Data & Knowledge Engineering*, 64(1):101 – 133.

Nadal, S., Romero, O., Abelló, A., Vassiliadis, P., and Vansummeren, S. (2019). An integration-oriented ontology to govern evolution in big data ecosystems. *Information Systems*, 79:3 – 19.

Quix, C., Hai, R., and Vatov, I. (2016). Metadata extraction and management in data lakes with gemms. *Complex Systems Informatics and Modeling Quarterly*, (9):67–83.

Solodovnikova, D. and Niedrite, L. (2018). Towards a data warehouse architecture for managing big data evolution. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications*, DATA 2018, page 63–70, Setubal, PRT. SCITEPRESS - Science and Technology Publications, Lda.

Solodovnikova, D. and Niedrite, L. (2020). Change discovery in heterogeneous data sources of a data warehouse. In *Databases and Information Systems*, pages 23–37, Cham. Springer International Publishing.

Solodovnikova, D., Niedrite, L., and Niedritis, A. (2019). On metadata support for integrating evolving heterogeneous data sources. In *New Trends in Databases and Information Systems*, pages 378–390, Cham. Springer International Publishing.

Sumbaly, R., Kreps, J., and Shah, S. (2013). The big data ecosystem at linkedin. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 1125–1134, New York, NY, USA. ACM.

Wang, Z., Zhou, L., Das, A., Dave, V., Jin, Z., and Zou, J. (2020). Survive the schema changes: Integration of unmanaged data using deep learning. *arXiv preprint arXiv:2010.07586*.

Wojciechowski, A. (2018). Etl workflow reparation by means of case-based reasoning. *Information Systems Frontiers*, 20(1):21–43.