ABSTRACT

Long-term data-driven studies have become indispensable in many areas of science. Often, the data formats, structures and semantics of data change over time, the data sets evolve. Therefore, studies over several decades in particular have to consider changing database schemas. The evolution of these databases lead at some point to a large number of schemas, which have to be stored and managed, costly and time-consuming. However, in the sense of reproducibility of research data each database version must be reconstructible with little effort. So a previously published result can be validated and reproduced at any time.

Nevertheless, in many cases, such an evolution can not be fully reconstructed. This article classifies the 15 most frequently used schema modification operators and defines the associated inverses for each operation. For avoiding an information loss, it furthermore defines which additional provenance information have to be stored. We define four classes dealing with dangling tuples, duplicates and provenance-invariant operators. Each class will be presented by one representative.

By using and extending the theory of schema mappings and their inverses for queries, data analysis, why-provenance, and schema evolution, we are able to combine data analysis applications with provenance under evolving database structures, in order to enable the reproducibility of scientific results over longer periods of time. While most of the inverses of schema mappings used for analysis or evolution are not exact, but only quasi-inverses, adding provenance information enables us to reconstruct a sub-database of research data that is sufficient to guarantee reproducibility.

KEYWORDS

CHASE-Inverses, Schema Evolution, Data Provenance, Research Data Management, Reproducibility

1 INTRODUCTION

Research institutions all around the world produce huge amounts of research data, which have to be managed by research data management. This includes collecting, evaluating, analyzing, archiving, and publishing the original or processed research data as well as the results of (data) analyses, theoretical considerations, or other scientific investigations. In this paper we are focusing on structured data, represented as a relational database, generated in scientific experiments, measurement series or always-on sensors.

The presentation and publication of research results increasingly requires publishing the corresponding research data, which ensures the findability, accessibility, interoperability, and re-usability in the sense of FAIR-Principles\(^1\). FAIR does not require the publication of all source data. It is sufficient to specify the source tuples necessary for the calculation of a specific research result. To perform this calculation, we use the so-called CHASE algorithm supplemented with provenance information such as witness bases [4], provenance polynomials [12] and/or additional side tables.

The CHASE is a procedure that modifies a database instance \(I\) by incorporating a set of dependencies \(\Sigma\) represented as (s-t) tgds or egds. Originally developed for database design and semantic query optimization, the CHASE nowadays is also used for database applications such as data exchange, data integration or answering queries using views. Formulated as schema mapping \( M = (I, J, Q)\) using s-t tgds, i.e. a dependency from one database \(I\) into a second database \(J\), the CHASE can process database queries \(Q\). This is equivalent to a mapping from red to green in Figure 1. Thus, the CHASE can be used for evaluating conjunctive queries or simple SQL queries, such as \(\text{SELECT}\ m, n \text{ FROM } F, S \text{ WHERE } F. m = S. m\). We call these queries against the research data evaluation queries.

Figure 1: Chasing evaluation queries and evolution mappings

In addition, the CHASE can be used for inverting queries. For this, we choose \(J\) as database instance and the inverse evaluation query \(Q^{-1}\) as schema mapping \(M^* = (J, I, Q^{-1})\) formalized as s-t tgd like before. Running both evaluation query and inversion in sequence, this results in a CHASE\&BACKCHASE process [8]. Here the BACKCHASE phase corresponds to a second CHASE phase with the CHASE result of the first phase as input. However, the result \(I^* = \text{CHASE}_{M^*}(\text{CHASE}_{M^*}(I))\) after applying CHASE\&BACKCHASE to \(I^*\) may not be equal to \(I\). In general, \(I^*\) is a subset of \(I\), called (minimal) sub-database (highlighted in red, Figure 1). The corresponding inverse, calculated with the CHASE is called CHASE-inverse. Such inverses are not necessarily exact and unique. They are so-called pseudo-inverse functions of different granularity [1, 10].

In research data management, not only the data is changing (e.g. by adding results of new experiments), but also the data’s

---

\(^1\)https://www.go-fair.org/
schemas. Changing schemas often results in freezing the entire research database to be able to perform the evaluation queries used so far. However, this can be time-consuming, expensive and leads to a dramatically increasing amount of data to be archived. Formulating a schema evolution $E$ as a schema mapping by $s$-$t$ tgds allows processing it by the Chase just like the evaluation queries before. The composition of inverse evolution steps and original evaluation queries now enables the reconstruction of research data for changing database structures, summarized in Figure 1.

At first glance, choosing the Chase algorithm may seem a bit unusual, but the choice has some significant benefits:

1. Both schema evolution and simple evaluation queries can be interpreted as schema mappings and processed with the Chase algorithm [10]. So, both mappings can be treated with the same theory.
2. An elaborated theory about quasi-inverses of schema mappings as well as their composition already exists [9, 10].
3. An examination of different types of inverses of the evaluation queries can be found in [1]. It can be improved by additional provenance information. This allows the reconstruction of lost null tuples and duplicates, as well as the replacement of null values by concrete values.

Interpreting evolution and evaluation as schema mapping and processing by CHASE & BACKCHASE enables the reconstruction of a research result even in the presence of changing database structures (see Figure 1). For this, an evolution mapping, represented as schema modification operator, an evaluation query as well as the two inverses are formulated as $s$-$t$ tgds. The Chase calculates the query result, which is then processed by the BACKCHASE. This provides the (minimal) sub-database in question (highlighted in red).

The inverse type can be used to determine how much and what part of the original data can be recovered without additional effort. In [1] the Chase-inverse types of 14 typical evaluation operators were specified with and without using data provenance. Provenance deals with the traceability of a result back to its original, possibly physical object. In the case of data provenance, we are interested in where a result tuple comes from, why and how it is calculated. In [1], it is observed that provenance improves the inverse types in about half of the cases.

In this article, we will perform this analysis for the evolution. For this purpose, we are looking at 15 schema modification operators, which are particularly common in practice [2, 6, 7, 15, 19, 21], and verify their invertibility. Then we improve the inverse type again by adding provenance. This again improves about half of the cases. Other approaches for combining schema evolution and provenance are given in [11] and [15]. In total, we identify four classes of invertibility. For each class we consider one representative in more detail (see Table 2).

Structure of the Article. To determine the relevant original tuples, we use the Chase algorithm not only for processing an evaluation query but also for evolution, see Section 3. For this purpose, in Section 4, we divide schema modification operators into four classes and examine five representatives with respect to their Chase-inverses with and without additional provenance information. Before that, in Section 2, we introduce the necessary definitions regarding Chase and data provenance.

2 Basics and Related Work

This article combines Chase, inverse schema mappings and data provenance from different areas of database theory. We introduce the most important terms here, further details can be found in the respective literature.

CHASE. The Chase algorithm is a technique used in a number of data management tasks like data exchange, data cleaning or answering queries using views. Its wide range of applicability can be attributed to the fact that Chase generalizes well. Generally speaking, it incorporates a set of dependencies $\Delta$ into a given object $\Omega$ - usually an instance or a query. Only condition: The dependencies must be representable as (s-t) tgds and egds.

A source-to-target tuple generating dependency (s-t tgd) is a formula of the form $\forall x : \phi(x) \rightarrow \exists y : \psi(x, y)$, with $x, y$ tuples of variables, and $\phi(x), \psi(x, y)$ are conjunctions of atoms, called body and head. It can be seen as inter-database dependency, representing a constraint within a database. An intra-database dependency is called tuple generating dependency (tgd). A dependency with equality atoms in the head equality generating dependency (egd).

In the case of instances, chasing (s-t) tgds create new tuples and chasing egds clean the database by replacing null values (by other null values or constants). So, the Chase database satisfies all dependencies of $\Delta$. We use the Chase to process schema evolution. Thus, we deal with inter-database mapping and can restrict ourselves to s-t tgds in the following. Since chasing s-t tgds always terminates, we refer to [5, 13] for further explanations about Chase variants and termination criteria.

CHASE-inverse functions. Let $I$ and $J$ be two database instances before and after applying the Chase, i.e. $J = \text{CHASE}(I)$. A Chase-inverse is a function from $J$ to $I^*$, which applies the Chase algorithm to $J$ with $I^*$ section of $I$. $I^*$ is a section of $I$ (short: $I^* \subseteq I$) iff a homomorphism $h : I^* \rightarrow I$ exists, whereby constants are mapped to constants and null values are mapped to constants or (other) null values [1]. Chase-inverse functions belong to the quasi-inverse functions. Formally, we call $g : J \rightarrow I$ a quasi-inverse function to $f : I \rightarrow J$ iff $I^* \subseteq I$ and $f \circ g \circ f = f$ [9]. This means that quasi-inverse functions usually recover the original data only partially. In the case of Chase five inverse types are distinguishable: exact, classical, relaxed [10], tuple-preserving relaxed and result equivalent [1]. The sufficient and necessary conditions of the different Chase-inverse types can be seen in Table 1.

For this, let $M$ be a schema mapping, $I$ and $I'$ two database instances with $I^* = \text{CHASE}_{M^*}(\text{CHASE}_M(I))$. $I'$ is called sub-database of $I$. A schema mapping $M$ can be formalized as a triple $(S_I, S_{I+1}, \Sigma)$ with $S_I$ source schema, $S_{I+1}$ target schema and $\Sigma$ set of dependencies that specify the relationship between $S_I$ and $S_{I+1}$. The affiliated inverse mapping is defined as $M^* = (S_{I+1}, S_I, \Sigma^{-1})$. The star " symbolizes that the inverse function $M^*$ may not be exact or unique. Again it is a quasi-inverse function.

While an exact Chase-inverse ($\simeq$) reconstructs the original database instance $I$, i.e. $I^* = I$, the classical Chase-inverse ($\approx$) only returns an instance $I^*$, which is equivalent to the original database instance $I$, i.e. $I^*$ and $I$ are equal except for isomorphism. The relaxed Chase-inverse ($\simeq$) does not require an equivalent relationship between the original instance $I$ and the recovered instance $I^*$, but
To preserve the number of tuples, the definition of relaxed Chase-inverse is extended [1]. The corresponding function, called tp-relaxed Chase-inverse ($\approx_{tp}$), additionally requires $|I^*| = |I|$. In contrast, a weakened definition is the result equivalent Chase-inverse ($\leftrightarrow$), which only satisfies data exchange equivalence, i.e., $\text{CHASE}_M(I) \equiv \text{CHASE}_M(I^*)$.

As studied in [1], evaluation queries formalised as s-t tgds and applied by CHASE are classified as exact, (tp-)relaxed or result equivalent depending on query under consideration. The addition of provenance information like provenance polynomials [14] or query-based provenance polynomials allows the specification of stronger Chase-inverses than without. For example, in the case of projection or union, a stronger Chase-inverse can be constructed by using provenance. For other operations, such as selection, the inverse type can not be improved despite additional information.

### Chase&Backchase

A schema mapping can be processed using the CHASE and be inverted using the Backchase [8]. The CHASE phase computes a modified instance $I^*$ generated from $I$ by applying the CHASE using a schema mapping $M = (S, T_{s+1}, \Sigma)$. After that, the Backchase phase computes an instance $\hat{I}$ by applying the Backchase to $J$ using a schema mapping $M^* = (S_{t+1}, S_T, \Sigma^{-1})$:

$$I^* = \text{CHASE}_{M^*}(J) = \text{CHASE}_{M^*}(\text{CHASE}_M(I)).$$

The calculated instance $I^*$ can, but must not be sub-database of $I$, short $I^* \prec I$. This depends on the inverse type defined above.

### Data Provenance

Depending on the authors (see [16] or [18]), a distinction is made between different types of provenance. These include data provenance, workflow provenance and metadata provenance. The questions to be answered depend on the provenance type and can be answered in different ways. In this article, we will focus on data provenance.

Let $I$ database instance and $Q$ query. Data provenance describes (1) where a result tuple $r \in Q(I)$ does come from, (2) why and (3) how $r$ exists in the result $Q(I)$. This can be answered extensively by specifying concrete tuples and database values, intentionally by characterizing a set of database values [17] or query-based by query transformation [3]. We consider only extensional answers.

The so-called why-provenance [4] specifies a witness basis that identifies the tuples involved in the calculation of $r$. This tuple-based basis contains all ids necessary for reconstructing $r$. How-provenance, on the other hand, uses provenance polynomials [14, 20] to explain how a result tuple $r$ is calculated. The polynomials are defined by a commutative semi-ring $(\mathbb{N}[X], +, \cdot, 0, 1)$ with $+$ for union and projection, as well as $\cdot$ for natural join [12]. If we are simply interested in identifying ID or relation name of the tuples involved, we choose where-provenance. It is the weakest of the three provenance questions, but is already sufficient in many cases.

### Schema Evolution

Schema evolution deals with the problem of maintaining schema and functionality of a software system considering a changing data structure. It can be addressed by so-called Schema Modification Operators (SMOs) that support semantic conversion of columns, column concatenation/splitting and other schema modifications. Common operators in the case of research data management include ADD Column and MERGE Column as well as SPLIT Column [2]. Other applications are described in [19, 21]. For initial approaches to combining schema evolution and provenance, see [11] and [7].

## 3. SMOS AND THEIR CHASE-INVURSES

Especially in the case of long-term data-driven studies, the specification of source data may become a problem. Therefore, we process not only the evaluation but also the evolution by the CHASE&Backchase and extend it with data provenance. For this, we classify the most frequently used schema modification operators (SMOs) and divide them into four classes. We define the associated inverses for each operation and examine it with additional provenance information. While the first class includes all operators with an exact inverse, i.e., the operators are provenance-invariant, the second and third class deal with dangling tuples and duplicates. The remaining operators we summarize in Class IV.

### Chase&Backchase

Evaluation queries and schema evolution can be defined as schema mappings. Let be $S_T, S_{t+1}$ two database schemas and $\Sigma, \Sigma^{-1}$ two sets of dependencies, whereby $\Sigma$ corresponds to an evaluation query $Q$ and $\Sigma^{-1}$ corresponds to a schema modification operator. Then the evolution $E$ can be formalized as schema mapping $M = (S_T, S_{t+1}, \Sigma)$ and its corresponding inverse $E^{-1}$ as $M^* = (S_{t+1}, S_T, \Sigma^{-1})$. Let further be $I(S_T)$ and $J(S_{t+1})$ two database instances. Then the CHASE&Backchase calculates a (minimal) sub-database

$$I^*(S_T) = \text{CHASE}_{M^*}(J(S_{t+1})) = \text{CHASE}_{M^*}(\text{CHASE}_M(I(S_T)))$$

(highlighted red in Figure 2). If we are not interested in processing the entire instance $J$, we can alternatively process a subset $J^* \subset J$ in the Backchase (highlighted in blue). For simplicity, we will use $J$ in the following. In summary, we state: Evolution is processed...
by the Chase in the so-called Chase phase. The BackChase phase corresponds to the inversion of the evolution.

Since we allow quasi-inverse functions to invert the evolution, we symbolize the inverse mapping by $f^{-1}$. The exact inverse mapping of $f$ used for merging and splitting, on the other hand, is denoted by $f^{-1}$, which we consider as additional provenance information.

![Figure 2: Inverting schema evolution](image)

**Schema Modification Operators.** The atomic schema changes during an evolution can be represented by so-called schema modification operations. Based on [2, 6, 7, 15, 19, 21] and more we define 15 frequently used operators, summarized in Table 3. Some publications also speak of a sixteenth operator NOP, which performs no action but namespace management.

Each operator has an associated (quasi-)inverse function. The invertibility of an operator corresponds to the existence of a perfect or quasi-inverse and its uniqueness. ADD Column, MERGE Column, RENAME Column, SPLIT Column, CREATE Table, PARTITION Table, RENAME Table and NOP has a unique quasi-inverse function. According to [7], DROP Table, COPY Table, MERGE Table and DROP Column have one or more quasi-inverses. Our studies show that the SMO formalization as s-t tgds always has a unique (quasi-)inverse. Therefore, another inverse can exist only in the case of a second s-t tdg formalization. However, this only applies to COPY Table and DECOMPOSE Table (see Table 3).

All operators can be easily formalized as set of one or at most two s-t tdgs. Similar to our formalization is for example the formalization of [7]. However, there is a crucial difference in the interpretation of MERGE Table. The authors define this operator by $R(a, b, c) \rightarrow T(a, b, c)$, $R(a, b, c) \rightarrow V(a, b, c)$ with associated inverse $T(a, b, c) \lor V(a, b, c) \rightarrow R(a, b, c)$. Since Chase-inverses are not defined for disjunctive embedded dependency, we consider only s-t tdgs. However, the information loss of using a s-t tdg here can be compensated by adding additional provenance information.

Some operators such as PARTITION Table, COPY Column and MOVE Column require special conditions $cond_A$, which give restrictions on the attributes. The corresponding s-t tdgs are restricted to constant and attribute selection $\sigma_A \theta c$, either $\sigma_A \theta A_j$ with $\theta \in \{\lt, \le, =, \ge, \gt\}$. Other conditions can be formalized using additional side tables.

**Side tables** contain attribute values required for the reconstruction of the (minimal) sub-database defined above. Each tuple is a restriction of a source tuple $t$ and is uniquely identified by its ID $r_t$. This ID is not used as a key, but as additional provenance information. Thus, side tables always exist only in association with a witness basis or a provenance polynomial.

### Table 2: SMOs with exact (=), tp-relaxed (≤tp), relaxed (<) or result equivalent (↔) Chase-inverse extended by provenance (highlighted in orange); dangling tuples are highlighted in magenta and duplicates in cyan

| SMO          | Without Provenance | With Provenance |
|--------------|--------------------|-----------------|
| COPY Table   | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |
| MERGE Table  | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |
| CREATE Table | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |
| RENAME Table | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |
| SPLIT Table  | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |
| DROP Table   | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |
| NOP          | $R(a, b, c) \rightarrow T(a, b, c)$ | $R(a, b, c) \rightarrow T(a, b, c)$ |

**Inverse Types.** Adding Provenance may improve the inverse type. However, in many cases this is not necessary at all. Thus, about half of the operators already guarantee an exact Chase inverse without requiring additional information. The associated operators – COPY Table, CREATE Table, PARTITION Table, RENAME Table, ADD Column, COPY Column and RENAME Column plus NOP group together to form Class I. In the case of the remaining operators, additional provenance causes an improvement of the inverse type.

To cover the scope of Chase-inverse types as well as their provenance extensions, and to cover special cases such as dangling tuples and duplicates, we focus our analyses here on COPY Table (Class I),
### Table 3: Schema modification operators with corresponding inverse and class

| SMO        | description                                                                 | inverse               | class |
|------------|-----------------------------------------------------------------------------|-----------------------|-------|
| COPY Table | creates a duplicate of an existing table                                    | DROP Table            | I     |
| CREATE Table | introduces a new, empty table                                               | MERGE Table           | I     |
| DECOMPOSE Table | decomposes a source table into two tables without changing the data stored inside | DROP Table            | I     |
| DROP Table | removes an existing table                                                   | ADD Column            | III   |
| JOIN Table | combines two tuples from different tables                                    | JOIN Table            | I     |
| MERGE Table | takes two source tables and creates a new table that stores their union     | CREATE Table          | IV    |
| PARTITION Table | distributes tuples into two newly created tables according to the specified condition | DECOMPOSE Table       | II    |
| RENAME Table | changes a table name                                                        | PARTITION Table       | IV    |
| ADD Column | introduces a new column with values created by a user-defined constant or function | DROP Column           | I     |
| COPY Column | copies a column to another table according to the join condition             | DROP Column           | I     |
| DROP Column | removes an existing column from a table                                      | ADD Column            | III   |
| MERGE Column | sequence of ADD Column and DROP Column using a user-defined function         | SPLIT Column          | III   |
| MOVE Column | same like COPY Column, but the original column is deleted                    | MOVE Column           | II, III|
| RENAME Column | changes the name of a column                                                | RENAME Column         | I     |
| SPLIT Column | sequence of ADD Column and DROP Column using a user-defined function         | MERGE Column          | III   |
| NOP | nothing happens                                                             | NOP                   | I     |

JOIN Table (Class II), MERGE Table (Class IV) and MERGE Column (Class III). The results are summarized in Table 2. Each operator as well as its associated inverse is formalized as schema mapping $M = (S_i, S_{i+1}, \Sigma)$ respectively $M^* = (S_{i+1}, S_i, \Sigma^{-1})$ with $\Sigma$ (column 2) and $\Sigma^{-1}$ (columns 4 and 6) sets of s-t tgds. The inverse types are summarized in column 3 and 5. The last three columns describe the facts including additional provenance.

Splitting, merging or deleting columns often creates lost tuples. Tuples can also be eliminated when joining tables. Thus, we distinguish two cases: duplicates (Table 2, highlighted in cyan) and dangling tuples (highlighted in magenta). Duplicates are generated by the operators DECOMPOSE Table, DROP Column, MERGE Column, MOVE Column and SPLIT Column. They are summarized in Class III. Since MOVE Column may produce dangling tuples as well as duplicates, this operator is also part of Class II, along with JOIN Table. Except for DROP Table and PARTITION Table, all operators are thus assigned to one of the classes 1 to 3. These three classes are the class of further operators.

As seen in Table 2, the formalization of some operators as sets of s-t tgds is not always unique. Thus, COPY Table, DECOMPOSE Table, ADD Column and COPY Column can be represented in two different ways. The choice of the representation influences the resulting Chase-inverse type as well as type of the underlying dependencies. Even additional provenance information may affect the type.

#### Inverse Types using Provenance

Reconstructing lost attribute values in JOIN Table provides an exact Chase-inverse instead of a relaxed inverse, whereas the restriction to the relevant tuples in MERGE Table provides an exact Chase-inverse instead of a result-equivalent. Also, in the case of MERGE Column, additional provenance information can be used to improve the Chase-inverses significantly. Knowing the witness bases or provenance polynomials allows the reconstruction of duplicates. If we also know the inverse of function $f$ and store lost attribute values in a separate side table, it is also possible to specify an exact Chase inverse. Thus, provenance enables, among other things, the reconstruction of lost tuples (duplicates) and attribute values.

All in all, we obtain four classes: Class I contains all operators whose inverse type does not change due to additional provenance information. The operators of Class II and Class III deal with dangling tuples or duplicates. Here additional witness basis, provenance polynomials and side tables are indispensable. The remaining two operators are summarized in Class IV. The results of Table 2 are described in detail in Section 4.

#### Composition of several SMOs

Since the SMOs operate independently within a sequence, we can determine the inverses in isolation from each other. So, the inverse mapping of composition

$$M^* = (M_1 \circ \ldots \circ M_n)^{-1} = M_n^* \circ \ldots \circ M_1^*$$

results in a composition of the inverse sub-operations $M_1^*, \ldots, M_n^*$. Thus, the inverse type of $M^*$ corresponds to the type of the weakest partial inverse $M_i^*$. If no inverse exists for one of the partial operations $M_i$, neither for its composition $M = M_1 \circ \ldots \circ M_n$.

For example, MERGE Column is defined as a sequence of ADD Column and DROP Column operators. DROP Column has a (tp) relaxed Chase-inverse depends on whether duplicates exist and additional provenance is used. ADD Column, on the other hand, always has an exact Chase-inverse. So, the inverse type of MERGE Column corresponds to the inverse type of DROP Column. This results in tp-relaxed inverse with additional provenance and a relaxed inverse without. Adding side tables we can define an exact Chase-inverse, such as summarized in Table 2. Same applies for SPLIT Column and other user-defined operators.

#### Similarities to Query Evaluation

Processed by the Chase schema modification and evaluation queries have similar characteristics. MERGE Table, for example, corresponds to union $\cup$, PARTITION Table to constant or attribute selection $\sigma_{A,B,C}$ respectively $\sigma_{A,B}$, and JOIN Table refers to the natural join $\Join$. It follows that the inverse types match those of [1]. In the case of evaluation queries, about half of the inverses can be improved by additional provenance information. The same is true for evaluation. In both cases, the (tp) relaxed Chase-inverse dominates. However, an exact
We will examine the four classes one by one. For this, we formulate all SMOs of this class have an exact inverse. This SMO can be formalized in two different ways. It can be represented both as one or as a set of two s-t tgds. Here we consider the first case. Let \( M = (S_t, S_{t+1}, \Sigma) \) and \( M^* = (S_{t+1}, S_t, \Sigma^{-1}) \) be two schema mappings with

\[
\Sigma = \{ R(a, b, c) \rightarrow R'(a, b, c) \land V(a, b, c) \}, \\
\Sigma^{-1} = \{ R'(a, b, c) \lor V(a, b, c) \rightarrow R(a, b, c) \}
\]

that formalizes Copy Column and its inverse mapping. W.l.o.g. let \( R \) and \( V \) each be restricted to three attributes. Let further \( I = \{ R(x_1, x_2, x_3) \mid i \in \mathbb{N} \} \) be an arbitrarily chosen source instance. Chasing \( M^* \) after chasing \( M \) yields:

\[
I^* = \text{CHASE}_{M^*}(\text{CHASE}_{M}(I))
\]

This, we have no loss of information and obtain an exact Chase-inverse. Since we can already guarantee an exact inverse, adding provenance, i.e. additional witness bases or provenance polynomials over the source IDs, is not necessary here. We do not receive any additional or necessary information from it.

In addition to COPY table, COPY Column can also be formalized in two different ways. In both cases an exact Chase-inverse can be specified. For all other SMOs of Class I only one formalization exists, which guarantees an exact Chase inverse with and without using additional provenance.

Class II: Dangling Tuples. Depending on the source instance, dangling tuples may occur. In this case an exact Chase-inverse can not be guaranteed. Also, the addition of witness bases and provenance polynomials may not be sufficient in some cases. However, if we note the lost tuples in so-called side tables, we can also specify exact Chase-inverses. This concerns the SMOs JOIN table and MOVE Column.

**Lemma 4.2.** Let \( M = (S_t, S_{t+1}, \Sigma) \) be a schema mapping of Class II. Then the corresponding inverse function \( M^* = (S_{t+1}, S_t, \Sigma^{-1}) \) is exact if we do not have dangling tuples. However, if such tuples are present, the Chase-inverse can be defined as result equivalent or relaxed. By using provenance as well as additional side tables, we can specify an exact Chase inverse, too.

We will take a closer look at JOIN Table. For this, let \( M = (S_t, S_{t+1}, \Sigma) \) and \( M^* = (S_{t+1}, S_t, \Sigma^{-1}) \) two schema mappings with

\[
\Sigma = \{ R(a, b) \land V(c, d) \land b = d \rightarrow T(a, b, c) \}, \\
\Sigma^{-1} = \{ T(a, b, c) \rightarrow R(a, b) \land V(c, d) \land b = d \}.
\]

Let further be \( R(id, name), V(name, subject) \) and \( T(id, name, subject) \) three relations with an arbitrarily chosen source instance. Chasing \( M \) after chasing \( M^* \) returns

\[
I^* = \text{CHASE}_{M^*}(\text{CHASE}_{M}(I))
\]

Thus, we have no loss of information and obtain an exact Chase-inverse. Since we can already guarantee an exact inverse, adding provenance, i.e. additional witness bases or provenance polynomials over the source IDs, is not necessary here. We do not receive any additional or necessary information from it.

In addition to COPY table, COPY Column can also be formalized in two different ways. In both cases an exact Chase-inverse can be specified. For all other SMOs of Class I only one formalization exists, which guarantees an exact Chase inverse with and without using additional provenance.

Class II: Dangling Tuples. Depending on the source instance, dangling tuples may occur. In this case an exact Chase-inverse can not be guaranteed. Also, the addition of witness bases and provenance polynomials may not be sufficient in some cases. However, if we note the lost tuples in so-called side tables, we can also specify exact Chase-inverses. This concerns the SMOs JOIN table and MOVE Column.

**Lemma 4.2.** Let \( M = (S_t, S_{t+1}, \Sigma) \) be a schema mapping of Class II. Then the corresponding inverse function \( M^* = (S_{t+1}, S_t, \Sigma^{-1}) \) is exact if we do not have dangling tuples. However, if such tuples are present, the Chase-inverse can be defined as result equivalent or relaxed. By using provenance as well as additional side tables, we can specify an exact Chase inverse, too.

We will take a closer look at JOIN Table. For this, let \( M = (S_t, S_{t+1}, \Sigma) \) and \( M^* = (S_{t+1}, S_t, \Sigma^{-1}) \) two schema mappings with

\[
\Sigma = \{ R(a, b) \land V(c, d) \land b = d \rightarrow T(a, b, c) \}, \\
\Sigma^{-1} = \{ T(a, b, c) \rightarrow R(a, b) \land V(c, d) \land b = d \}.
\]

Let further be \( R(id, name), V(name, subject) \) and \( T(id, name, subject) \) three relations with an arbitrarily chosen source instance. Chasing \( M \) after chasing \( M^* \) returns

\[
I^* = \text{CHASE}_{M^*}(\text{CHASE}_{M}(I))
\]

Thus, we have no loss of information and obtain an exact Chase-inverse. Since we can already guarantee an exact inverse, adding provenance, i.e. additional witness bases or provenance polynomials over the source IDs, is not necessary here. We do not receive any additional or necessary information from it.

In addition to COPY table, COPY Column can also be formalized in two different ways. In both cases an exact Chase-inverse can be specified. For all other SMOs of Class I only one formalization exists, which guarantees an exact Chase inverse with and without using additional provenance.
Adding witness basiss or provenance polynomials (highlighted in orange) does not change the Chase-inverse type. These do not provide any additional information for us. However, if we additionally store the dangling tuple $r_2$ in an external side table (highlighted in gray), it can be added back to the reconstructed instance $I'$. Summarized we can guarantee an exact Chase-inverse, because:

$$I^* = \{R(1, Alice), S(Alice, Math), V(Alice, IT) \} \cup \{R(2, Bob)\} = I.$$

MOVE Column may contain duplicates in addition to dangling tuple. The corresponding SMO can be formulated in two different ways. It is the only SMO in our study that does not have a Chase-inverse in every case.

Class III: Duplicates. The presence of duplicates typically permits only the specification of a relaxed Chase-inverse. SMOs belonging to this class are DROP Column, MERGE Column, SPLIT Column and DECOMPOSE Table. An improvement of the inverse type is only possible by adding provenance information. In some cases even an exact Chase-inverse can be guaranteed.

**Lemma 4.3.** Let $M = (S_I, S_{I+1}, \Sigma)$ be a schema mapping of Class III. Then the corresponding Chase-inverse $M^* = (S_{I+1}, S_I, \Sigma^{-1})$ is (tp-)relaxed, depending on the existence of duplicates. By adding provenance information and side tables, the Chase-inverse type can be defined as tp-relaxed or exact.

**Proof.** Let $M = (S_I, S_{I+1}, \Sigma)$ and $M^* = (S_{I+1}, S_I, \Sigma^{-1})$ be two schema mappings with

$$\Sigma = \{(a, b, c) \rightarrow T(a, f(b, c))\},$$

$$\Sigma^{-1} = \{T(a, g) \rightarrow \exists D, E : R(a, D, E)\}$$
formalizing MERGE Column and its associated inverse mappings. W.l.o.g. let $R$ be restricted to three and $T$ to two attributes. Let further $f$ be any MERGE function, $I = \{(x_{i1}, x_{i2}, x_{i3})_r \mid i \in \mathbb{N} \land f(x_{i1}, x_{i2}) = f(x_{i2}, x_{i3}) \land r_1 \text{ is tuple identifier} \}$ an instance with tuple identifier $r_1$ and duplicate in $t_1$ and $t_2$. Let $w_j$ witness basis of duplicate $r_j$ with $w_j = w_k = \{\{r_j\}, \{r_k\}\}$. So, $w_j$ contains two sets of tuple identifiers, so-called witnesses. A witness, here \{\{r_j\}\} and \{\{r_k\}\}, holds all tuple IDs needed for reconstructing one tuple. Two witnesses represent a duplicate. Chasing $M^*$ after chasing $M$ then returns

$$I^* = \text{CHASE}_{M^*}(\text{CHASE}_M(I))$$

$$= \text{CHASE}_{M^*}(\{T(x_{i1}, f(x_{i2}, x_{i3}))_w \mid i \in \mathbb{N} \land f(x_{i1}, x_{i2}) = f(x_{i2}, x_{i3}) \land w_i \text{ witness basis with } w_j = w_k = \{\{r_j\}, \{r_k\}\}\})$$

$$= \{R(x_{i1}, \eta_{i1}, \eta_{i2}) \mid i \in \mathbb{N} \land \eta_{i1} = \eta_{i2} = \eta_{i3}\}$$

$$\subseteq_{(tp)} I.$$

Due to the existence quantifiers, the inverse s-tgd always generates two new null values $\eta_{i1}$ and $\eta_{i2}$. Thus, the attribute values processed in $f$ can not be recovered. Due to the duplicate $t_j = t_k$, the reconstructed instance $I'$ contains less tuples than the original instance $I$. The inverse $M^*$ is thus relaxed without and tp-relaxed with existing duplicates. Adding provenance guarantees an tp-relaxed Chase-inverse. In this case, $\eta_{i1} \neq \eta_{i2}$ as well as $\eta_{i2} \neq \eta_{i3}$ holds.

Let alternatively be $M^* = (S_{I+1}, S_I, \Sigma^{-1})$ inverse mapping with

$$\Sigma^{-1} = \{(T(a, b)) \rightarrow R(a, f^{-1}(a, f^{-1}(g, c)), c)\}.$$

This formalization can only be used in combination with provenance, since polynomials, side tables and the knowledge of $f^{-1}$ are necessary here. Then chasing $M^*$ after chasing $M$ returns

$$I^* = \text{CHASE}_{M^*}(\text{CHASE}_M(I))$$

$$= \text{CHASE}_{M^*}(\{T(x_{i1}, f(x_{i2}, x_{i3}))_p \mid i \in \mathbb{N} \land f \text{ invertible function}
\land p_i \text{ provenance polynomial with } p_j = p_k = r_k \}
\land f(x_{j2}, x_{j3}) = f(x_{k2}, x_{k3})
\cup \{(x_{i1})_r \mid i \in \mathbb{N} \land r_1 \text{ tuple identifier}\})
$$

$$= \{R(x_{i1}, f^{-1}(f(x_{i2}, x_{i3}), x_{i1})_{\Sigma_{x_{i1}}}, f^{-1}(f(x_{i2}, x_{i3}), x_{i1})_{\Sigma_{x_{i1}}}) \mid i \in \mathbb{N} \land f(x_{j2}, x_{j3}) = f(x_{k2}, x_{k3})\}$$

$$= \{R(x_{i1}, x_{i2}, x_{i3}) \mid i \in \mathbb{N}\}$$

$$= I.$$
Also duplicates can be reconstructed using this additional information. Overall an exact Chase-inverse can be guaranteed.

Figure 4 shows the situation above in a concrete example. Let be R a student-table where the points of the two passed modules should be summarized. The inverse without knowing f⁻¹ yields a relaxed Chase-inverse (see Figure 4a). However, the duplicate (Alice, 5) in T can be reconstructed using additional provenance information, here a provenance polynomial (highlighted in orange). Adding a sustainable side even allows the exact Chase-inverse if f⁻¹ is known (see Figure 4b). Proving the other SMOs of Class III is quite similar.

Class IV: Special SMOs. Almost all SMOs can be assigned to Classes 1 to 3. Only MERGE Table1 and DROP Table1 remain. These contain neither dangling tuple (see Class II) nor duplicates (see Class III). In contrast to Class I, however, adding provenance improves the inverse type in both cases.

Lemma 4.4. The SMO MERGE Table1 has a result equivalent Chase-inverse. Adding provenance information, the Chase-inverse type can be defined as exact.

The unification of two tables R and V can be formalized as schema mapping M with inverse mapping M⁺ = (S⁺₁, S⁻₁).

Σ⁺ = \{R(a, b, c) \rightarrow T(a, b, c), V(a, b, c) \rightarrow T(a, b, c)\},

Σ⁻ = \{T(a, b, c) \rightarrow R(a, b, c), T(a, b, c) \rightarrow V(a, b, c)\}.

Let further be R, S and T three relations with an arbitrarily chosen source instance I. Chasing M⁺ after chasing M then yields

I⁺ = Chase(M⁺(Chase(M(I)))) = Chase(M⁻(I)) \neq I,

whereby I⁺ may contain more tuples than I, since M⁺ preserves all existing tuples like (2, Bob, IT) and (1, Alice, IT) in Figure 5, which are not originally contained in R and T, respectively.

However, the redundant tuples in I⁺ can be identified and deleted by using how-provenance (highlighted orange in Figure 5). Knowing the tuples used for calculating J, allows us to reassign the tuples to their source relations. We can thus reconstruct the source instance error-free and guarantee an exact Chase-inverse. It is not significant how exactly the result is produced. It is sufficient to know which tuple ids are involved in the generation of (2, Bob, IT) or (1, Alice, IT). A generalized witness basis or witness list is sufficient, too.

Lemma 4.5. The SMO DROP Table1 has a relaxed Chase-inverse. Adding provenance information the Chase-inverse type can be defined as ip-relaxed.

CONCLUSION

Especially for long-term data-driven studies, changing databases lead at some point to a large number of schemas. However, in the sense of reproducibility of research data each database version must be reconstructable with little effort. Nevertheless, in many cases, such an evolution can not be fully reconstructed.

This article classified the 15 most frequently used schema modification operators and defined the associated inverses for each operation. For avoiding an information loss, it furthermore defined which additional provenance information have to be stored. All in all, we define four classes dealing with dangling tuples, duplicates and provenance-invariant operators.

By using and extending the theory of schema mappings and their inverses for queries, we are able to combine data analysis applications with provenance under evolving database structures, thus enabling the reproducibility of scientific results over longer periods of time.

REFERENCES

[1] T. Auge and A. Heuer: The Theory behind Minimizing Research Data: Result equivalent Chase-inverse Mappings. LWD24, CEUR Workshop Proceedings, vol. 2391, CEUR-WS.org, pp.1–12, 2018.
[2] T. Auge, E. Manthey, S. Jürgensmann, S. Feistel and A. Heuer: Schema Evolution and Reproducibility of Long-term Hydrographic Data Sets at the IOW. LWD2A, CEUR Workshop Proceedings, vol. 2738, CEUR-WS.org, pp.258–269, 2020.
[3] N. Bidlot, M. Herschel and K. Tzompanaki: Query-Based Why-Not Provenance with NedExplain. EDBT, OpenProceedings.org, pp.145–156, 2014.
[4] P. Buneman, S. Khamra and W. C. Tan: Why and Where: A Characterization of Data Provenance. ICDT, Lecture Notes in Computer Science, vol. 1793, Springer, pp.316–330, 2001.
[5] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro and E. Tsamoura: Benchmarking the CHASE. PODS, ACM, pp.57–58, 2017.
[6] C. Curino, H. J. Moon, L. Tanca and C. Zaniolo: Schema Evolution in Wikipedia – Toward a Web Information System Benchmark. ICES (1), pp.323–332, 2008.
[7] C. Curino, H. J. Moon and C. Zaniolo: Graceful database schema evolution: the PRem workbench. Proc. VLDB Endow., 1(1), pp.761–772, 2008.
[8] A. Deutsch, L. Popa and V. Tannen: Physical Data Independence, Constraints, and Optimization with Universal Plans. VLDB, Morgan Kaufmann, pp.459–470, 1999.
[9] R. Fagin, P. G. Kolaitis, L. Popa and W. C. Tan: Quasi-inverses of schema mappings. ACM Trans. Database Syst., 33(2), pp.111–1152, 2008.
[10] R. Fagin, P. G. Kolaitis, L. Popa and W. C. Tan: Schema Mapping Evolution Through Composition and Inversion. Schema Matching and Mapping, Data-Centric Systems and Applications, Springer, pp.191–222, 2011.
[11] B. Glavic, G. Alonso, R. J. Miller and L. M. Haas: TRAMP: Understanding the Behavior of Schema Mappings through Provenance. Proc. VLDB Endow., 3(4), pp.1314–1325, 2010.
[12] T. J. Green and V. Tannen: Provenance semirings. PODS, ACM, pp.31–40, 2007.
[13] S. Greco, C. Molinaro and F. Spesiano: Incomplete Data and Data Dependencies in Relational Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2012.
[14] T. J. Green and V. Tannen: The Semiring Framework for Database Provenance. PODS, ACM, pp.93–99, 2017.
[15] S. Gao and C. Zaniolo: Provenance Management in Databases Under Schema Evolution. TaPF, USENIX Association, 2012.
[16] M. Herschel, R. Dietelkämper, and H. Ben Lahmar: A survey on provenance of systems. Knowl. Inf. Syst., 35(7), pp.495–543, 2018.
[17] A. Motro: Using Integrity Constraints to Provide Intensional Answers to Relational Queries. VLDB, Morgan Kaufmann, pp.237–246, 1989.
[18] B. Pérez, J. Bohin, and C. Saizte-Adan: A systematic review of provenance systems. CEUR Workshop Proceedings, vol. 2738, CEUR-WS.org, pp.1–12, 2018.
[19] D. Qiu and I. Neamtiu: Schema evolution analysis for embedded databases. ICDE Workshops, IEEE Computer Society, pp.151–156, 2011.