FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 50 (2025) No. 2
ISSN 0867-6356
DOI: 10.2478/fcds-2025-0006 e-ISSN 2300-3405

sciendo

Repairing ETL Processes using Extended Relational Algebra

Judith Awiti * Robert Wrembel † Esteban Zimányi ‡

Abstract. In a data warehouse architecture, heterogeneous and distributed data sources (DSs) are integrated by means of an extract-transform-load (ETL) layer, which runs integration processes (a.k.a. ETL processes). This layer is not static, since DSs being integrated change their schemas in time. A DS schema change impacts ETL processes, which typically stop working and need to be re-designed (i.e., repaired). Our overall goal is to repair automatically these ETL processes that were affected by DS schema changes. In this paper we focus on ETL processes specified by extended relational algebra, since relational data warehouses are among the most popular for business applications. For such a processes, we contribute a repair method. The method uses a rule engine that maps a possible DS schema change with: (1) an ETL operation on the changed schema element and with (2) a repair rule applicable if a DS schema element is changed. Based on this mapping, when a DS schema change occurs, our solution allows to apply adequate ETL rules to repair the affected ETL processes.

Keywords: data warehouse, ETL process, data source evolution, ETL process repair, relational algebra, API

1. Introduction

A data warehouse (DW) system architecture typically consists of: (1) a data source (DS) layer, (2) an extract-transform-load (ETL) layer, (3) a repository layer, that is, a DW or data lake (DL) [50], and (4) an analytical layer. The ETL layer runs the so-called extract-transform-load (ETL) processes that perform manipulations on data [1]. ETL processes typically perform the following tasks: (1) extract data from DSs, which often use different models, (2) transform data into a common model, (3) clean

*Université libre de Bruxelles, Belgium, judith.awiti@ulb.be
†Poznan University of Technology, Poland, robert.wrembel@put.poznan.pl
‡Université libre de Bruxelles, Belgium, esteban.zimanyi@ulb.be

data by removing missing, inconsistent, and redundant records, and (4) integrate and load data into a final storage, which is a DW. Notice that similar components are used for building a data lake architecture [26].

Most of the existing ETL tools, both commercial and open source, tacitly assume that the structure of every DS is static. Such an assumption is often incorrect, as in reality DSs change their structures (schemas) in time, and these changes may be frequent. A study of the schema evolution of 195 free open source software projects [51, 52, 54] with similar schema evolution profiles revealed that only 17% of these projects had not experienced database schema changes. The rest of the projects had experienced changes ranging from a few intra-table attribute modifications (33%) to significant schema changes (11%). Other studies [55, 56] showed that the most frequent changes in DS schemas include: (1) creating or dropping a table, (2) creating or dropping an attribute of a table, and (3) changing the definition of an attribute. Finally, MediaWiki, originally developed to run Wikipedia, had 171 different database schema versions in 4 years and 7 months of its life [16].

A model that allows the visualization and understanding of how the different entities of a relational database schema co-evolve was developed to mine interesting patterns of change from a large number of schema histories [22]. The model was applied to study the presence of patterns in a dataset containing 195 schema histories of Free-Open Source projects, collected from GitHub with specific collection criteria filtering out projects with 0 stars or just 1 contributor, DDL files with 'example', 'demo', 'test', terms in their path, and, projects without a history of versions for the DDL file. This study shows that a massive 'birth' of entities were present in 56% of the studied projects. This was due to the fact that most of the analyzed databases start with a 'big-bang' of introducing a significant percentage of their schema in the 0-th version. The authors discovered also less frequent schema change patterns, like: the removal of tables (present in just 9% of the projects), the progressive expansion in subsequent schema versions (in 19% of the studied projects), and the existence of massive updates (in 21% of the projects).

In the analysis of schema changes presented in [18], tables were classified in increasing order of complexity as: isolated (not involved in foreign keys), source (with outgoing foreign keys only), lookup (with incoming foreign keys only), and internal (with both kinds). The study revealed that less complex tables are more likely to have a life with few or zero schema updates, whereas more complex tables are more likely to undergo high numbers of updates. Early versions of the analyzed database includes the births of complex tables, as opposed to the simple ones, demonstrating a pattern of reducing the creation of complex, heavily updated tables in the schema as time progresses.

It is evident that DS schema changes impact the ETL, repository, and analytical layers in the DW architecture. For this reason, such changes have to be registered and their impact on the DW system layers has to be tracked. The industry accepted means of handling change tracking is the so-called *lineage* methods.

A *data lineage* [2] tool allows a user to monitor where data is coming from, where data is being transported to, and what transformations are applied to data as it flows through an ETL process. Some organisations rely on data lineage tools to trace errors

in their data pipeline as a result of DS schema changes. A *structure lineage* provides means to track dependencies between data objects in a DW/DL system, i.e., between a DS schema, ETL objects, and a DW schema. Even though data lineage and structure lineage tools show which part of an ETL process was impacted by a DS change, they do not provide means for automatic or at least semi-automatic reparation of affected ETL processes. As a consequence, such repairs have to be done manually.

In practice, large companies deploy dozens or even hundreds of thousands of ETL processes (e.g., in banking or pharma). Thus, a manual reparation of ETL processes is complex and costly w.r.t. time and money. Since structural changes in DSs are frequent, being able to use a solution for an automatic or semi-automatic reparation of an ETL process after DS schema changes would decrease ETL maintenance time and cost. Handling and incorporating structural changes to the ETL layer received so far little attention from the research community [3, 35, 36, 40, 46, 58]. As a consequence, none of the commercial or open-source ETL tools existing on the market supports this functionality.

This paper is part of a series of papers that attempts to address the issue of failing ETL workflows as a result of DS schema changes. The papers manage ETL processes in two ways. First, their modelling and second, their reparation upon DS schema changes. BEXF [7], an Extensible Markup Language (XML) interchange format for BPMN4ETL, an extended BPMN model for ETL was developed to express and interchange BPMN4ETL model information across tools that are compliant with BPMN 2.0. In [4, 5], relational algebra (RA) is extended with update operations for specifying ETL processes at a logical level. Data tasks can be automatically translated from RA operations into SQL queries and executed over a DBMS. The authors provide a translation mechanism from BPMN4ETL to the Extended RA specification. The overall Framework for ETL workflow reparation called Extended Evolving-ETL (E3TL) is proposed for the first time in [3]. The E3TL framework makes it feasible to develop algorithms for (semi-) automatic repair of ETL workflows upon DS schema changes. E3TL is further extended [6] based on three concepts: (1) Rule-Based-Reasoning, (2) Case-Based-Reasoning, and (3) Automatic Discovery of Rules from cases.

In this paper, we **contribute** a repair method for ETL processes affected by a DS schema change. The method uses rules that associate: (1) a possible DS schema change with (2) an ETL operation on the schema element and with (3) a repair rule applicable if a DS schema element is changed. Our solution prototype applies the applicable ETL repair rules to repair the affected ETL processes upon DS schema change. In this method, a set of default rules defined in a database table is applied automatically to repair a deployed ETL process upon DS schema changes. The default rules are developed to fit any ETL scenario and are extensible, hence an ETL developer does not need to develop them beforehand. Our method allows a user to propose other solutions for ETL process reparation in which case the proposed solution can replace the default ones for future use.

This paper is organized as follows. Sec. 2 describes the Extended RA specification for ETL processes. Sec. 3 discusses a running example. Sec. 4 outlines basic schema modification operations. Based on the running example, we show in Sec. 5, how an ETL process is repaired after schema modifications of Sec. 4 are applied at the

DS. In Sec. 6 we discuss our method of an automatic ETL process repair. Sec. 8 discusses related research approaches to handling ETL process repairs. Finally, Sec. 9 summarizes and concludes the paper.

2. ETL Process Specification

This section presents the Extended RA [4, 5] specification for ETL processes. Extended RA extends well-known RA operations with update operations. RA is a well-studied formal language, that can be used to automatically generate SQL queries to be executed over any Relational Database Management System (RDBMS). The Extended RA operators are shown in Tab. 1. The left table shows typical RA operations (following the concepts and notations presented in [21]). The right table, shows Extended RA operators, discussed by the authors in [4, 5]. Below we outline the Extended RA operators used in this paper.

Table 1. RA operators (left), Extended RA (right)

Operator	Notation
Selection	$\sigma_C(R)$
Projection	$\pi_{A_1,..,A_n}(R)$
Cartesian Product	$R_1 \times R_2$
Union	$R_1 \cup R_2$
Intersection	$R_1 \cap R_2$
Difference	$R_1 - R_2$
Join	$R_1 \bowtie_C R_2$
Natural Join	$R_1 * R_2$
Left Outer Join	$R_1 ⟕_C R_2$
Right Outer Join	$R_1 ⟖_C R_2$
Full Outer Join	$R_1 ⟗_C R_2$
Semijoin	$R_1 \ltimes_C R_2$

Operator	Notation
Aggregate	$\mathcal{A}_{A_1,..,A_m \mid C_1=F_1(B_1),..,C_n=F_n(B_n)}(R)$
Extend	$\mathcal{E}_{A_1=Expr_1,...,A_n=Expr_n}(R)$
Input	$R \leftarrow \mathcal{I}_{A_1,...,A_n}(F)$
Insert	$R \leftarrow R \cup S$ or $R \leftarrow S$
Lookup	$R \leftarrow \pi_{A_1,..A_n}(R_1 \bowtie_C R_2)$
Rename	$\rho_{A_1 \leftarrow B_1,...,A_n \leftarrow B_n}(R)$ or $\rho_S(R)$

Aggregate: Let F be an aggregate function such as Count, Min, Max, Sum, or Avg. The aggregate operator $\mathcal{A}_{A_1,...,A_m \mid\ C_1=F_1(B_1),...,C_n=F_n(B_n)}(R)$ partitions the tuples of R in groups that have the same values in attributes A_i and computes for each group new attributes C_i by applying the aggregate function F_i to the values of attribute B_i in the group, or the cardinality of the group if $F_i(B_i)$ is Count$(*)$. If no grouping attribute is given (i.e., $m = 0$), the aggregate functions are applied to the whole relation R. The schema of the resulting relation has the attributes $(A_1, \ldots, A_m, C_1, \ldots, C_n)$.

Extend: For a given relation R, the extend operation returns a relation where each tuple from R is extended with new attributes A_i, obtained by computing expression Exp_i. This operation is denoted as $\mathcal{E}_{A_1=Exp_1,...,A_n=Exp_n}(R)$. The schema of the resulting relation contains the attributes from R.

Input: This operation, denoted by $R \leftarrow \mathcal{I}_{A_1,...,A_n}(F)$ returns a relation with schema $R(A_1, \ldots, A_n)$ that contains a set of tuples from the content of a relation or file F.

Insert: Given two relations R and S, this operation, denoted $R \leftarrow R \cup S$, adds

to R the tuples from S. When a new relation R is created with the contents of S, the operation is denoted $R \leftarrow S$. Also if the two relations have different arity, the attributes of the second relation must be explicitly stated as, e.g., $R \leftarrow R \cup \pi_{B_1,\ldots,B_n} S$.

Lookup: The lookup operation $R \leftarrow \pi_{A_1,\ldots A_n}(R_1 \bowtie_C R_2)$, joins two relations R_1 and R_2 and returns tuples that satisfy a condition C. The join operation can be any of the six types in Tab. 1.

Rename: This operation is applied over relation names or over attribute names. For the former, the operation is denoted by $\rho_S(R)$, where the input relation R is renamed to S. For attributes, $\rho_{A_1 \leftarrow B_1,\ldots,A_n \leftarrow B_n}(R)$, returns a relation where the attributes A_i in R are renamed to B_i, respectively.

3. Running Example

This section presents an example to be used throughout this paper. Fig. 1 shows the schema of a DS table ProductDS and a DW table ProductDW. Eqs. 1 to 4 show the operations that load data from ProductDS into ProductDW.

The schema of the ProductDS table includes a unique primary key ProdID and other attributes describing the product name (ProdName), product description (ProdDesc), quantity per unit (QtyPerUnit), unit price (UnitPrice), department (Dept), supplier name (SupName), and supplier address (Address). The schema of the ProductDW

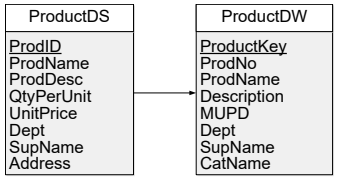

Figure 1. Original DS and DW schemas

table includes a surrogate key ProductKey, and a business key ProdNo, which is a copy of the ProdID attribute from the ProductDS table. The remaining attributes describe the product name (ProdName), product description (ProdDesc), maximum unit price per department (MUPD), department (Dept), supplier name (SupName), and category name (CatName). All attributes except ProductKey, ProdNo, and ProdName allow the insertion of null values.

3.1. Relational Algebra

The Extended RA ETL process specification begins with an extraction of all tuples in the ProductDS table into a relation called Temp0 (Eq. 1). Then, two attributes ProdID and ProdDesc are renamed to ProdNo and Description respectfully and the resulting relation is stored in Temp1 (Eq. 2). Next, Eq. 3 aggregates tuples of Temp1 grouping them by the attribute Dept and computing for each group a new attribute

called MUPD which is the maximum UnitPrice. The resulting relation is stored in Temp2. Temp1 is joined to Temp2 on the Dept attribute and all tuples are inserted into the ProductDW table (Eq. 4). A NULL value is automatically inserted in the CatName column of ProductDW since this column does not exist in the ETL process.

$$\text{Temp0} \leftarrow \mathcal{I}_{\text{ProdID,ProdName,ProdDesc,QtyPerUnit,UnitPrice,Dept,SupName}}(\text{ProductDS}) \quad (1)$$

$$\text{Temp1} \leftarrow \rho_{\text{ProdID}\leftarrow\text{ProdNo},\text{ProdDesc}\leftarrow\text{Description}}(\text{Temp0}) \quad (2)$$

$$\text{Temp2} \leftarrow \mathcal{A}_{\text{Dept}|\text{MUPD}=\text{MAX}(\text{UnitPrice})}(\text{Temp1}) \quad (3)$$

$$\text{ProductDW} \leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,Description,MUPD,Dept,SupName}}(\text{Temp1}) \rtimes_{\text{Dept} = \text{Dept}} \text{Temp2}) \quad (4)$$

To show the practicability of the Extended RA operations used in this section, we provide an SQL translation of Eqs. 1 to 4 in Appendix A.

4. Schema Modifications

In our work we incorporated a database evolution language (DEL) to specify schema modifications at a DS. A DEL contains operations which specify units of changes that modify the schema of a database. Such operations are called Schema Modification Operations (SMOs). SMOs typically include addition, deletion, and partitioning of tables and columns and evolve new versions from existing database schema versions. Bidirectional DEL (BiDEL) [27, 28] is an SMO-based DEL that facilitates the propagation of data between schema versions allowing the storage of physical data in any schema version or in a combination of tables from different schema versions. CoDEL [29] is a relationally complete SMO-based DEL in which all evolution steps can be specified with a given set of SMOs. Hence, a user does not need to fall back on traditional SQL in any situation. Below we show the SMOs of CoDEL used in this paper.

- **REN**(table): **REN**(table)(R, R') renames table R into R'.
- **REN**(column): **REN**(column)(R, c, c') renames column c in table R into c'.
- **SPLIT**(row): **SPLIT**(row)$(R, (S, cond_S), (T, cond_T))$ takes table R and partitions it horizontally into two tables S and T, based on conditions $cond_S$ and $cond_T$, respectively. Both conditions $cond_S$ and $cond_T$ are independent. If the original tables contain rows that fulfill neither of the conditions, the resulting partitioning is incomplete. Rows that fulfill both conditions are copied resulting in overlapping partitions.
- **UNITE**(row): **UNITE**(row)(R, S, T) is the inverse operation of **SPLIT**(row). It merges two given tables R and S into a resulting table T along the row dimension and removes the original tables. The schema of R and S are not required to be equivalent and duplicates in T are eliminated. In case both schema differ, T contains null values in the respective cells and vice versa.

- **SPLIT**(column): **SPLIT**(column)$(R, (S, \{s_1, ..., s_n\}), (T, \{t_1, ..., t_m\}))$ partitions table R vertically into two tables S_i and T_i. The partitions overlap on a common column to maintain the connection between the resulting tables.
- **UNITE**(column): **UNITE**(column)$(R, S, T, cond, o)$ is the inverse operation of **SPLIT**(column) and joins two tables R and S, based on a given condition *cond* and optionally a boolean o to indicate an outer join. In case $o = \top$, an outer join is performed, so that no rows from the original tables are lost. In case $o = \perp$ (or not specified) an inner join is performed. With the inner join, **UNITE**(column) loses all rows from R and S that do not find a join partner, since R and S are dropped after the join.
- **ADD**(column): **ADD**(column)$(Ri, c, f(c_1, ..., c_n))$ applies function $f(c1, ..., cn)$ to each row of Ri to calculate the row value for new column c. Function f is a standard SQL function that accesses existing values of a row.
- **DEL**(column): **DEL**(column)(R_i, c) removes a column c from table R_i.
- **ADD**(table): **ADD**(table)$(R, \{c1, ..., cn\})$ creates empty table R with columns c_i.

The SMOs described above are representative of the changes that occur frequently in any DW project [16, 55, 56, 60]. We consider in Fig. 2, 12 schema modifications from the SMOs described in CoDEL [29] on DS table ProductDS. The arrows in Fig. 2 indicate the transformation from one schema to another.

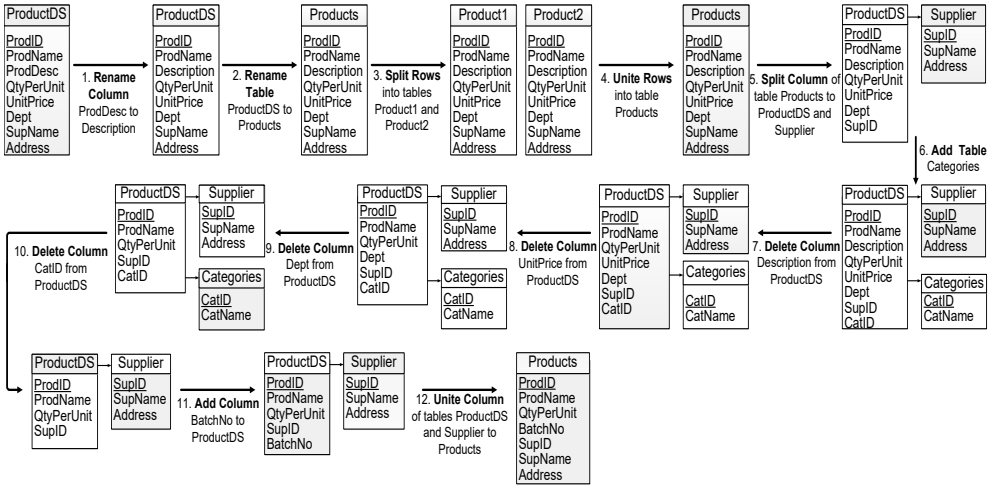

Figure 2. Schema modifications of the running example

5. ETL Repair by Means of Extended Relational Algebra

The solution that we propose for repairing ETL processes is based on seven assumptions that concern a DS and a DW, to enable a successful application of the automatic ETL process reparation described in this paper. Taking into account the assumptions, we show with the running example how an ETL process is automatically repaired after each DS schema modification visualized in Fig. 2.

Assumptions

1. *The schema of a DW is not being changed as the result of a DS change.* This behaviour is motivated by the fact that only a DW schema designer has adequate knowledge to correctly modify the DW schema. For example, adding an attribute to a DS may not necessarily be reflected in the DW, if values of the attribute are not analyzed in applications. Removing an attribute from a DS does not impact the schema of the DW either, as it would cause the loss of historical data. Moreover, modifying a DW schema automatically would stop the work of analytical applications.

2. *During the execution of an ETL process, data is held in temporary tables.* These tables are stored in an operational data store (a.k.a. data stage, staging area, landing zone) [45]. This behaviour prevents from any interaction between the ETL layer and the DW.

3. *Tables are split in a way that it guarantees the completeness, reconstruction, and disjointness of the results.* These standard split (partitioning) criteria guarantee that inconsistent data are not produced.

4. *User-defined functions* (UDFs) *are not used in ETL workflows.* This is a limitation, but at this stage of our research we focus on standard operators. UDFs may be written in multiple programming languages and being analyze and understand their semantics is a separate research problem. Moreover, UDFs are often made available in ETL workflows as black-boxes, i.e., its internal logic is opaque to an ETL engine. As a consequence, before being able to repair an ETL workflow with a black-box UDF, one has to discover its internal logic. Again, this is a separate (extremely difficult) problem, addressed in some publications (see [12] for the overview of the problem).

5. *Detecting schema changes at DSs is made by an external dedicated module.* For each DSs such a module exist and it delivers detected schema changes to an ETL engine. The detection mechanism depends on: (1) the functionality of a DS and (2) the possibility of accessing a DS schema by the module. If: (1) a DS supports triggers on its schema, (2) such triggers can be created, (3) and the module can access data produced by the triggers, then such a mechanism is used. Otherwise, the detection mechanism compares two schema snapshots (i.e., metadata describing the schema) made available by the DSs. From our practice, the most frequently, the snapshot detection mechanism is the only available.

6. *ETL repairing rules* (see Section 6) provided by our solution *are maintained by an ETL engine administrator and are extensible*, i.e., if a new rule is designed, it is accepted by the administrator and included into the Rule Engine.

7. In our repair of ETL processes, *we focus on structural changes in the DS*. We assume that Primary Key or Foreign Key constraints between tables at the DS are automatically dropped before a key column is deleted.

Notice that some of the assumptions allowed us to simplify the problem to such one that is solvable and manageable in our approach. However, alleviating some of the assumptions will open new research paths, as discussed in Section 9.

Example Using the SMOs outlined in Sec. 4 to specify each of the twelve schema transformations of Fig. 2, we describe below the incremental ETL process repair required after each transformation occurs.

$\mathbf{REN}_{column}(ProductDS, ProdDesc, Description)$: when a column in a DS is renamed, an automatic reparation of the ETL process requires a renaming of the column at its entry point(s) into the ETL process. The entry point of the ProdDesc column is the Input operation hence, Eq. 1 becomes Eq. 5. Since the new column name (Description) now corresponds to that of the ProductDW table, it is removed from the Rename operation (Eq. 2 becomes Eq. 6). We show the transformed ETL process of the running example with Eqs. 5 to 8.

$$\mathsf{Temp0} \leftarrow \mathcal{I}_{\mathsf{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName}}(\mathsf{ProductDS}) \quad (5)$$

$$\mathsf{Temp1} \leftarrow \rho_{\mathsf{ProdID} \leftarrow \mathsf{ProdNo}}(\mathsf{Temp0}) \quad (6)$$

$$\mathsf{Temp2} \leftarrow \mathcal{A}_{\mathsf{Dept|MUPD=MAX(UnitPrice)}}(\mathsf{Temp1}) \quad (7)$$

$$\mathsf{ProductDW} \leftarrow \mathsf{ProductDW} \cup (\pi_{\mathsf{ProdNo,ProdName,Description,MUPD,Dept,SupName}}\,\mathsf{Temp1} \rtimes_{\mathsf{Dept}\,=\,\mathsf{Dept}}\,\mathsf{Temp2}) \quad (8)$$

$\mathbf{REN}_{table}(ProductDS, Products)$: similar to renaming a column, the only repair required in an ETL process after a table is renamed at the source is to update the table name at its entry point(s) in the ETL process (Eq. 5 becomes Eq. 9). Eqs. 9-12 shows the transformed ETL process.

$$\mathsf{Temp0} \leftarrow \mathcal{I}_{\mathsf{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName}}(\mathsf{Products}) \quad (9)$$

$$\mathsf{Temp1} \leftarrow \rho_{\mathsf{ProdID} \leftarrow \mathsf{ProdNo}}(\mathsf{Temp0}) \quad (10)$$

$$\mathsf{Temp2} \leftarrow \mathcal{A}_{\mathsf{Dept|MUPD=MAX(UnitPrice)}}(\mathsf{Temp1}) \quad (11)$$

$$\mathsf{ProductDW} \leftarrow \mathsf{ProductDW} \cup (\pi_{\mathsf{ProdNo,ProdName,Description,MUPD,Dept,SupName}}\,\mathsf{Temp1} \rtimes_{\mathsf{Dept}\,=\,\mathsf{Dept}}\,\mathsf{Temp2}) \quad (12)$$

$\mathbf{SPLIT}_{row}(Products, (Product1, cond_S), (Product2, cond_T))$: this SMO creates two horizontal partitions of a given table. Each partition is stored in a different table and the original table is removed. To automatically repair the ETL process, two Input operations (Eqs. 13 and 14) are needed to fetch data from each of the new partitions (Product1 and Product2). Afterwards, a Union operation is used to merge the fetched data into one (Eq. 15). The transformed ETL process is shown in Eqs. 13 to 18.

$$\begin{aligned}
\text{Temp0} &\leftarrow \mathcal{I}_{\text{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName}}(\text{Product1}) && (13)\\
\text{Temp1} &\leftarrow \mathcal{I}_{\text{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName}}(\text{Product2}) && (14)\\
\text{Temp2} &\leftarrow \text{Temp0} \cup \text{Temp1} && (15)\\
\text{Temp3} &\leftarrow \rho_{\text{ProdID}\leftarrow\text{ProdNo}}(\text{Temp2}) && (16)\\
\text{Temp4} &\leftarrow \mathcal{A}_{\text{Dept}\mid\text{MUPD=MAX(UnitPrice)}}(\text{Temp3}) && (17)\\
\text{ProductDW} &\leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,Description,MUPD,Dept,SupName}}\,\text{Temp4} \rtimes_{\text{Dept = Dept}}\,\text{Temp3}) && (18)
\end{aligned}$$

UNITE$_{row}(Product1, Product2, Products)$: this SMO is the inverse of the Split Rows SMO, described above. Therefore all changes made to the ETL process in Eqs. 13 to 15 are reversed, as shown in Eqs. 19 to 22.

$$\begin{aligned}
\text{Temp0} &\leftarrow \mathcal{I}_{\text{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName}}(\text{Products}) && (19)\\
\text{Temp1} &\leftarrow \rho_{\text{ProdID}\leftarrow\text{ProdNo}}(\text{Temp0}) && (20)\\
\text{Temp2} &\leftarrow \mathcal{A}_{\text{Dept}\mid\text{MUPD=MAX(UnitPrice)}}(\text{Temp1}) && (21)\\
\text{ProductDW} &\leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,Description,MUPD,Dept,SupName}}\,\text{Temp1} \rtimes_{\text{Dept = Dept}}\,\text{Temp2}) && (22)
\end{aligned}$$

SPLIT$_{column}(Products, (ProductDS, \{ProdID, ProdName, Description, QtyPerUnit, UnitPrice, Dept\}), (Supplier, \{SupName\}))$: Split Column vertically splits a table into two tables and removes the original table. As stated in the assumptions, in splitting a table, the completeness, reconstruction, and disjointness of the resulting tables is guaranteed. In the running example, a foreign key constraint on column SupID of ProductDS ensures referential integrity between ProductDS and Supplier. To automatically repair the ETL process, we join tables ProductDS and Supplier in the Input operation on the column SupID (Eq. 23). Eqs. 23 to 26 shows the repaired ETL process.

$$\begin{aligned}
\text{Temp0} &\leftarrow \mathcal{I}_{\text{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName}}(\text{ProductDS} \bowtie_{\text{SupID = SupID}}\,\text{Supplier}) && (23)\\
\text{Temp1} &\leftarrow \rho_{\text{ProdID}\leftarrow\text{ProdNo}}(\text{Temp0}) && (24)\\
\text{Temp2} &\leftarrow \mathcal{A}_{\text{Dept}\mid\text{MUPD=MAX(UnitPrice)}}(\text{Temp1}) && (25)\\
\text{ProductDW} &\leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,Description,MUPD,Dept,SupName}}\,\text{Temp1} \rtimes_{\text{Dept = Dept}}\,\text{Temp2}) && (26)
\end{aligned}$$

ADD$_{table}(Categories, \{CatID, CatName\})$: naturally, a new table added to the DS is not automatically included into the ETL process unless an updated business requirement requires this. This may be the case if the new table is referenced by another table already present in the ETL process and/or at least one column of the new table exist in the DW table. As seen in Fig 1, the CatName column already exists in the DW table ProductDW. In the sixth operation of Fig 2, table Categories is added with a reference column CatID which is added to table ProductDS as well. To add CatName to the ETL process, we need to join the Categories table to the ProductDS table in (Eq. 27), and add CatName to the inserted columns of Eq. 30. Eqs. 27 to 30 shows the repaired ETL process.

$$\text{Temp0} \leftarrow \mathcal{I}_{\text{ProdID,ProdName,Description,QtyPerUnit,UnitPrice,Dept,SupName,CatName}}(\text{Categories} \bowtie_{\text{CatID = CatID}} \quad (27)$$

$$\begin{aligned}
&\text{ProductDS} \bowtie_{\text{SupID} = \text{SupID}} \text{Supplier})\\
\text{Temp1} \quad &\leftarrow \quad \rho_{\text{ProdID} \leftarrow \text{ProdNo}}(\text{Temp0}) && (28)\\
\text{Temp2} \quad &\leftarrow \quad \mathcal{A}_{\text{Dept}\mid\text{MUPD}=\text{MAX}(\text{UnitPrice})}(\text{Temp1}) && (29)\\
\text{ProductDW} \quad &\leftarrow \quad \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,Description,MUPD,Dept,SupName,CatName}} && (30)\\
&\quad \text{Temp1} \rtimes_{\text{Dept} = \text{Dept}} \text{Temp2})
\end{aligned}$$

$\mathbf{DEL}_{column}(ProductDS, Description)$: The repair of an ETL process after a column deletion at the source depends on whether the deleted column is used in an expression/condition or not. After column Description is input in the ETL process, it is not found in any condition or expression of any of the ETL operations until it is finally inserted in the ProductDW table. To repair the ETL process, we need to remove column Description from the Input (Eq. 31) and Insert (Eq. 34) operations.

$$\begin{aligned}
\text{Temp0} \quad &\leftarrow \quad \mathcal{I}_{\text{ProdID,ProdName,QtyPerUnit,UnitPrice,Dept,SupName,CatName}}(\text{Categories} \bowtie_{\text{CatID} = \text{CatID}} \text{ProductDS} && (31)\\
&\quad \bowtie_{\text{SupID} = \text{SupID}} \text{Supplier})\\
\text{Temp1} \quad &\leftarrow \quad \rho_{\text{ProdID} \leftarrow \text{ProdNo}}(\text{Temp0}) && (32)\\
\text{Temp2} \quad &\leftarrow \quad \mathcal{A}_{\text{Dept}\mid\text{MUPD}=\text{MAX}(\text{UnitPrice})}(\text{Temp1}) && (33)\\
\text{ProductDW} \quad &\leftarrow \quad \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,MUPD,Dept,SupName,CatName}} \text{Temp1} \rtimes_{\text{Dept} = \text{Dept}} \text{Temp2}) && (34)
\end{aligned}$$

$\mathbf{DEL}_{column}(ProductDS, UnitPrice)$: UnitPrice is used in the expression that calculates the MUPD value in Eq. 33. First, UnitPrice is removed from the Input operation (Eq. 35). After that, we are presented with two options in repairing the ETL process. The first option is to replace UnitPrice with another column in the calculation of MUPD and the second option is to store a NULL value in MUPD. The first option requires the input of the user since the user knows best how the expression must be updated. In this paper, we focus on the basic modifications of the ETL process to keep it running without errors and without the user's input therefore we apply the second option that is to store a NULL value in MUPD (Eq. 37).

$$\begin{aligned}
\text{Temp0} \quad &\leftarrow \quad \mathcal{I}_{\text{ProdID,ProdName,UnitPrice,Dept,SupName,CatName}}(\text{Categories} \bowtie_{\text{CatID} = \text{CatID}} \text{ProductDS} && (35)\\
&\quad \bowtie_{\text{SupID} = \text{SupID}} \text{Supplier})\\
\text{Temp1} \quad &\leftarrow \quad \rho_{\text{ProdID} \leftarrow \text{ProdNo}}(\text{Temp0}) && (36)\\
\text{Temp2} \quad &\leftarrow \quad \mathcal{A}_{\text{Dept}\mid\text{MUPD}=\text{NULL}}(\text{Temp1}) && (37)\\
\text{ProductDW} \quad &\leftarrow \quad \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,MUPD,Dept,SupName,CatName}} \text{Temp1} \rtimes_{\text{Dept} = \text{Dept}} \text{Temp2}) && (38)
\end{aligned}$$

$\mathbf{DEL}_{column}(ProductDS, Dept)$: column Dept is the only column in the Partition or Group By part of the Aggregate operation in Eq. 37. The ETL process is repaired by removing the Aggregate operation (Eq. 37), and also column Dept from the Input (Eq. 39) and Insert (Eq. 41) operations. As a result, column MUPD is removed from Eq. 38. The left join to Temp2 in Eq. 38 is removed, since the joining column Dept no longer exist (Eq. 41). Recall that all attributes of the ProductDW table except ProdKey, ProdNo, and ProdName are nullable columns. This means that the ETL process will not fail when columns MUPD and Dept are deleted from it.

$$\text{Temp0} \leftarrow \mathcal{I}_{\text{ProdID,ProdName,UnitPrice,SupName,CatName}}(\text{Categories} \bowtie_{\text{CatID = CatID}} \text{ProductDS} \bowtie_{\text{SupID = SupID}} \text{Supplier}) \quad (39)$$

$$\text{Temp1} \leftarrow \rho_{\text{ProdID} \leftarrow \text{ProdNo}}(\text{Temp0}) \quad (40)$$

$$\text{ProductDW} \leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,SupName,CatName}} \text{Temp1}) \quad (41)$$

DEL$_{column}(ProductDS, CatID)$: CatID joins tables ProductDS and Categories. Hence, when deleted, all column(s) related to the Categories table (CatName) is/are removed from the ETL process. To repair the ETL process, column CatName and table Categories are removed from the Input operation in Eq. 42. CatName is removed from the Insert operation in Eq. 44.

$$\text{Temp0} \leftarrow \mathcal{I}_{\text{ProdID,ProdName,UnitPrice,SupName}}(\text{ProductDS} \bowtie_{\text{SupID = SupID}} \text{Supplier}) \quad (42)$$

$$\text{Temp1} \leftarrow \rho_{\text{ProdID} \leftarrow \text{ProdNo}}(\text{Temp0}) \quad (43)$$

$$\text{ProductDW} \leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,SupName}} \text{Temp1}) \quad (44)$$

ADD$_{column}(ProductDS, BatchNo, getBatchNo(ProdID, SupName))$: this SMO adds column BatchNo to the ProductDS DS table and calculates the value from a function (getBatchNo) based on available information (ProdID and SupName). BatchNo does not exist in DW table ProductDW. Recall that in the first assumption, we mentioned that we do not change the DW schema. Since the addition of BatchNo does not affect the ETL process already in place, no changes are made.

UNITE$_{column}(ProductDS, Supplier, Products, SupID = SupID)$: this SMO is the inverse of Split Column. It joins two tables vertically based on a given condition and removes the original tables. ProductDS is joined to Supplier based on equal names (SupID). After uniting the columns of ProductDS and Supplier tables into a new table Products, the only ETL repair needed is to replace the joined tables with new table Products (Eq. 45)

$$\text{Temp0} \leftarrow \mathcal{I}_{\text{ProdID,ProdName,UnitPrice,SupName}}(\text{Products}) \quad (45)$$

$$\text{Temp1} \leftarrow \rho_{\text{ProdID} \leftarrow \text{ProdNo}}(\text{Temp0}) \quad (46)$$

$$\text{ProductDW} \leftarrow \text{ProductDW} \cup (\pi_{\text{ProdNo,ProdName,SupName}} \text{Temp1}) \quad (47)$$

In the next section, we show how the repair methods discussed above are applied to generate rules for repairing ETL tasks/operations after SMOs are applied to the DS schema.

6. ETL Process Repair by Means of Rules

Fig. 3 visualizes our method of automatically repairing ETL processes modeled with Extended RA or SQL. A central component is an Application Programming Interface (API), which applies rules from a Rule Engine to repair an erroneous ETL process.

The Rule Engine selects the best rules for repairing an ETL process from a list of rules stored in a database table.

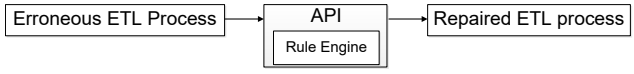

Figure 3. A schematic view on the ETL process repair using the API

Each rule is expressed in a general rule format. The general rule is a decision rule with a simple IF-THEN statement. A single rule or a combination of several rules can be used to make predictions on how the ETL process is to be repaired. The IF part evaluates an SMO in Fig. 4 which if applied at the DS on a Scope (table, column), will require the THEN part to predict a Required Action to be applied to an ETL Task provided a given Condition is true. If no condition has to be evaluated, then the Condition part is omitted from the rule.

General Rule: **IF** [**SMO**(Scope)] **THEN** [**ETL Task**(Required Action | Condition)]

Below, we discuss some SMOs that may occur at the DS and possible rules for repairing a variety of ETL tasks to keep the ETL process from failure. While specifying rules in this section, we refer to the Extended RA operations of Tab. 1. We specify the input operation (applied in Eq. 1) as ETL-Task-Extract, the rename operation (applied in Eq. 2) as ETL-Task-Rename, the aggregate operation (applied in Eq. 3) as ETL-Task-Aggregate, and the insert operation (applied in Eq. 4) as ETL-Task-Insert.

6.1. Rename Column: REN(column)

Several rules can be generated to repair each ETL task in case a column is renamed at the source. The rules evaluate the same IF condition which is a rename of a column at the DS. Below we describe rules for repairing some ETL Tasks after the **REN**(column) SMO is applied at the DS.

- **ETL-Task-Extract**: The action required to repair an ETL-Task-Extract if a column is renamed at the DS depends on whether the renamed column is included in the columns extracted in the ETL process.

  1. If the renamed column is not included in the extracted columns, no change to the ETL-Task-Extract is required. For example, if the ProductDS DS table contains a column called BatchNo which is renamed to Batch, then the ETL-Task-Extract of Eq. 1 will not change. This rule is specified as:
     IF [**REN**(column)] **THEN** [**ETL-Task-Extract**
     (No Change | IsRenamedColumnIncluded:No)]
  2. If the renamed column at the DS is included in the extracted columns of the ETL process, then that column is renamed in the ETL-Task-Extract. For example, If ProdID is renamed to ProdNo in the ProductDS DS table,

then ProdID of Eq. 1 is renamed to ProdNo. The rule below is applied to rename that column in the ETL-Task-Extract.
IF [**REN**(column)] **THEN** [**ETL-Task-Extract**
(Rename Column in ETL-Task-Extract | IsRenamedColumnIncluded:Yes)]
This rule can be applied several times if there are more than one renamed columns at the DS.

- **ETL-Task-Rename**: The action required to repair an ETL-Task-Rename if a column is renamed at the DS may differ under certain conditions. The required action of the THEN part depends on the condition specified.

  If the renamed column at the DS is not one of the columns renamed by the ETL-Task-Rename, no repair of the ETL-Task-Rename is required. For example, if the ProductDS DS table contains a column called BatchNo which is renamed to Batch, then the ETL-Task-Rename of Eq. 2 will not change. The rule is specified as:
  IF [**REN**(column)] **THEN** [**ETL-Task-Rename**
  (No Change | IsRenamedColumnIncluded:No)]

  If the column renamed at the DS is included in the columns renamed by the ETL-Task-Rename, then one of the two situations below may apply.

  1. The first situation occurs if the new column name in the ETL-Task-Rename is the same as the name at the DS after the **REN**(column) SMO is applied. In other words, the column at the DS has been given the exact name it has been renamed into by the ETL-Task-Rename. For example, in Eq. 2, ProdDesc is renamed to Description by the ETL-Task-Rename. If ProdDesc is renamed to Description in the ProductDS DS table, then one of the repair rules below will apply.
     - **IF** [**REN**(column)] **THEN** [**ETL-Task-Rename**
       (Delete ETL-Task-Rename | IsOnlyRenamedColumn:Yes,
       IsNewColumnNameInDSandETLTaskSame:Yes)]
       This rule applies if the column in question is the only column renamed by the ETL-Task-Rename and the new name of the column in both the DS table and the ETL-Task-Rename are the same. Let us assume that the ETL-Task-Rename of Eq. 2 only renames ProdDesc to Description. If ProdDesc is renamed to Description in the ProductDS DS table, then the column renaming of Eq. 2 is not needed anymore. Hence, the Required Action is to delete the ETL-Task-Rename.
     - **IF** [**REN**(column)] **THEN** [**ETL-Task-Rename**
       (Remove column from ETL-Task-Rename | IsOnlyRenamedColumn:No,
       IsNewColumnNameInDSandETLTaskSame:Yes)]
       This rule applies if the column in question is not the only column renamed by the ETL-Task-Rename and the new name of the column in both the DS table and the ETL-Task-Rename are the same. This situation is seen in Eq. 2 where two columns are renamed by the Rename ETL Task but only one, ProdDesc has been renamed at the DS

to Description. In this case the ETL-Task-Rename is not deleted but only ProdDesc is removed from the list of renamed columns.

2. The second situation occurs if the new column name in the ETL-Task-Rename is different from the name at the DS after the **REN**(column) SMO is applied. For example, if ProdDesc is renamed to ProdDescription after applying the **REN**(column) SMO at the DS, Eq. 2 becomes Eq. 48.

$$\text{Temp1} \leftarrow \rho_{\text{ProdID}\leftarrow\text{ProdNo},\text{ProdDescription}\leftarrow\text{Description}}(\text{Temp0}) \tag{48}$$

Hence, ETL-Task-Rename is repaired by replacing the old column name ProdDesc with the new name ProdDescription. This repair rule is specified as:
IF [**REN**(column)] **THEN** [**ETL-Task-Rename**
(Rename Column in ETL-Task-Rename | IsOnlyRenamedColumn:No
IsNewColumnNameInDSandETLTaskSame:No)]

- **ETL-Task-Aggregate**: The action required to repair an ETL-Task-Aggregate if a column is renamed at the DS depends on whether the renamed column is used in the Expression part or GROUP BY part of the ETL-Task-Aggregate

1. If the renamed column at the DS is neither included in the Expression nor GROUP BY part of the ETL-Task-Aggregate, no repair of the ETL-Task-Aggregate is required. For example, if the column ProdDesc is renamed to Description in the ProductDS table, the ETL-Task-Aggregate of Eq. 3 will not change. The rule is specified as:
IF [**REN**(column)] **THEN** [**ETL-Task-Aggregate**
(No Change | InExpressionOrGroupBy:No)]
2. If the renamed column at the DS is included in either the expression or GROUP BY part of the ETL-Task-Aggregate, then a repair of the ETL-Task-Aggregate is required. For example, if the column Dept used in the GROUP BY part of the ETL-Task-Aggregate of Eq. 3 is renamed to Department in the ProductDS table, Eq. 3 will become Eq. 49.

$$\text{Temp2} \leftarrow \mathcal{A}_{\text{Department}|\text{MUPD}=\text{MAX}(\text{UnitPrice})}(\text{Temp1}) \tag{49}$$

Alternatively, if UnitPrice which is used in the expression part of the ETL-Task-Aggregate of Eq. 3 is renamed to UP, then Eq. 3 will become Eq. 50.

$$\text{Temp2} \leftarrow \mathcal{A}_{\text{Dept}|\text{MUPD}=\text{MAX}(\text{UP})}(\text{Temp1}) \tag{50}$$

The rule specified for both situations is:
IF [**REN**(column)] **THEN** [**ETL-Task-Aggregate**
(Rename Column in ETL-Task-Aggregate | InExpressionOrGroupBy:Yes)]

- **ETL-Task-Insert**: If a column is renamed at the DS, an ETL-Task-Insert is repaired depending on whether the renamed column is included in the columns inserted into the DW table or not.

  1. If the renamed column is not included in the list of columns inserted into the DW table, no repair of ETL-Task-Insert is required. For example, if the ProductDS DS table contains a column called BatchNo, which is renamed to Batch, then the ETL-Task-Insert of Eq. 4 will not change. The rule is specified as:
     IF [**REN**(column)] **THEN** [**ETL-Task-Insert**
     (No Change | IsRenamedColumnIncluded:No)]
  2. If the renamed column is included in the list of columns inserted into the DW table, then the ETL-Task-Insert has to be repaired. It is possible that a column renamed in the DS is renamed a second time by an ETL-Task-Rename in the ETL process. For this reason, replacing the column name in the ETL-Task-Insert with the new name at the DS may not always provide the correct solution. One of the two situations below may be applicable.
     - If a renamed column is not renamed a second time by an ETL-Task-Rename in the ETL process, then an ETL-Task-Insert is repaired by replacing the column name with the new name at the DS. Referring to the running example, if SupName is renamed to Supplier in the ProductDS DS table, then SupName is replaced by Supplier in Eq. 4. This rule is specified as:
       IF [**REN**(column)] **THEN** [**ETL-Task-Insert**
       (Rename Column in ETL-Task-Insert |
       ColumnIsRenamedInETLProcess:No)]

     - If a renamed column is renamed a second time by an ETL-Task-Rename in the ETL process, then an ETL-Task-Insert requires no change. For example, if ProdID is renamed to ProductID in ProductDS DS table, then Eq. 4 requires no repair since it already has the column name given by the ETL-Task-Rename (ProdNo). This rule is specified as:
       IF [**REN**(column)] **THEN** [**ETL-Task-Insert**
       (No Change | ColumnIsRenamedInETLProcess:Yes)]

6.2. Add Column: ADD(column)

Adding a new column to a DS table does not affect an already deployed ETL process. For this reason, we do not provide any rules corresponding to the actions to be taken to repair ETL tasks when a column is added to the DS. Based on new business requirements, an ETL developer may provide steps on how a new column should be included in the ETL pipeline. Implementing those steps is out of scope of this paper since it is not considered a repair of an erroneous ETL process.

6.3. Delete Column: DEL(column)

Here we describe some rules and corresponding actions that can be taken to repair four ETL tasks (ETL-Task-Extract, ETL-Task-Rename, ETL-Task-Aggregate, ETL-Task-Insert) when a **DEL**(column) SMO is applied on a DS table.

- **ETL-Task-Extract**: The action required to repair an ETL-Task-Extract if a column is deleted at the DS depends on whether the deleted column is included in the columns extracted in the ETL process.
  - If the deleted column is not included in the extracted columns, no change to the ETL-Task-Extract is required. For example, if the ProductDS DS table contains a column called BatchNo which is deleted, then the ETL-Task-Extract of Eq. 1 will not change. This rule is specified as:
    IF [**DEL**(column)] **THEN** [**ETL-Task-Extract**
    (No Change | IsDeletedColumnIncluded:No)]
  - If the deleted column at the DS is included in the extracted columns of the ETL process, then that column is deleted in the ETL-Task-Extract. For example, If ProdID is deleted in the ProductDS DS table, then ProdID of Eq. 1 is removed. The rule applied is:
    IF [**DEL**(column)] **THEN** [**ETL-Task-Extract**
    (Delete Column in ETL-Task-Extract | IsDeletedColumnIncluded:Yes)]
    This rule can be applied several times if there are more than one deleted columns at the DS.
- **ETL-Task-Rename**: The action required to repair an ETL-Task-Rename if a column is deleted at the DS may differ under certain conditions. The required action of the THEN part depends on the condition specified. If the deleted column at the DS is not one of the columns renamed by the ETL-Task-Rename, no repair of the ETL-Task-Rename is required. For example, if the ProductDS DS table contains a column called BatchNo which is deleted, then the ETL-Task-Rename of Eq. 2 will not change. The rule is specified as:
  IF [**DEL**(column)] **THEN** [**ETL-Task-Rename**
  (No Change | IsDeletedColumnIncluded:No)]

  If a column renamed by an ETL-Task-Rename is deleted at the DS, then one of the two situations below may apply.

  1. **IF** [**DEL**(column)] **THEN** [**ETL-Task-Rename**
     (Delete ETL-Task-Rename | IsOnlyRenamedColumn:Yes)]
     This rule applies if the column in question is the only column renamed by the ETL-Task-Rename. Let us assume that the ETL-Task-Rename of Eq. 2 only renames ProdDesc to Description. If ProdDesc is deleted from the ProductDS DS table, then the column renaming of Eq. 2 is not needed anymore. Hence, the Required Action is to delete the ETL-Task-Rename.
  2. **IF** [**DEL**(column)] **THEN** [**ETL-Task-Rename**
     (Remove column from ETL-Task-Rename | IsOnlyRenamedColumn:No)]

This rule only applies if the column in question is not the only column renamed by the ETL-Task-Rename. This situation is seen in Eq. 2 where two columns are renamed by the Rename ETL Task but only one, ProdDesc is deleted at the DS. In this case the ETL-Task-Rename is not deleted but only ProdDesc is removed from the list of renamed columns.

- **ETL-Task-Aggregate**: For simplicity, in this paper we focus on ETL-Task-Aggregates that have only one aggregate function like the one in Eq. 3. The rules to be applied to repair an ETL-Task-Aggregate with multiple aggregate functions may differ from the one explained here.

  The rule required to repair an ETL-Task-Aggregate with one aggregate function if a column is deleted at the DS depends on whether the deleted column is used in the Expression or GROUP BY part of the ETL-Task-Aggregate

  If the deleted column at the DS is neither included in the Expression nor GROUP BY part of the ETL-Task-Aggregate, no repair of the ETL-Task-Aggregate is required. For example, if the column ProdDesc is deleted from the ProductDS table, the ETL-Task-Aggregate of Eq. 3 will not change.

  The rule is specified as:
  IF [**DEL**(column)] **THEN** [**ETL-Task-Aggregate**
  (No Change | InExpressionOrGroupBy:No)]

  If the deleted column at the DS is included in either the Expression or GROUP BY part of the ETL-Task-Aggregate, then one of three options may be applied to repair of the ETL-Task-Aggregate.

  1. If the deleted column is the only column in the GROUP BY part of the ETL-Task-Aggregate, then the ETL-Task-Aggregate can be removed from the ETL process. For example, if the column Dept which happens to be the only column in the GROUP BY part of Eq. 3 is deleted then Eq. 3 can be removed from the ETL process.

     The rule is specified as:
     IF [**DEL**(column)] **THEN** [**ETL-Task-Aggregate**
     (Remove ETL-Task-Aggregate | IsOnlyGroupByColumn:Yes)]

  2. If the deleted column is not the only column in the GROUP BY part of the ETL-Task-Aggregate, then the deleted column can be removed from the ETL-Task-Aggregate. For example, if column Dept is deleted and it happens not to be the only column in the GROUP BY part of Eq. 3, then Dept can be removed from Eq. 3.

     The rule is specified as:
     IF [**DEL**(column)] **THEN** [**ETL-Task-Aggregate**
     (Remove column from ETL-Task-Aggregate | IsOnlyGroupByColumn:No)]

  3. If the deleted column is found in the expression part of the ETL-Task-Aggregate, then the computed value will be incorrect. We store a NULL

value in the computed column to repair the ETL-Task-Aggregate. For example, if the column UnitPrice is deleted from the ProductDS table, Eq. 3 becomes Eq. 51.

$$\text{Temp2} \leftarrow \mathcal{A}_{\text{Dept}|\text{MUPD=NULL}}(\text{Temp1}) \tag{51}$$

The rule is specified as:
IF [**DEL**(column)] **THEN** [**ETL-Task-Aggregate**
(Expression = NULL in ETL-Task-Aggregate | InExpressionPart:Yes)]

Inserting a NULL value in a computed column may not provide the expected data in the corresponding DW table but this will keep the entire ETL process from failing. The user may have to rely on data quality checks to identify and provide a preferred solution.

- **ETL-Task-Insert**: Repairing an ETL-Task-Insert after a column is deleted at the DS depends on whether the deleted column is included in the ETL process or not.

  1. If the deleted column is not included in the ETL process, no change to ETL-Task-Insert is required. For example, if the ProductDS DS table contains column called BatchNo which is deleted, then the ETL-Task-Insert of Eq. 4 will not change. This rule is specified as:
  IF [**DEL**(column)] **THEN** [**ETL-Task-Insert**
  (No Change | IsDeletedColumnIncluded:No)]
  2. If the deleted column at the DS is included in the ETL process, then that column is removed from ETL-Task-Insert on condition that the corresponding column in the DW table is nullable. Notice that the deleted column may have been renamed in the ETL process by an ETL-Task-Rename. In that case, the name given by the ETL-Task-Rename is used. For example, if ProdID is deleted in the ProductDS DS table, then ProdNo is removed from Eq. 4. The rule applied is:
  IF [**DEL**(column)] **THEN** [**ETL-Task-Insert**
  (Delete Column in ETL-Task-Insert |
  IsDeletedColumnIncluded:Yes,
  IsCorrespondingDWHColumnNullable:Yes)]
  This rule can be applied multiple times if there are more than one deleted columns at the DS.
  3. If the deleted column at the DS is included in the ETL process and the corresponding column in the DW table is not nullable, then that column in the ETL-Task-Insert is replaced by a default value considering the data type of the column in the DW table. A default value for each data type is set in advance. For example, if ProdName is deleted from the ProductDS DS table, then ProdName of Eq. 4 is replaced by an empty string (‘ ’). Recall that ProdName of the ProductDW table is not nullable. The rule

applied is:
IF [**DEL**(column)] **THEN** [**ETL-Task-Insert**
(Replace Column with a default value | IsDeletedColumnIncluded:Yes,
IsCorrespondingDWHColumnNullable:No)]

6.4. Split Column: SPLIT(column)

This SMO Vertically spits a table into two tables overlapping on a common column(s), which are used to reconstruct the original table. The only ETL task that requires repair after a SPLIT(column) is applied on a DS table is the ETL-Task-Extract. Tables resulting from the split are joined in the ETL-Task-Extract to provide columns as they were before the split. No change is required for all other ETL tasks in the ETL process since they receive same columns from the repaired ETL-Task-Extract.

1. **ETL-Task-Extract**: After a SPLIT(column) has been applied to a DS, ETL-Task-Extract is repaired by joining the resulting tables on the overlapping column(s). For example, If the Products table of Eq. 19 is split into ProductDS and Supplier, then ProductDS and Supplier are joined on the overlapping column SupID (Eq. 23). This rule is specified as:
   IF [**SPLIT**(column)] **THEN** [**ETL-Task-Extract**
   (Join tables on the overlapping column(s))]

   For all other ETL tasks, no change is required after a SPLIT(column) has been applied to a DS. The repaired ETL-Task-Extract passes down the same colunms as before through the ETL pipeline.

2. **ETL-Task-Rename**: For this ETL task, no change is required after a SPLIT(column) SMO has been applied to the DS. The rule is specified as:
   IF [**SPLIT**(column)] **THEN** [**ETL-Task-Rename**
   (No Change)]

3. **ETL-Task-Aggregate**: For this ETL task, no change is required after a SPLIT(column) SMO has been applied to the DS. The rule is specified as:
   IF [**SPLIT**(column)] **THEN** [**ETL-Task-Aggregate**
   (No Change)]

4. **ETL-Task-Insert**: For this ETL task, no change is required after a SPLIT(column) SMO has been applied to the DS. The rule is specified as:
   IF [**SPLIT**(column)] **THEN** [**ETL-Task-Insert**
   (No Change)]

6.5. Unite Column: UNITE(column)

This SMO joins two tables vertically based on a given condition and removes the original tables. As in the case of the SPLIT(column) SMO, ETL-Task-Extract is the

only ETL task that needs to be repaired after UNITE(column) is applied to the DS tables.

1. **ETL-Task-Extract**: After a UNITE(column) SMO is applied to a DS, an ETL-Task-Extract is repaired by replacing the original table(s) with the resulting table.

   An ETL-Task-Extract can extract data from a single DS table or joined DS tables. In the case of a single table, if it is united with another DS table, the ETL-Task-Extract is repaired by replacing it with the table resulting from the UNITE(column) SMO. In the case of joined tables, if a UNITE(column) SMO applied at the DS joins them into a single resulting table, the ETL-Task-Extract is repaired by replacing the joined tables with the resulting table. For example, if the ProductDS and Supplier tables of Eq. 42 are united into Products, then ProductDS and Supplier will be replaced by Products in Eq. 45. This rule is specified as:
   IF [**UNITE**(column)] **THEN** [**ETL-Task-Extract**
   (Replace single or joined tables with resulting table in ETL-Task-Extract)]

   For all other ETL tasks, no change is required after a UNITE(column) has been applied to a DS. The repaired ETL-Task-Extract passes down the same colunms as before through the ETL pipeline.

2. **ETL-Task-Rename**: For this ETL task, no change is required after a UNITE(column) SMO has been applied to the DS. The rule is specified as:
   IF [**UNITE**(column)] **THEN** [**ETL-Task-Rename**
   (No Change)]

3. **ETL-Task-Aggregate**: For this ETL task, no change is required after a UNITE(column) SMO has been applied to the DS. The rule is specified as:
   IF [**UNITE**(column)] **THEN** [**ETL-Task-Aggregate**
   (No Change)]

4. **ETL-Task-Insert**: For this ETL task, no change is required after a UNITE(column) SMO has been applied to the DS. The rule is specified as:
   IF [**UNITE**(column)] **THEN** [**ETL-Task-Insert**
   (No Change)]

7. Overview of Rule Generation

Below we provide a summary to map ETL Tasks to repair actions as described in Sec. 6. Afterward, we deduce some situations in which certain ETL tasks do not require repair after particular SMOs are applied at the DS.

SMO: **ADD**(column)

ETL Task: **All ETL tasks**

Repair Action: No Change
Condition:

SMO: **REN**(column)

ETL Task: **ETL-Task-Extract**
Repair Action: No Change
Condition: IsRenamedColumnIncluded:No

ETL Task: **ETL-Task-Extract**
Repair Action: Rename Column in ETL-Task-Extract
Condition: IsRenamedColumnIncluded:Yes

ETL Task: **ETL-Task-Rename**
Repair Action: No Change
Condition: IsRenamedColumnIncluded:No

ETL Task: **ETL-Task-Rename**
Repair Action: Delete ETL-Task-Rename
Condition: IsOnlyRenamedColumn:Yes,
IsNewColumnNameInDSandETLTaskSame:Yes

ETL Task: **ETL-Task-Rename**
Repair Action: Remove column from ETL-Task-Rename
Condition: IsOnlyRenamedColumn:No,
IsNewColumnNameInDSandETLTaskSame:Yes

ETL Task: **ETL-Task-Rename**
Repair Action: Rename Column in ETL-Task-Rename
Condition: IsOnlyRenamedColumn:No,
IsNewColumnNameInDSandETLTaskSame:No

ETL Task: **ETL-Task-Aggregate**
Repair Action: No Change
Condition: InExpressionOrGroupBy:No

ETL Task: **ETL-Task-Aggregate**
Repair Action: Rename Column in ETL-Task-Aggregate
Condition: InExpressionOrGroupBy:Yes

ETL Task: **ETL-Task-Insert**
Repair Action: No Change
Condition: IsRenamedColumnIncluded:No

ETL Task: **ETL-Task-Insert**
Repair Action: Rename Column in ETL-Task-Insert
Condition: ColumnIsRenamedInETLProcess:No

ETL Task: **ETL-Task-Insert**
Repair Action: No Change
Condition: ColumnIsRenamedInETLProcess:Yes

SMO: **DEL**(column)

ETL Task: **ETL-Task-Extract**
Repair Action: No Change
Condition: IsDeletedColumnIncluded:No

ETL Task: **ETL-Task-Extract**
Repair Action: Delete Column in ETL-Task-Extract
Condition: IsDeletedColumnIncluded:Yes

ETL Task: **ETL-Task-Rename**
Repair Action: No Change
Condition: IsDeletedColumnIncluded:No

ETL Task: **ETL-Task-Rename**
Repair Action: Delete ETL-Task-Rename
Condition: IsOnlyRenamedColumn:Yes

ETL Task: **ETL-Task-Rename**
Repair Action: Remove column from ETL-Task-Rename
Condition: IsOnlyRenamedColumn:No

ETL Task: **ETL-Task-Aggregate**
Repair Action: No Change
Condition: InExpressionOrGroupBy:No

ETL Task: **ETL-Task-Aggregate**
Repair Action: Remove ETL-Task-Aggregate
Condition: IsOnlyGroupByColumn:Yes

ETL Task: **ETL-Task-Aggregate**
Repair Action: Remove column from ETL-Task-Aggregate
Condition: IsOnlyGroupByColumn:No

ETL Task: **ETL-Task-Aggregate**
Repair Action: Expression = NULL in ETL-Task-Aggregate

Condition: InExpressionPart:Yes

ETL Task: **ETL-Task-Insert**
Repair Action: No Change
Condition: IsDeletedColumnIncluded:No

ETL Task: **ETL-Task-Insert**
Repair Action: Delete Column in ETL-Task-Insert
Condition: IsDeletedColumnIncluded:Yes,
IsCorrespondingDWHColumnNullable:Yes

ETL Task: **ETL-Task-Insert**
Repair Action: Replace Column with a default value
Condition: IsDeletedColumnIncluded:Yes,
IsCorrespondingDWHColumnNullable:No

SMO: **SPLIT**(column)

ETL Task: **ETL-Task-Extract**
Repair Action: Join tables on the overlapping column(s)
Condition:

ETL Task: **All Other ETL tasks**
Repair Action: No Change
Condition:

SMO: **UNITE**(column)

ETL Task: **ETL-Task-Extract**
Repair Action: Replace single or joined tables with resulting table in ETL-Task-Extract
Condition:

ETL Task: **All Other ETL tasks**
Repair Action: No Change
Condition:

Based on the summary above, we make the following deductions.

1. No change is required on any ETL task after an ADD(column) SMO is applied at the DS.
2. No change is required on any ETL task if a REN(column) or DEL(column) SMO is applied at the DS on a column that is not present in the ETL workflow.

3. Apart from ETL-Task-Extract, no change is required on any ETL task after a SPLIT(column) or UNITE(column) SMO is applied at the DS.

4. No change is required on an ETL-Task-Insert after a REN(column) SMO is applied at the DS if the renamed column is renamed a second time by an ETL-Task-Rename in the ETL process

8. Related Work

The topic of this paper is loosely related to handling schema changes in relational data warehouses. It is also strongly related to handling DS schema changes at the ETL layer.

8.1. Handling Schema Changes in Relational Data Warehouses

Schema evolution has been studied extensively in several directions, e.g., [15, 17, 43, 51, 56, 53, 55, 60]. The approaches in this category can be categorized as follows: (1) *schema and data evolution*, e.g. [10, 31, 33], (2) *simulation*, e.g. [8, 9], (3) *temporal extensions* [14, 19, 20, 47], and (4) *versioning* [11, 23, 27, 28, 37, 44, 49].

Schema and data evolution approaches maintain one DW schema and one set of data that evolve in time. Schema modifications require data conversions from a previous state to a new state, in order to assure the consistency of data with respect to their schema. *Simulation* approaches use virtual data structures like views. Such structures are applied to simulating or to screening a DW schema evolution. *Temporal extensions* use timestamps on modified data in order to create temporal versions. In *versioning* techniques, depending on the approach, a DW evolution is managed partially by means of schema versions and partially by data versions. A more detailed description of these approaches can be found in [59].

8.2. Handling DS Schema Changes at the ETL Layer

In [38] the authors presented PRIMA - a system for rewriting queries for the appropriate versions of the evolved schema. PRIMA uses a method for grouping and publishing the history of a relational database in XML. This temporally grouped representation makes it easy to formulate sophisticated historical queries on any given schema version using standard XQuery. This enables a user to write queries against the current schema while retrieving the data from one or more schema versions. The system then rewrites such queries into equivalent ones for the appropriate versions of the schema.

Solutions for handling DS schema changes by ETL process include: Hecataeus [39], MAIME [13], E-ETL [57], and E3TL [3, 6]. The first two approaches propose to manually annotate ETL processes with repair rules before a DS schema evolution

occurs. This way, ETL processes can be repaired by applying the rules. E-ETL and E3TL apply case-based-reasoning (CBR) to find and apply the most similar case to a given repair problem at hand. CBR means understanding and solving a new problem by remembering how a similar problem was solved in the past.

Hecataeus [39] applies rules to predict the impact of a DS change on an ETL process. An ETL process is modeled as a graph. Nodes of the graph represent attributes, relations, views, functions, constraints, and ETL tasks. A user defines rules on each node to specify actions that will be performed on the node when a schema change occurs that impacts the node. A user examines the impact of an event on the graph by identifying affected parts of the graph in response to a schema change event.

MAIME [13] builds also on the concept of Hecataeus. An altering algorithm configures a graph representing an ETL process to block, propagate, or manually repair an ETL process. MAIME handles a limited number of DS changes and ETL operations. DS changes applicable in MAIME are limited to adding, renaming, deleting of a property, and changing its data type. The set of ETL operations that MAIME handles is limited to data input, data insert, aggregate, split, data conversion, derived column, lookup, sort, union all. Rules defining how to handle schema changes must be explicitly defined for each graph element in advance, like in Hecataeus.

The E-ETL framework [57] applies CBR to repairing ETL processes. E-ETL requires: (1) a structure, called the Library of Repair Cases and (2) a knowledge base that is constantly augmented with cases (i.e., a DS change and associated with it the ETL reparation applied). When there is a new DS change, the Case Detection Algorithm selects cases that best match the problem. The solution of the selected case is applied to the new DS change problem to repair an ETL process.

The E3TL framework [3, 6] combines CBR with repair rules stored in a rule base to repair an ETL processes. A DS schema change and the user's reparation are stored together as a case in a knowledge base called the case base. Periodically, cases from the case base are translated into rules. These inferred rules are stored in the rule base and are used to handle similar DS schema changes in the future. Both the E-ETL and E3TL frameworks require a knowledge base in advance, which may not be in place when a given ETL process was initially deployed. The knowledge base is built as the number of cases increases. For each DS schema change, a similar case should already exist in the knowledge base, to enable E-ETL and E3TL frameworks to find a suitable solution.

In [41], the authors provide ways to predict the maintenance effort on ETL workflows and explore techniques to assess the quality of ETL designs by monitoring the vulnerability of DS modules to future changes. They identified schema size and complexity of the DS as two important factors for the vulnerability of an ETL workflow. DS tables comprising many attributes in their schema are more likely to be altered and hence, more likely to affect the ETL workflow they feed.

The main goal of the Extended RA [5] approach to ETL process repair described in this paper is to keep ETL processes running without errors and to minimize user intervention during the ETL repair process. All modifications needed to repair an ETL process are defined as rules in a database. Unlike Hecataeus and MAIME, a

user does not need to define modification rules before running the ETL processes. The rules are developed in advance and implemented together with an API that interprets them. The API modifies the ETL process by applying these modification rules. If needed, the default rules can be modified by a user.

A platform-independent modeling of ETL orchestration tasks has been missing in the ETL tools available on the market. Recently, [42] created a new domain-specific language (DSL) called ETL Control Language (ETLCL) to facilitate the orchestration and maintenance of ETL processes at the level of platform-independent models (PIMs). ETLCL allows a user to create specifications for controlling data flow and loading data into DWs, regardless of platform or ETL tool.

8.3. ETL Process Recovery

Another research problem within ETL processing is a process recovery after its failure. Even though this problem is loosely related to the one we address, some works focus on efficient restarting or self-repairing a failed process. For example, [34] presents algorithm called *Design and Resume* that allows to resume an interrupted ETL process from the moment where it was interrupted. This algorithm was further extended in [24] in order to resume a parallel loading after failure. In order to reduce time overhead of restarting a failed process, check-points were proposed in [25]. In [30] the authors proposed a fault-tolerant distributed ELT engine, which based on error analysis from an execution log is able to run a recovery algorithm and in [48] the authors proposed a redo strategy for various fault scenarios of ETL processes.

All of the aforementioned approaches focus on ETL process failures as the result of a system malfunctioning, like the lack of resources (RAM, disk, CPU), operating system errors, hardware errors. None of the approaches takes into consideration an ETL process failure as the result of a DS schema change.

9. Conclusions and Future Work

Repairing ETL processes triggered by structural changes in DSs is a very interesting and difficult problem, even though known and researched for years already. For this reason, solutions supporting only simple DS changes have been proposed so far. Methods for repairing ETL processes automatically or at least semi-automatically are still to be developed.

In this paper, we contributed an alternative approach to automatically repairing ETL processes affected by a DS schema change. Our solution is based on repair rules that are managed by the Rule Engine of an API. The advantage of our approach, as compared to the ones outlined in Sec. 8 is that it does not require a knowledge base to be constructed in advance for a particular ETL process. Moreover, our approach utilizes Extended RA, which is a relational algebra language extended with update operations and can be translated to any relational query language like TSQL or PL/SQL. As a consequence, any ETL model that can be translated into Extended RA can be

repaired by the API.

Notice that in Section 5 we made a few assumptions, which allowed us to find the solution to the already difficult problem. Alleviating some of the assumptions will open new research paths in semi-automatic ETL repairing. Therefore, future works of our and other research groups can focus on the following issues:

1. applying machine learning algorithms (possibly large language models) to rule discovery and generation;
2. methods for automatic management of rules in the Rule Engine - this includes discovering redundant rules, rule visualization, scheduling their executions, and allowing multiple rules with priorities for each unit of ETL repair;
3. efficient storage and retrieval of rules;
4. managing multiple versions of repaired ETL processes;
5. developing techniques for semi-automatic propagation of schema modifications to a DW;
6. developing methods for repairing ETL processes with UDFs of open code;
7. developing methods for learning internal logic of black-box UDFs.
8. methods of quantifying the impact of a DS change on complex ETL workflows before a change occurs.

Moreover, another approach to ease repairing ETL processes, which could be considered, is to provide means for defining how DSs could evolve and which would be the impact of such evolution on ETL processes. This should be done during architectural design, as partially addressed in [32, 40].

Appendix

A. SQL Translation of Eqs. 1 to 4

In this section we show a translation of the Extended RA operations in the ETL process of Sec. 5 (Eq. 1 to Eq. 4) to SQL (PL/SQL) commands.

Command 1 creates temporary tables (Temp0, Temp1, Temp2) to hold data from one ETL task to another. Command 2 (equivalent to Eq. 1) extracts data from the ProductDS table into Temp0. Command 3 (equivalent to Eq. 2) renames two attributes ProdID and ProdDesc to ProdNo and Description, respectively, and the resulting relation is stored in Temp1. Command 4 (equivalent to Eq. 3) aggregates tuples of Temp1 by grouping them by attribute Dept and computing for each group a new attribute called MUPD, which is the maximum UnitPrice. The resulting relation is stored in Temp2. Command 5 (equivalent to Eq. 4) joins Temp1 with Temp2 on the Dept attribute and all the resulting tuples are inserted into the ProductDW table.

```
--Command 1: Drop all temporary tables
DROP TABLE IF EXISTS Temp0, Temp1, Temp2;

--Command 2: Input ETL Operation (Equivalent to equation 1)
CREATE TEMPORARY TABLE Temp0 AS
SELECT prodid, ProdName, ProdDesc, UnitPrice, Dept, SupName
FROM ProductDS;

--Command 3: Rename ETL Operation (Equivalent to equation 2)
CREATE TEMPORARY TABLE Temp1 AS
SELECT prodid AS ProdNo, ProdName, ProdDesc as Description,
UnitPrice, Dept, SupName,
FROM Temp0;

--Command 4: Aggregate ETL Operation (Equivalent to equation 3)
CREATE TEMPORARY TABLE Temp2 AS
SELECT max(UnitPrice) AS MUPD, Dept
FROM   Temp1
GROUP BY Dept;

--Command 5: Insert ETL Operation (Equivalent to equation 4)
INSERT INTO ProductDW (ProdNo,ProdName,Description,MUPD,Dept,SupName)
SELECT T1.ProdNo,T1.ProdName,T1.Description,T2.MUPD,T1.Dept,T1.SupName
FROM Temp1 T1 LEFT JOIN Temp2 T2 ON T1.Dept = T2.Dept
```

References

[1] Ali S. M. F. and Wrembel R. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. *International Journal on Very Large Data Bases (VLDB)*, 26(6):777–801, 2017.

[2] Allen M. and Cervo D. *Multi-domain master data management: Advanced MDM and data governance in practice*. Morgan Kaufmann, 2015.

[3] Awiti J. Algorithms and architecture for managing evolving ETL workflows. In *Proc. of ADBIS Workshops*, volume 1064 of *CCIS*, pages 539–545. Springer, 2019.

[4] Awiti J., Vaisman A. A., and Zimányi E. From conceptual to logical ETL design using BPMN and relational algebra. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, volume 11708 of *LNCS*, pages 299–309. Springer, 2019.

[5] Awiti J., Vaisman A. A., and Zimányi E. Design and implementation of ETL processes using BPMN and relational algebra. *Data & Knowledge Engineering (DKE)*, 129:101837, 2020.

[6] Awiti J. and Wrembel R. Rule discovery for (semi-)automatic repairs of ETL processes. In *International Baltic Conference on Databases and Information Systems (DB&IS)*, volume 1243 of *CCIS*, pages 250–264. Springer, 2020.

[7] Awiti J. and Zimányi E. An XML interchange format for ETL models. In *New Trends in Databases and Information Systems (ADBIS) Workshops*, volume 1064 of *CCIS*, pages 427–439. Springer, 2019.

[8] Balmin A., Papadimitriou T., and Papakonstantinou Y. Hypothetical queries in an OLAP environment. In *International Conference on Very Large Data Bases (VLDB)*, pages 220–231, 2000.

[9] Bellahsene Z. View adaptation in data warehousing systems. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 300–309. LNCS 1460, 1998.

[10] Blaschka M., Sapia C., and Hofling G. On schema evolution in multidimensional databases. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, pages 153–164. LNCS 1676, 1999.

[11] Body M., Miquel M., Bédard Y., and Tchounikine A. A multidimensional and multiversion structure for OLAP applications. In *International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 1–6, 2002.

[12] Bodziony M., Krzyzanowski H., Pieta L., and Wrembel R. On discovering semantics of user-defined functions in data processing workflows. In *International Workshop on Big Data in Emergent Distributed Environments (BiDEDE) @ ACM SIGMOD/PODS Conference*. ACM, 2021.

[13] Butkevicius D., Freiberger P. D., and Halberg F. M. Maime: a maintenance manager for ETL processes. In *Workshops @ EDBT/ICDT Joint Conference*, volume 1810 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[14] Chamoni P. and Stock S. Temporal structures in data warehousing. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, pages 353–358. LNCS 1676, 1999.

[15] Cleve A., Gobert M., Meurice L., Maes J., and Weber J. Understanding database schema evolution: A case study. *Science of Computer Programming*, 97:113–121, 2015.

[16] Curino C., Moon H. J., Tanca L., and Zaniolo C. Schema evolution in Wikipedia - toward a web information system benchmark. In *International Conference on Enterprise Information Systems (ICEIS)*, pages 323–332, 2008.

[17] Delplanque J., Etien A., Anquetil N., and Auverlot O. Relational database schema evolution: An industrial case study. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 635–644. IEEE, 2018.

[18] Dimolikas K., Zarras A. V., and Vassiliadis P. A study on the effect of a table's involvement in foreign keys to its schema evolution. In *International Conference on Conceptual Modeling ER*, volume 12400 of *LNCS*, pages 456–470. Springer, 2020.

[19] Eder J. and Koncilia C. Changes of dimension data in temporal data warehouses. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, pages 284–293. LNCS 2114, 2001.

[20] Eder J., Koncilia C., and Morzy T. The COMET metamodel for temporal data warehouses. In *International Conference on Advanced Information Systems Engineering (CAISE)*, pages 83–99. LNCS 2348, 2002.

[21] Elmasri R. and Navathe S. B. *Fundamentals of Database Systems, 7th Edition.* Pearson, 2016.

[22] Giachos F., Pantelidis N., Batsilas C., Zarras A. V., and Vassiliadis P. Parallel lives diagrams for co-evolving communities and their application to schema evolution. In *Companion Proceedings of the International Conference on Conceptual Modeling: ER Forum*, volume 3618 of *CEUR Workshop Proceedings.* CEUR-WS.org, 2023.

[23] Golfarelli M., Lechtenbörger J., Rizzi S., and Vossen G. Schema versioning in data warehouses. In *ER 2004 Workshops*, pages 415–428. LNCS 3289, 2004.

[24] Gorawski M. and Marks P. Resumption of data extraction process in parallel data warehouses. In *International Conference Parallel Processing and Applied Mathematics (PPAM)*, volume 3911 of *LNCS*, pages 478–485. Springer, 2005.

[25] Gorawski M. and Marks P. Checkpoint-based resumption in data warehouses. In *Software Engineering Techniques: Design for Quality (SET)*, volume 227 of *IFIP*, pages 313–323. Springer, 2006.

[26] Hai R., Koutras C., Quix C., and Jarke M. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12571–12590, 2023.

[27] Herrmann K., Voigt H., Behrend A., Rausch J., and Lehner W. Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language. In *ACM International Conference on Management of Data (SIGMOD)*, pages 1101–1116. ACM, 2017.

[28] Herrmann K., Voigt H., Pedersen T. B., and Lehner W. Multi-schema-version data management: data independence in the twenty-first century. *International Journal on Very Large Data Bases (VLDB)*, 27:547–571, 2018.

[29] Herrmann K., Voigt H., Rausch J., Behrend A., and Lehner W. Robust and simple database evolution. *Information Systems Frontiers*, 20:45–61, 2018.

[30] Huang J. and Guo C. An mas-based and fault-tolerant distributed ETL workflow engine. In *IEEE International Conference on Computer Supported Cooperative (CSCWD)*, pages 54–58. IEEE, 2012.

[31] Hurtado C. A., Mendelzon A. O., and Vaisman A. A. Maintaining data cubes under dimension updates. In *International Conference on Data Engineering (ICDE)*, pages 346–355. IEEE, 1999.

[32] Hyun S. and Hurtado J. A. Traceability of architectural design decisions and software artifacts: A systematic mapping study. *Foundations of Computing and Decision Sciences (FCDS)*, 48(4):401–423, 2023.

[33] Kaas C. K., Pedersen T. B., and Rasmussen B. D. Schema evolution for stars and snowflakes. In *International Conference on Enterprise Information Systems (ICEIS)*, pages 425–433, 2004.

[34] Labio W., Wiener J. L., Garcia-Molina H., and Gorelik V. Efficient resumption of interrupted warehouse loads. In *ACM SIGMOD International Conference on Management of Data*, pages 46–57. ACM, 2000.

[35] Manousis P., Vassiliadis P., and Papastefanatos G. Automating the adaptation of evolving data-intensive ecosystems. In *International Conference on Conceptual Modeling (ER)*, volume 8217 of *LNCS*, pages 182–196, 2013.

[36] Manousis P., Vassiliadis P., and Papastefanatos G. Impact analysis and policy-conforming rewriting of evolving data-intensive ecosystems. *Journal on Data Semantics*, 4(4):231–267, 2015.

[37] Mendelzon A. O. and Vaisman A. A. Temporal queries in OLAP. In *International Conference on Very Large Data Bases (VLDB)*, pages 242–253, 2000.

[38] Moon H. J., Curino C., Deutsch A., Hou C., and Zaniolo C. Managing and querying transaction-time databases under schema evolution. *Proceedings of the VLDB Endowment*, 1(1):882–895, 2008.

[39] Papastefanatos G., Vassiliadis P., Simitsis A., Sellis T., and Vassiliou Y. Rule-based management of schema changes at ETL sources. In *European Conference on Advances in Databases and Information Systems (ADBIS)*, volume 5968 of *LNCS*, pages 55–62. Springer, 2010.

[40] Papastefanatos G., Vassiliadis P., Simitsis A., and Vassiliou Y. Policy-regulated management of ETL evolution. *Journal on Data Semantics*, 5530:147–177, 2009.

[41] Papastefanatos G., Vassiliadis P., Simitsis A., and Vassiliou Y. Metrics for the prediction of evolution impact in ETL ecosystems: A case study. *Journal on Data Semantics*, 1:75–97, 2012.

[42] Popovic A., Ivkovic V., Trajkovic N., and Lukovic I. A domain-specific language for managing ETL processes. *PeerJ Computer Science*, 10:e1835, 2024.

[43] Qiu D., Li B., and Su Z. An empirical analysis of the co-evolution of schema and code in database applications. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 125–135. ACM, 2013.

[44] Ravat F., Teste O., and Zurfluh G. A multiversion-based multidimensional model. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, pages 65–74. LNCS 4081, 2006.

[45] Romero O. and Wrembel R. Data engineering for data science: Two sides of the same coin. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, volume 12393 of *LNCS*. Springer, 2020.

[46] Rundensteiner E. A., Koeller A., Zhang X., Lee A. J., Nica A., Van Wyk A., and Lee Y. Evolvable view environment (EVE): Non-equivalent view maintenance under schema changes. In *International Conference on Management of Data (SIGMOD)*, pages 553–555, 1999.

[47] Schlesinger L., Bauer A., Lehner W., Ediberidze G., and Gutzman M. Efficiently synchronizing multidimensional schema data. In *International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 69–76, 2001.

[48] Tu S. and Zhu L. An optimized etl fault-tolerant algorithm in data warehouses. In *IEEE International Conference on Information Science and Technology (ICIST)*, pages 484–487, 2013.

[49] Vaisman A. and Mendelzon A. A temporal query language for OLAP: Implementation and case study. In *Database Programming Languages (DBPL)*, pages 78–96. LNCS 2397, 2001.

[50] Vaisman A. A. and Zimányi E. *Data Warehouse Systems - Design and Implementation, Second Edition*. Data-Centric Systems and Applications. Springer, 2022.

[51] Vassiliadis P. Profiles of schema evolution in free open source software projects. In *International Conference on Data Engineering (ICDE)*, pages 1–12. IEEE, 2021.

[52] Vassiliadis P. and Kalampokis G. Taxa and super taxa of schema evolution and their relationship to activity, heartbeat and duration. *Information Systems*, 110:102109, 2022.

[53] Vassiliadis P., Kolozoff M., Zerva M., and Zarras A. V. Schema evolution and foreign keys: a study on usage, heartbeat of change and relationship of foreign keys to table activity. *Computing*, 101:1431–1456, 2019.

[54] Vassiliadis P., Shehaj F., Kalampokis G., and Zarras A. V. Joint source and schema evolution: Insights from a study of 195 FOSS projects. In *International Conference on Extending Database Technology (EDBT)*, pages 27–39. OpenProceedings.org, 2023.

[55] Vassiliadis P. and Zarras A. V. Schema evolution survival guide for tables: Avoid rigid childhood and you're en route to a quiet life. *Journal on Data Semantics*, 6:221–241, 2017.

[56] Vassiliadis P., Zarras A. V., and Skoulis I. Gravitating to rigidity: Patterns of schema evolution – and its absence – in the lives of tables. *Information Systems*, 63:24–46, 2017.

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

[58] Wojciechowski A. and Wrembel R. On case-based reasoning for ETL process repairs: Making cases fine-grained. In *International Conference on Databases and Information Systems (DB&IS)*, volume 1243 of *CCIS*, pages 235–249. Springer, 2020.

[59] Wrembel R. On handling the evolution of external data sources in a data warehouse architecture. In *Integrations of Data Warehousing, Data Mining and Database Technologies - Innovative Approaches*, pages 106–147. Information Science Reference, 2011.

[60] Wu S. and Neamtiu I. Schema evolution analysis for embedded databases. In *Workshops @ International Conference on Data Engineering (ICDE)*, pages 151–156. IEEE, 2011.

Received 21.10.2024, Accepted 20.03.2025