SYSTEMATIC ETL MANAGEMENT – EXPERIENCES WITH HIGH-LEVEL OPERATORS

(Practice-Oriented)

Alexander Albrecht
Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Germany
Alexander.Albrecht@hpi.uni-potsdam.de

Felix Naumann
Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Germany
Felix.Naumann@hpi.uni-potsdam.de

Abstract: Large organizations load much of their data into data warehouses for subsequent querying, analysis, and data mining. Extract-Transform-Load (ETL) workflows populate those data warehouses with data from various data sources by specifying and executing a set of transformations forming a directed acyclic transformation graph (DAG). Over time, hundreds of individual ETL workflows evolve as new sources and new requirements are integrated continuously into the system. Managing these, often complex, ETL workflows is a daunting task.
We built an ETL management framework to improve this difficult task by providing high-level operations, such as searching, matching, or merging ETL workflows. In this paper, we present our lessons learned throughout the implementation of a prototypical ETL management framework. We discuss our observations and experiences and highlight selected suggestions and algorithms, which we propose to be suitable for building useful ETL management operators.

Key Words: ETL, Data Warehousing and Repository, Data Integration

INTRODUCTION

Extract-Transform-Load (ETL) tools are visual programming tools that allow the definition of complex workflows to extract, transform, and load heterogeneous data from one or more sources into a data warehouse. ETL workflows are stored in repositories to be executed periodically, e.g., every hour, daily or once a week. In the course of a complex data integration project up to several hundred ETL workflows are created by different ETL developers [1] and stored in such repositories. ETL gained significant popularity and has established itself as a backbone in real production data warehouses: With 70%, ETL development dominates the development efforts required in current data warehouse scenarios [2].
We propose ETL management as a framework for supporting ETL workflow development and maintenance based on high-level operations, such as searching, matching, or merging entire ETL workflows. In general, an ETL management operator is an abstraction of individual ETL management tasks often done manually in today's ETL tools. The inputs and outputs for any ETL management operator are ETL workflows. For example, the input for the SEARCH operator is the entire ETL repository passing the keyword query as a parameter and it returns all ETL workflows that satisfy the specified search query.
In previous work [3, 4], the initial idea of a framework for supporting the systematic management of large ETL workflow repositories was presented. In this paper, we discuss algorithms and methods that turned out to be useful for implementing typical ETL management operators. Furthermore, we provide a detailed walkthrough of a simple ETL management scenario to highlight our main findings. We tested our implementation on real-world ETL repositories created separately by different departments of a banking organization in Switzerland (CH), Germany (DE), and Spain (ES) using Informatica PowerCenter.

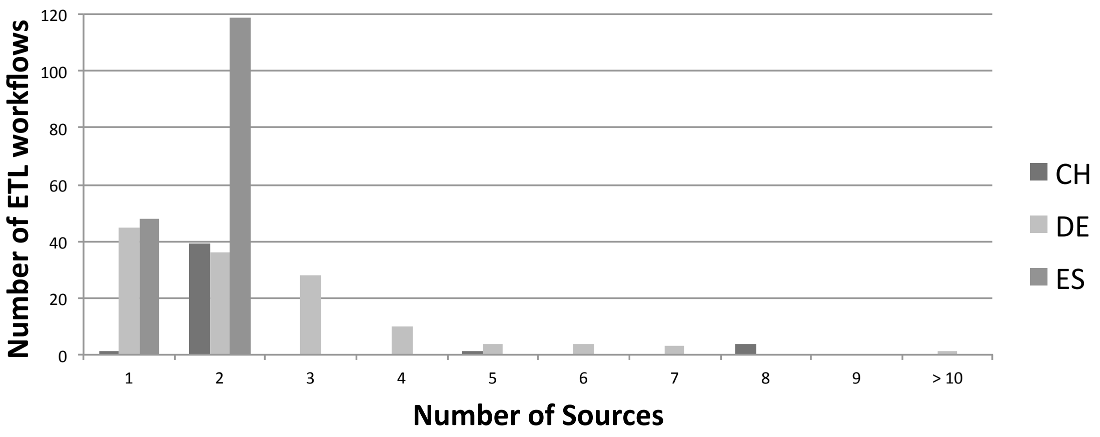


Fig. 1. ETL workflow complexity characterized by number of sources

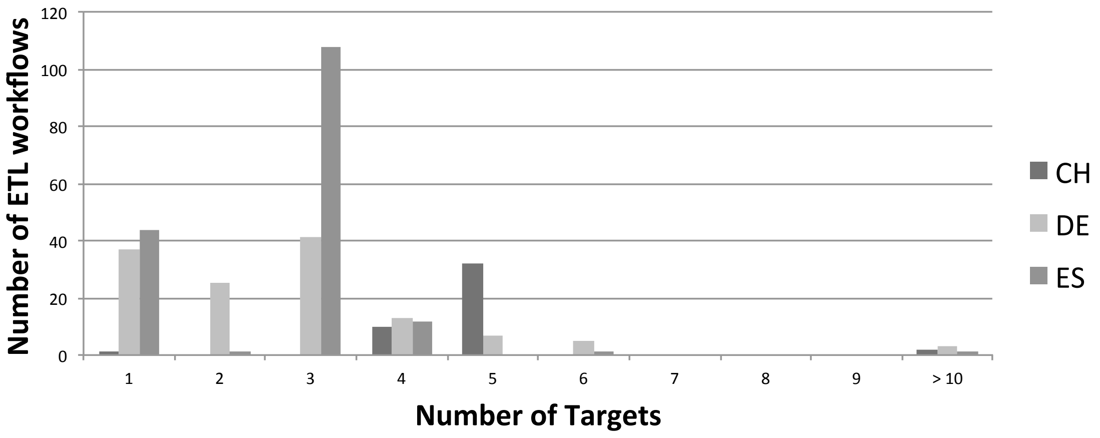


Fig. 2. ETL workflow complexity characterized by number of targets

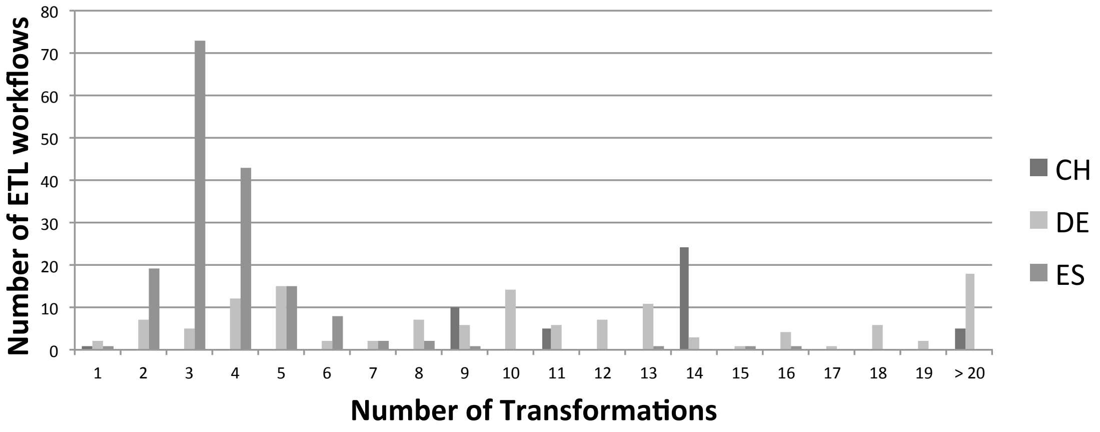


Fig. 3. ETL workflow complexity characterized by number of transformations

The test ETL repositories contain 45 (CH), 131 (DE), and 167 (ES) ETL workflows. Figures 1 – 3 show some characteristics of the three ETL repositories to emphasize that ETL development results in complex

ETL workflows. On average, each ETL workflow extracts data from 3 (CH), and 2 (DE, ES) sources. The average number of transformations is 14 (CH), 12 (DE), and 4 (ES). The average number of targets are 5 (CH), and 3 (DE, ES), respectively.

The following ETL management operators are implemented in our prototypical framework:

- IMPORT: Most commercial and open source ETL tools provide ETL workflow specifications in some proprietary XML format. To support ETL management in a tool-independent manner, IMPORT creates a tool-independent ETL workflow representation from a product-specific ETL workflow specification.
- DEPLOY: Generates from a tool-independent ETL workflow representation an ETL workflow for a specific ETL tool.
- SEARCH: Retrieves all ETL workflows that satisfy the specified search query. Search queries may address different elements of an ETL workflow, such as transformation names or database schemata.
- MATCH: Given an ETL workflow, find and rank all ETL workflows that extract, transform, or load data in a similar way.
- MERGE: Takes two or more ETL workflows as input and returns a merged ETL workflow.
- PUSHDOWN: Propagates data cleaning steps that are applied within an ETL workflow back to its data sources.

Example 1 (ETL Management). Figure 4 shows three simplified ETL workflows. All ETL workflows load data into a customer relationship management database (marked as CRM) providing a single view of customer information from different sources. Let us assume that the lower ETL workflow in Fig. 4 is the result of merging the upper two independently developed ETL workflows, one that integrates customer data from an online shop and the other loads customer data from catalog selling. A simplified address correction is performed in all ETL workflows: The data flow is split into two streams of tuples – one stream with missing zip codes, the other with existing zip codes. A tuple with a missing zip code is assigned a value using an address lookup. Finally, the two split data streams are combined into one, which is in turn loaded into the CRM database.

We use Example 1 to motivate different ETL management tasks: Due to the large number of ETL workflows created in the course of a complex data integration project, searching and then matching similar ETL workflows are typical tasks in ETL management: ETL developers begin a search for an ETL workflow by entering keywords relevant to the search goal. They then start using the returned ETL workflows as input to search for similar ETL workflows in the repository. In Example 1, the upper two ETL workflows might have been found by a keyword-based search and a subsequent similarity search against the ETL workflows in the given ETL repository. The matching ETL workflows perform the same data cleaning step and load data into the same target. They can then be merged into a single ETL workflow replacing the two matched ETL workflows in the ETL repository. In Example 1, the merged ETL workflow contains only one address lookup for all customer data. This yields to a better utilization of system resources, compared to a separate execution of both single ETL workflows, because the address-lookup is done for all customer data once. In addition, the merged ETL workflow offers an up-to-date customer view with data extracted from all sources at the same time. Finally, a pushdown of an ETL transformation is performed: Pushing the lookup transformation down to the sources directly replaces the missing data with correct address information in the original sources. In this scenario, data quality of the original data sources is improved by using a data cleaning step specified within the ETL workflow.
We regard ETL workflows as transformation graphs of the well-known model introduced by Cui and Widom [5]. This model is generally applicable to ETL workflows from common ETL tools: An ETL workflow is a directed acyclic transformation graph (DAG) and the topologically ordered graph structure determines the execution order of the connected transformations. In ETL, most transformations are a

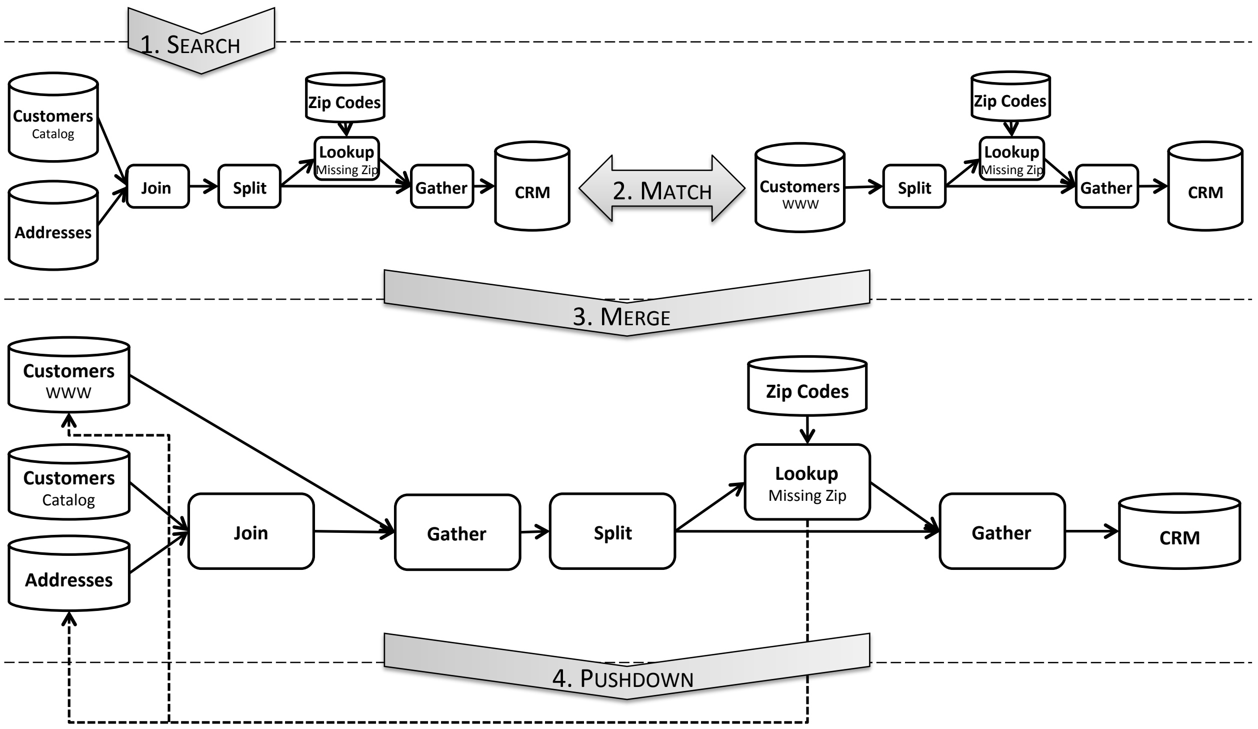


Fig. 4. Sample ETL scenario: Marketing database with a single view of customers

generalization of relational operators supporting multiple inputs and outputs. Two transformations are *connected* in the graph if one transformation is applied to the output obtained by the other transformation.

Definition 1 (ETL Workflow). An ETL workflow comprises a set of transformations T with input and output schemata, interconnected with each other forming a DAG. Let W = (V, E) be a DAG representing an ETL workflow consisting of a set of vertices V representing the involved transformations. The edges $e \in E \subseteq V \times V$ connect the output schema of one transformation with the input schema of another transformation, i.e., *e* represents an ordered pair of transformations.

The remainder of this paper is structured as follows: The next section summarizes our main contributions. In Section 3, we describe our prototype in more detail. In Section 4, we give a brief overview of the SEARCH operator implemented in our prototypical ETL management framework. Section 5 presents the MATCH operator and describes details of our similarity measures. In Section 6, we discuss our implementation of the MERGE operator and present in detail its application to ETL workflows introduced in Example 1. In Section 7, we describe the PUSHDOWN operator. Finally, we conclude this paper with a short summary and an outlook.

STATE-OF-THE-ART ETL MANAGEMENT

ETL has become a tool of major interest for companies integrating data and managing their quality. Current ETL tools focus on ETL development in terms of powerful data transformation capabilities and efficient and robust ETL workflow execution. However, state-of-the-art approaches for ETL management are mostly manual and there is a clear need for ETL management functionality in real production ETL systems. For example, in data integration projects we observe a continuous improvement of ETL workflows to ensure data quality. But data quality rules are often implemented in a single ETL workflow in a way

that they can only be used there. With PUSHDOWN we present a novel ETL management operator to improve data quality of source systems based on data quality rules specified in ETL workflows. Another ETL management task to improve data quality is merging same views on different source systems to achieve an up-to-date view on company-wide data. This task is supported by MERGE.
There are further examples showing that managing ETL workflows, such as exploring, maintaining, and reusing them, is a manual and time consuming task in today's tools. For example, the only way to explore a large ETL workflow repository is by manual browsing or by keyword-based search. There exists some functionality, such as *job compare* in IBM's InfoSphere Information Server, to report on differences between two ETL workflows. But similarity search to rank ETL workflows by similarity is not provided in current ETL tools, although this is a useful functionality to explore large ETL workflow repositories. Therefore, we propose the ETL management operator MATCH.
Some existing research concentrates on other ETL management aspects, such as optimization or configuration of ETL workflows [6]. Furthermore, there is a significant amount of recent research efforts to model ETL workflows with more abstract schema mapping languages, of which the Clio project is only one example [7]. Bernstein proposed in [8] a general framework for managing schema mappings. A technique for merging schema mappings into larger schema mappings is presented in [9]. Unfortunately, current schema mapping languages only partially capture ETL and still lack the ability to express the variety of ETL transformation steps [10].

PROTOTYPE

We implemented a prototype for generic ETL management, meaning that all high-level operators, such as SEARCH, MATCH, or MERGE, are generally applicable to arbitrary ETL workflows from different ETL tools. The generic approach is achieved by treating ETL workflows as transformation graphs in a tool-independent manner. The introduced IMPORT operator enables tool-independent ETL management by converting ETL workflow specifications from an ETL tool, such as Informatica's PowerCenter or Talend Open Studio, into a generic ETL workflow representation. In our prototype, ETL management operators are applied to the imported ETL workflows and in case of MERGE or PUSHDOWN the result is, in turn, deployed as an executable ETL workflow into the original ETL tool. To this end, we implemented the DEPLOY operator.
In our implementation, we focus only on the ETL specification representing the ETL workflow as a transformation graph and do not assume additional information. This is a realistic setting, since on the one hand most ETL tools export ETL workflow specifications in some proprietary XML format. On the other hand, it is unrealistic to assume additional information about ETL workflows, such as snapshot data processed by single ETL transformations. In real-world scenarios limited access rights, limited user permissions, and limited system resources are the main arguments against instance-based approaches: A typical ETL developer has no overall rights to access data stored in arbitrary database tables. Furthermore, extracted data is transformed in several subsequent transformation steps. As the data created in those intermediate processing steps is usually not persisted, we cannot assume to have access to such helpful information.
With our prototypical ETL management framework, we do not claim to cover the full range of possible ETL management operators, but we suggest novel operators, which we believe are useful in ETL management. In the next sections, we discuss these operators in detail and present algorithms for implementation. We refer to existing research concentrating on other ETL management aspects, such as optimization or configuration of ETL workflows [6]. These techniques from ETL research can be directly implemented as operators in our prototype, such as the ETL management operator OPTIMIZE that automates the redesign of a given ETL workflow to achieve better performance.

ETL workflow	I	II	III
Sample keyword set	Customers, Catalog, Addresses, Zip, Codes, CRM	Customers, WWW, Zip, Codes, CRM	Customers, WWW, Catalog, Addresses, Zip, Codes, CRM

Table 1: Sample keyword sets for ETL workflows from Example 1

Dictionary of keywords		**ETL workflow list**		
addresses	→	I	III	
catalog	→	I	III	
codes	→	I	II	III
crm	→	I	II	III
customers	→	I	II	III
www	→	II	III	
zip	→	I	II	III

Fig. 5. Inverted index for ETL workflows from Example 1

crm	→	I	II	III
www	→	II	III	
Intersection	⇒	II	III	

Fig. 6. Intersecting keyword lists from Fig. 5 for keyword query *WWW, CRM*

SEARCH

We have observed and experimented with ETL repositories containing many hundreds of ETL workflows: Searching ETL workflows becomes more and more important as the size of an ETL repository increases. SEARCH enables an ETL developer to easily and efficiently find all relevant ETL workflows within an ETL repository. As with traditional search engines, the SEARCH operator in our ETL management framework receives a set of keywords as input and returns a list of ETL workflows that contain all keywords in some ETL workflow elements, such as transformation names or database schemata. Table 1 shows three example keyword sets for ETL workflows from Example 1, each including keywords extracted from database table names.

To speed up the keyword search, the implementation of our SEARCH operator employs an inverted index. The inverted index comprises a dictionary of keywords from all ETL workflow elements together with a list for each keyword describing its occurrence in the repository of ETL workflows (see Fig. 5). To retrieve all ETL workflows that contain a set of keywords, we only return those ETL workflows that are present in all corresponding keyword lists of the inverted index. These ETL workflows are found by intersecting the keyword lists for each keyword from the inverted index. For efficiency, we keep the inverted index in memory. Furthermore, the computation of the intersection of keyword lists is done quickly in linear time, because the ETL workflow IDs in the keyword lists are maintained in sorted order.

Example 2 (Processing keyword queries). Consider the conjunctive keyword query *CRM WWW* and the inverted index shown in Fig. 5. For the two keyword matches found in the dictionary the corresponding keyword lists are retrieved and intersected to quickly find the two ETL workflows that contain both keywords, as shown in Fig. 6.

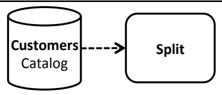
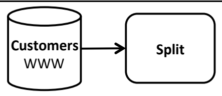
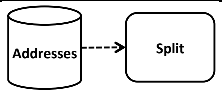
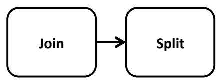
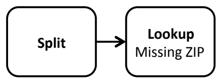
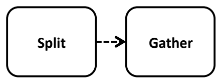
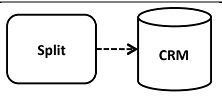
	ETL workflow I	ETL workflow II
	1/2	0
	0	1/2
	1/2	0
	1	0
	1	1
	2/3	2/3
	2/5	2/5

Fig. 7. Weights for directly and indirectly interdependent transformation pairs from Example 1

In contrast to best-match searching in traditional search engines, our experiences have revealed that ETL developers are interested in exploring all ETL workflows from the search result [11]. For example, an ETL developer who wants to replace an erroneous lookup transformation starts to submit a keyword search with the corresponding lookup table name and explores the entire search result. In this example, a result ranking would not support an ETL developer. Instead, we observed a need for meaningful filters to make accessing relevant ETL workflows more efficient. The use of ranking strategies according to some relevance criteria and focusing only on the top results is often not appropriate in ETL management. Furthermore, it is practically impossible to obtain only the relevant ETL workflows from an ETL repository if the keyword search consists of only one or two search terms.

Building upon this key observation, our implementation improves the access to the search result by providing a set of facets representing particular ETL workflow characteristics. We implemented facets as filters and the set of predefined filters includes, for example, the list of databases loaded within the returned ETL workflows. An ETL developer manually selects different filters to separate relevant ETL workflows from irrelevant ones. The set of applied filters is shown to the ETL developer and the search result is refined accordingly. Thus, an applied filter does not execute a new keyword search, but updates the way the search result is displayed. An ETL workflow belongs to the updated result list if it contains the selected filter criteria, for example a certain target database.

MATCH

Identifying similar ETL workflows brings a number of benefits in ETL management, including an improved reuse and maintenance of ETL workflows. Furthermore, discovering matching ETL workflows in large ETL repositories supports consolidation of similar ETL workflows into one integrated ETL workflow (see Example 1).

Given an ETL workflow, the MATCH operator allows to discover similar ETL workflows or subworkflows stored within the ETL repository. In analogy to information retrieval systems, the operator uses a numeric similarity measure on how well each ETL workflow in the repository matches the given ETL workflow, and ranks the result accordingly: It is hard to define a generally suitable similarity measure for ETL workflows, because of the presence of semantic or syntactic heterogeneity. Already the definition of input and output schemata for equivalent transformations causes problems, because different ETL developers often define attribute labels independently. Thus, in different schemata a variety of abbreviations, synonyms, homonyms, and hypernyms for semantically related attribute labels occur. In consequence, finding similar transformations at the schema-level becomes difficult. To make matching even more challenging, known schema matching approaches assume that the regarded schemata are at least somehow related to one another. For schemata in different ETL workflows we cannot ensure that this assumption holds.

In order to find corresponding ETL workflows within a given ETL repository, we implemented a structure-aware MATCH operator in our prototypical ETL management framework. The structure of an ETL workflow is described by pairs of transformations: We consider pairs of transformations that are explicitly connected in one ETL workflow as directly interdependent. Two transformations that are connected by a path of length greater than 1, we consider as indirectly interdependent. Each pair of transformations is associated with a weight between 0 and 1 representing the reciprocal of the average path length between the two transformations.

As we want to measure similarity between ETL workflows according to how often they contain equivalent pairs of transformations, our model is akin to the well-known vector space model from information retrieval (IR) theory [12]: MATCH uses the associated weights to create for each ETL workflow from the repository a weighted multidimensional vector of directly and indirectly interdependent transformation pairs. To further improve the matching quality, we also consider the number of common transformations between two matching ETL workflows. Therefore, we introduce additional dimensions with weight 1 to the vector space capturing single ETL transformations.

If m is the total number of all directly and indirectly interdependent transformation pairs, and n is the number of all distinct transformations used in all ETL workflows within the ETL repository, each ETL workflow is characterized with an $(n+m)$-dimensional vector. We use the cosine similarity to measure the similarity between the given ETL workflow represented by vector **A** and every other ETL workflow from the ETL repository represented by vector **B**, i.e., *cosine similarity*: $\cos(\mathbf{A}, \mathbf{B}) = (\mathbf{A} \times \mathbf{B}) / (\|\mathbf{A}\| \times \|\mathbf{B}\|)$. The calculated similarity values range from 0 to 1 since every vector contains only non-negative weights. If both compared ETL workflows are in fact the same, the calculated similarity measure is 1. Finally, all ETL workflows from the repository are ranked according to the calculated similarity values.

Example 3 (Measuring similarity between ETL workflows). Consider the two matching ETL workflows from Example 1. Figure 7 shows a portion of all associated weights for directly (solid arrows) and indirectly (dashed arrows) interdependent transformation pairs. The similarity between both ETL workflows is measured with a cosine similarity of 0.68.

Our structure-aware MATCH operator is based on the work of Jung and Bae [13]. In our approach, a transformation pair corresponds to transformations pairs in other ETL workflows if the corresponding transformations belong to the same type and their input and output schemata are similar. Our approach is not restricted to a specific type of ETL transformation, but rather, we introduce a general solution applicable to ETL transformations of different types. In our prototype, we consider ETL workflows with various transformation types including joins, filters, expressions, lookups, and aggregations. A taxonomy of ETL transformations is given in [14] for different ETL tools.

To compute the similarity of input and output schemata from corresponding transformation pairs, we use again the cosine similarity. First, we tokenize all attribute labels in a schema based on case-change or non-alphabetical characters. Then, we index all tokens from a schema with a feature vector to calculate the cosine similarity between schemata of corresponding transformations. This approach is based on the following observations: We have observed that (1) whole attribute labels are too specific to identify similar schemata in different transformations; (2) a token of an attribute label has a semantics of its own or it

modifies the semantics of another token; and (3) transformations that process similar data often contain same tokens in different attribute labels in the input and output schemata.
In our implementation of the MATCH operator, we consider only relevant attributes in the input and output schemata of a transformation, i.e., we measure the input and output similarity between two transformations based on their required and generated attributes. We perform a static analysis to detect whether an attribute in the input schema is required or directly passed through the transformation or simply projected out. We also distinguish the attributes in the output schema as pass-through attributes or attributes generated by the transformation. This analysis requires individual implementation for the different transformation types. The idea of describing ETL transformations with characteristic schemata is based on the work of Simitsis [15, 16]. Furthermore, we suggest future work to explore how schema decryption techniques [17] may further improve finding transformations with similar inputs and outputs in an ETL repository with regard to the variety of abbreviations, synonyms, and hypernyms used in labeling semantically related attributes.

MERGE

ETL tools provide an easy-to-use interface to develop ETL workflows. For example, copy-and-pasting entire sub-workflows is a common practice in ETL development. This practice results in identical sub-workflows spread over the ETL repository. The cooperation with our industrial partner shows that over time there are many ETL workflows that may encompass shared sources, same targets and identical sub-workflows. A best-practice in ETL management is to encapsulate those frequent sub-workflows as a separate ETL component. The method of making parts of an ETL workflow reusable to apply it in different ETL workflows as separate ETL components is supported in most ETL tools, such as “shared containers” in IBM’s InfoSphere Information Server or “mapplets” in Informatica’s PowerCenter.
In our prototypical ETL management framework, the MERGE operator supports an ETL developer in combining two or more ETL workflows into one integrated ETL workflow. There is still human decision-making needed to ensure that the merged ETL workflows cause no data or scheduling conflicts. Example 1 illustrates this manual process of finding non-conflicting ETL workflows using the ETL management operators SEARCH and MATCH.

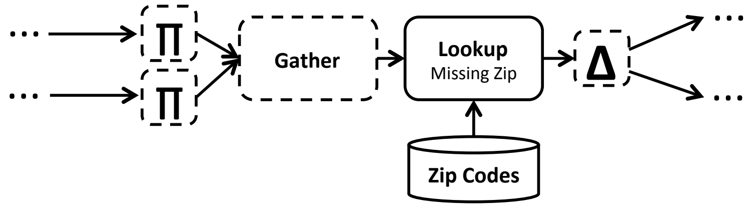


Fig. 8. Merging two identical lookup transformations into a single lookup transformation

Applying MERGE to a set of ETL workflows with identical sub-workflows, promises to enhance both the utilization of shared resources and the performance compared to executing each ETL workflow in a separate run. In addition to optimization, a merged ETL workflow provides a single view of all information that was originally processed separately. Apart from performance benefits, consolidating large sets of ETL workflows promises reduced overhead and maintenance.
Merging a set of ETL workflows starts with combining identical transformations and involves the application of rewrite and elimination rules. Equivalent transformations are merged and form common input and output data flows with compatible schemata. This can always be achieved by combining the attributes of the original data flows using a “Gather” transformation after appropriately renaming all attribute labels [18]. A combined data flow gathers the data flows of merged transformations, but each individual data flow is tagged with an additional provenance attribute. To achieve this, we introduce the two transfor-

mations Π and Δ. The transformation Π is implemented as a simple transformation adding an additional provenance attribute to the data flow. The transformation Δ is implemented as a split transformation to partition a combined data flow according to its provenance. Figure 8 shows the result of an example merge of two identical lookup transformations into a single lookup transformation based on gathering provenance tagged data flows and subsequent splitting. The behavior of the lookup transformations in the two original ETL workflows is retained, because the lookup transformation operates at the row level and we only pass the provenance attribute through the lookup transformation. This merging policy can be applied to other row-level transformations, such as filter or split. There are different merging policies for other ETL transformation types, such as join or aggregation.
In our implementation we use and extend techniques from query optimization in relational database systems. In particular, we employ a set of rewrite and elimination rules for ETL workflows to generate a best merged ETL workflow. The rewrite and elimination rules map one ETL workflow to another semantically equivalent ETL workflow, for instance by merging identical transformations, pushing a filter above a split operation, or eliminating redundant transformation pairs. To exhaustively investigate merged ETL workflow alternatives, we implemented a top-down enumeration procedure to create the search space based on step-by-step rewritings. We informally introduce some basic rules with Example 4. We refer to the work of Roy for a comprehensive overview of the used rule set [19]. That work also proposes several methods to improve the performance of the top-down search strategy. In a later publication by the same author similar techniques are applied to multi-query optimization [20].

Example 4 (Merging ETL workflows). Consider the two separate ETL workflows from Example 1. Figures 9 – 12 show the application of different rewrite and elimination rules introduced in [19, 21] to finally obtain the merged ETL workflow shown in Fig. 4 of Example 1. Transformations inserted during the application of Merge are highlighted with dashed borders.

In our implementation, equivalent sources and targets are merged in a straightforward manner. For example, merging the equivalent load of the CRM database in Fig. 9 is achieved by gathering the data flows from both ETL workflows. The new "Gather" transformation is connected to the corresponding outputs of all transformations that were originally connected to the CRM database, i.e., both Gather transformations in the original ETL workflows. This merge results in three adjacent Gather transformations. This redundancy is removed in Fig. 10 based on an elimination rule introduced in [19]. Furthermore, the equivalent address lookup is merged in Fig. 10. In the case of merging equivalent transformations, the input cannot simply be gathered: It is necessary to distinguish the input by tagging the data flow with an additional provenance attribute. This is performed with the Π transformation assigning a unique provenance number to each data flow passing through a merged transformation. The provenance attribute is used in a subsequent Δ transformation to split the output of a merged transformation accordingly. This approach of gathering provenance tagged data flows and subsequent splitting according to the provenance guarantees to preserve the transformation behavior in the original ETL workflows.
Figure 11 shows the result after applying further elimination rules, such as removing the redundant Δ transformation in Fig. 10. This transformation becomes obsolete, because both outputs are gathered in the subsequent transformation. After the application of this elimination rule, the two Π transformations in Fig. 10 are unnecessary and are removed from the ETL workflow. This removal can be performed, because the provenance attribute is never required in the subsequent transformations.
In Figure 12, the equivalent split transformation is merged. This leads to a redundant transformation pair: The Δ transformation is followed by a transformation gathering all its outputs. This redundant transformation pair is removed from the ETL workflow. The other Δ transformation and therefore both Π transformations are also redundant and removed from the ETL workflow, for the same reasons described for Fig. 11. Finally, we obtain the merged ETL workflow shown in Fig. 4 of Example 1.
One challenging problem we encountered using the MERGE operator was that already the input ETL workflows are often difficult to understand and the merged ETL workflow becomes even more complex. Therefore, gaining a big-picture understanding of all complex ETL workflows motivates us to propose the

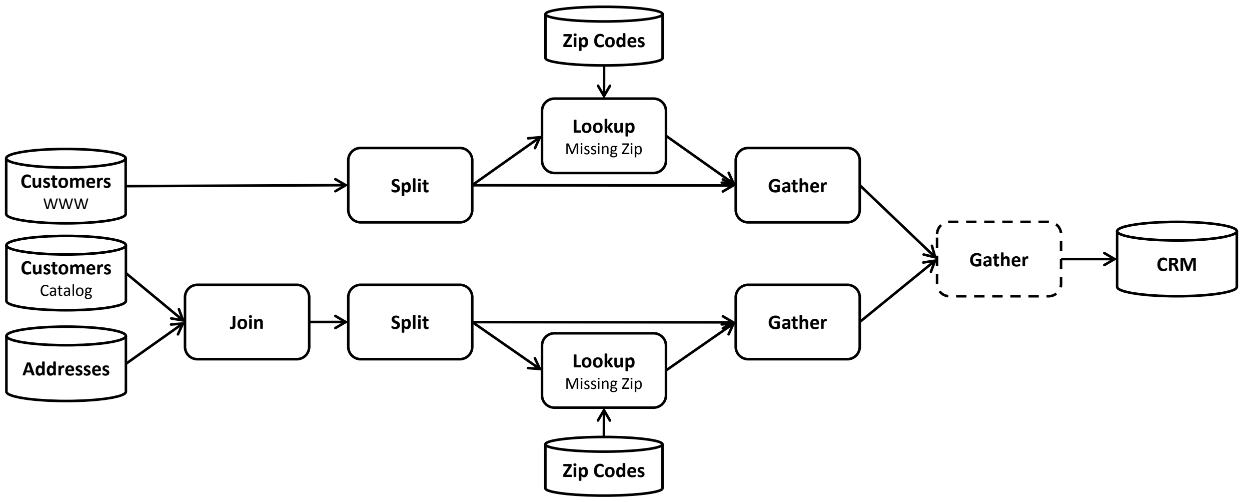


Fig. 9. Combine both ETL workflows by merging the same target CRM database

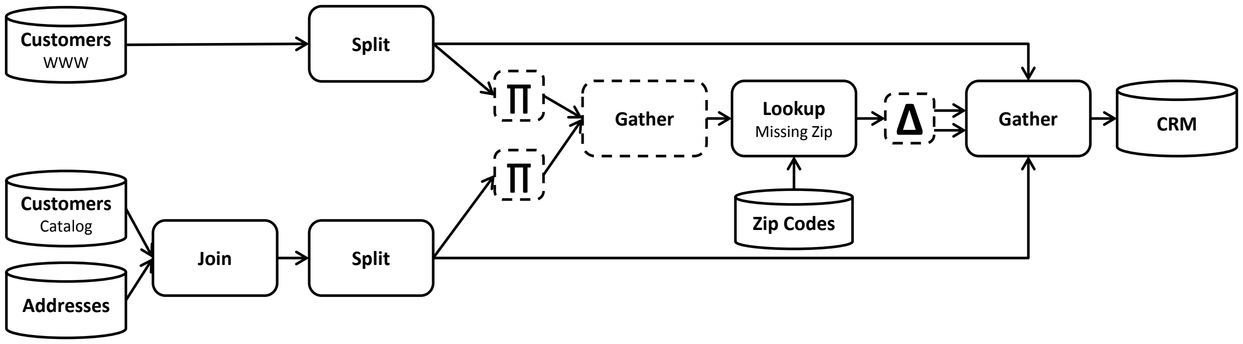


Fig. 10. Merge address lookup after coalescing adjacent gatherings of data flows

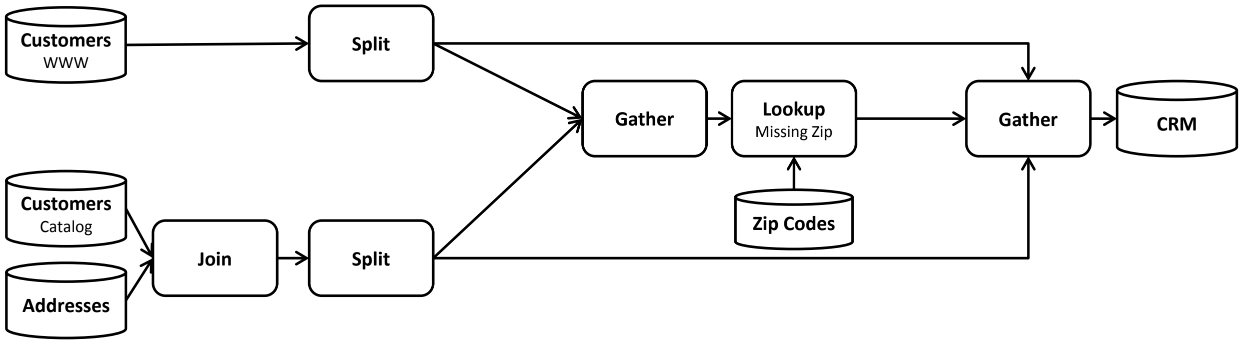


Fig. 11. Remove redundant Δ before subsequent data flow gathering and redundant Π

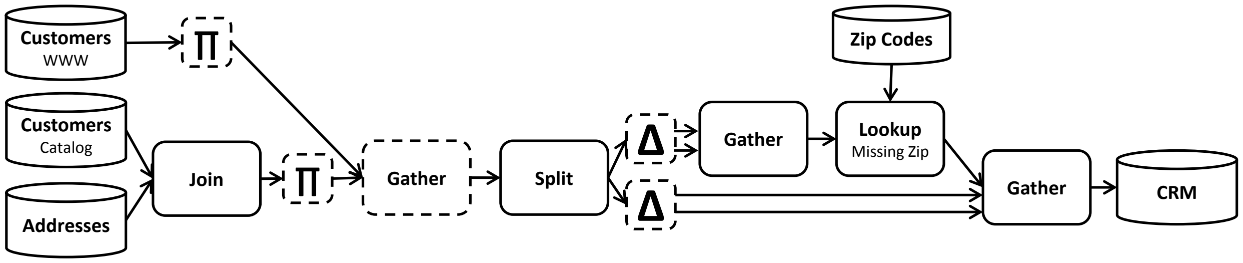


Fig. 12. Merge equivalent split transformations

concept of *ETL workflow abstraction* as future work. Because an ETL workflow that is more abstract is usually easier to comprehend, ETL workflow abstraction is a fundamental technique for coping with ETL workflow complexity in the context of MERGE.

PUSHDOWN

Real-world data extracted from external sources often contains errors. Thus, data cleaning, the task of detecting and correcting errors in data, plays an important role in data warehousing: Only high quality data produces reliable analysis and data mining results.

Within an ETL workflow specific data quality problems, such as assigning missing values, assuring referential integrity and uniqueness, can be solved using standard ETL transformations, such as lookup transformations. For instance, the ETL workflows in Example 1 complement source data lacking zip codes with a correct value using an address lookup. We consider a lookup transformation that replaces missing source values as a *candidate data repair*. Pushdown automatically discovers this type of candidate within a given ETL workflow. Then, the ETL developer decides out of all discovered repairs which ones to perform at the sources, such as replacing missing source values with data from a lookup table. Applying these cleaning steps to the sources improves data quality for applications working on the same data sources. Furthermore, it avoids multiple fixing of data quality problems in future ETL projects and thus reduces development time of new ETL workflows. The idea of sending back improved data to the sources is mentioned in the literature as backflow of cleaned data and was introduced in [22]. In that work, updating the original sources with clean data from an ETL workflow is identified as a main step in data cleaning. In the field of data integration, for example in current data warehouse environments, this task still remains a problem and requires much manual work. PUSHDOWN is a first attempt to automate the data cleaning of sources based on repairs specified in their corresponding ETL workflows.

The main problem we encountered in data cleaning with ETL tools is that the more dirty the data, the more complex it becomes to properly perform data cleaning with the fixed set of standard ETL transformations, as also observed in [23]. As data cleaning is often a complicated task, domain specific software systems are used and, if possible, embedded as “external transformations” in an ETL workflow. For example, for the address domain some ETL tools allow to link commercial address-matching systems, such as Trillium or Informatica AddressDoctor, in ETL workflows to resolve errors of addresses and consolidate address data represented in different ways or with different standards. These systems use pre-built rule sets, for example to extract from an attribute containing free-form address data its finer-grained attributes, such as street, city, state, and zip code.

The data cleaning transformations described above belong to the same class of row-level transformations, i.e., each transformation produces for each input data item, exactly one output data item. This may include the generation of new output attributes. We use the notion of generalized projection known from relation databases to describe this class of transformation. A detailed description about representing these transformations as projections can be found in [10]. This generalization allows us to apply traditional rewrite technique from query optimization in relational database systems to push such data cleaning transformations down to the sources. If a pushdown of a data cleaning transformation is possible, the result is presented to the user as a candidate data repair. For example, the pushdown of an address cleaning transformation results in an ETL workflow updating the free-form address attribute in the source using a standardized concatenation of the finer-grained address attributes extracted in the address cleaning transformation. Please note that the application of rewrite rules does not affect the original ETL workflow, i.e., the ETL workflow does not change during the execution of PUSHDOWN: Conceptually, we repeat the data cleaning transformation, i.e., the same data cleaning transformation is applied twice in the ETL workflow. This causes no side-effects, because the regarded data cleaning transformations are idempotent, i.e., applying them more than once results in the same output. Then, the pushdown is applied to the first of the two data cleaning transformations based on step-by-step rewritings of the ETL workflow.

*Example 4 (Updating sources with improved data).*Consider the merged ETL workflow from Example 1. Figures 13 – 15 show the pushdown of the data cleaning transformation replacing missing zip codes with correct data from a lookup table. Figure 16 shows the discovered candidate repairs.

Figure 13 illustrates the concept of doubling a data cleaning transformation to push it down to the sources without affecting the original ETL workflow: The lookup transformation is applied twice in succession and the first lookup transformation passes all its input attributes to the output. The output is then used as input for the original lookup transformation. Figure 14 shows interchanging the order of the first lookup transformation and split. This rewrite maps the initial ETL workflow to a semantically equivalent ETL workflow: The swapped lookup transformation captures the split condition by consolidating only the data partition with missing zip codes. To emphasize that the split condition is considered, the swapped lookup transformation is marked with a star (*) in the figure. Figure 15 shows the result after pushing the lookup transformation to the sources using traditional rewrite technique.
PUSHDOWN can be extended in a straightforward manner to support further data cleaning transformations that are not representable as a single projection. For example, duplicate elimination [24] is supported in some ETL tools, such as IBM's InfoSphere QualityStage. This data cleaning step locates and groups similar database records from a single input and identifies potential duplicate records, which are then consolidated into a single representation. This data cleaning transformation can be generalized as a combination of a projection generating a new output attribute to identify a group of duplicates, followed by an aggregation to perform the final consolidation of groups of duplicates. Similar to what was discussed above for a single projection, this generalization allows us to apply traditional rewrite techniques known from query optimization in relational database systems to push this data cleaning transformation down to the sources.

CONCLUSION

With this paper we reported our observations and experiences made during the implementation of a generic ETL management framework. We have highlighted some of our main findings and presented feasible algorithms and methods for building ETL management operators, such as SEARCH, MATCH, or MERGE. Furthermore, we investigated the usefulness of the introduced high-level operators to improve ETL development and maintenance, especially in the context of data warehousing.
Managing ETL systems is a laborious task that can be successfully accomplished only through full ETL workflow comprehension: As ETL workflows quickly become complex and difficult to understand, effective techniques are needed to support ETL developers in understanding ETL workflows in the context of ETL management. As a conclusion, we see in addition to generic ETL management operators the need for fundamental ETL workflow comprehension techniques in ETL, such as workflow abstraction or meaningful labeling of workflows. Previous work [16] already started to focus on the workflow comprehension problem. In that work, an effective and fully automated labeling technique is presented to support ETL developers in understanding cryptic schemata or schema fragments of complex ETL workflows.

ACKNOWLEDGMENT

This research was supported by InfoDyn AG; we want to thank Benjamin Böhm and Dieter Radler of InfoDyn AG for supplying large, real-world ETL repositories.

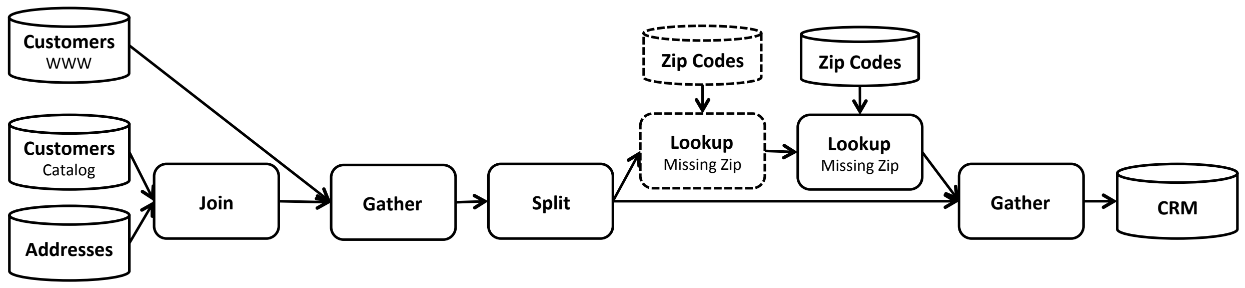


Fig. 13. Repeat the data cleaning transformation in the original ETL workflow

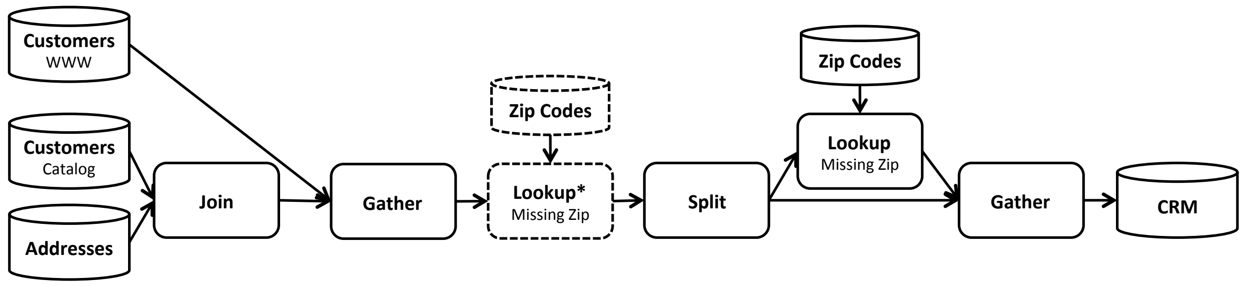


Fig. 14. Interchanging the order of split and lookup transformation

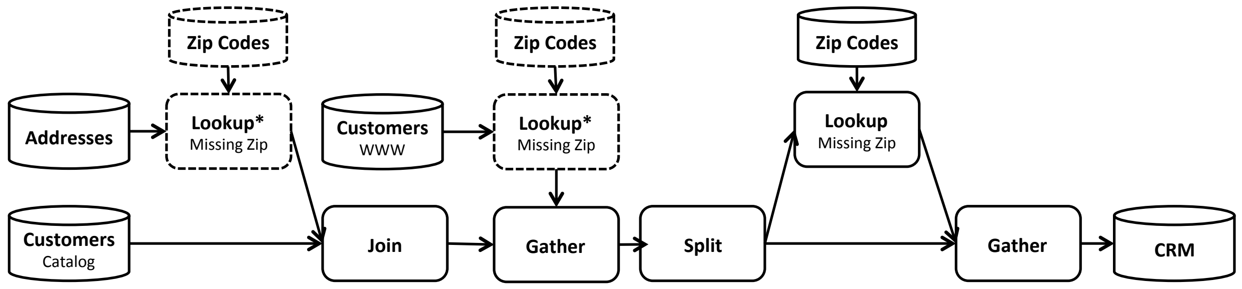


Fig. 15. Pushing lookup transformation back to the sources

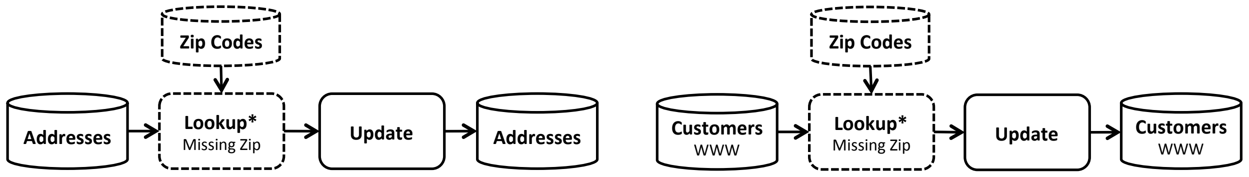


Fig. 16. Discovered candidate repairs

REFERENCES

[1] Himanshu Agrawal, Girish Chafle, Sunil Goyal, Sumit Mittal, and Sougata Mukherjea. An Enhanced Extract-Transform-Load System for Migrating Data in Telecom Billing. In Proceedings of the International Conference on Data Engineering (ICDE), Cancun, Mexico, 2008.

[2] Umeshwar Dayal, Malu Castellanos, Alkis Simitsis, and Kevin Wilkinson. Data Integration Flows for Business Intelligence. In Proceedings of the International Conference on Extending Database Technology (EDBT), Saint Petersburg, Russia, 2009.

[3] Alexander Albrecht. METL: Managing and Integrating ETL Processes. In Proceedings of the VLDB PhD workshop (VLDB), Lyon, France, 2009.

[4] Alexander Albrecht and Felix Naumann. Managing ETL Processes. In Proceedings of the VLDB International

Workshop on New Trends in Information Integration (NTII), Auckland, New Zealand, 2008.

[5] Yingwei Cui and Jennifer Widom. Lineage Tracing for General Data Warehouse Transformations. VLDB Journal, 12(1), 2003.

[6] Panos Vassiliadis and Alkis Simitsis. Extraction, Transformation, and Loading. In Encyclopedia of Database Systems, pages 1095–1101, 2009.

[7] Laura M. Haas, Mauricio A. Hernandez, Howard Ho, Lucian Popa, and Mary Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In Proceedings of the ACM International Conference on Management of Data (SIGMOD), Baltimore, MD, 2005.

[8] Philip Bernstein. Applying model management to classical meta data problems. In Proceedings of the Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, 2003.

[9] Bogdan Alexe and Mauricio A. Hernandez. Map-Merge: Correlating Independent Schema Mappings. In Proceedings of the International Conference on Very Large Databases (VLDB), Singapore, 2010.

[10] Mauricio A. Hernandez, Stefan Dessloch, Ryan Wisnesky, Ahmed Radwan, and Jindan Zhou. Orchid: Integrating Schema Mapping and ETL. In Proceedings of the International Conference on Data Engineering (ICDE), Cancun, Mexico, 2008.

[11] Thorsten Papenbrock and Sven Viehmeier. Development of the Server Achitecture for a Graphical ETL Process Management System. Bachelor's thesis, Hasso Plattner Institute, 2010.

[12] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, Boston, MA, USA, 1999.

[13] Jae-Yoon Jung and Joonsoo Bae. Workflow Clustering Method Based on Process Similarity. In Proceedings of the International Conference on Computational Science and Its Applications (ICCSA), Glasgow, UK, 2006.

[14] Panos Vassiliadis, Anastasios Karagiannis, Vasiliki Tziovara, and Alkis Simitsis. Towards a Benchmark for ETL Workflows. In Proceedings of the 5th International Workshop on Quality in Databases (QDB), Vienna, Austria, 2007.

[15] Alkis Simitsis, Panos Vassiliadis, and Timos Sellis. Optimizing ETL Processes in Data Warehouses. In Proceedings of the International Conference on Data Engineering (ICDE), Tokyo, Japan, 2005.

[16] Alkis Simitsis, Kevin Wilkinson, Umeshwar Dayal, and Malu Castellanos. Optimizing ETL workflows for Fault-Tolerance. In Proceedings of the International Conference on Data Engineering (ICDE), Long Beach, CA, USA, 2010.

[17] Alexander Albrecht and Felix Naumann. Schema Decryption for Large Extract-Transform-Load Systems. In Proceedings of the International Conference on Conceptual Modeling (ER), Florence, Italy, 2012.

[18] Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, and Alan Demers. Rule-Based Multi-Query Optimization. In Proceedings of the International Conference on Extending Database Technology (EDBT), Saint Petersburg, Russia, 2009.

[19] Prasan Roy. Optimization of DAG-structured Query Evaluation Plans. Master's thesis, Indian Institute of Technology, Bombay, 1998, http://citeseerx.ist.psu.edu/ viewdoc/summary?doi=10.1.1.46.9924

[20] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. In Proceedings of the ACM International Conference on Management of Data (SIGMOD), New York, NY, USA, 2000.

[21] Karsten Draba. Merging ETL Processes. Master's thesis, Humboldt-Universität zu Berlin, 2009.

[22] Erhard Rahm and Hong-Hai Do. Data Cleaning: Problems and Current Approaches. IEEE Data Engineering Bulletin, 23(4), 2000.

[23] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In Proceedings of the International Conference on Very Large Databases (VLDB), Rome, Italy, 2001.

[24] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating Fuzzy Duplicates in Data Warehouses. In Proceedings of the International Conference on Very Large Databases (VLDB), Hong Kong, China, 2002.