The Theory behind Minimizing Research Data
— Result equivalent CHASE-inverse Mappings

Tanja Auge and Andreas Heuer

University of Rostock, Rostock, Germany
tanja.auge@uni-rostock.de
heuer@informatik.uni-rostock.de
https://dbis.informatik.uni-rostock.de

Abstract. In research data management and other applications, the primary research data have to be archived for a longer period of time to guarantee the reproducibility of research results. How can we minimize the amount of data to be archived, especially in the case of constantly changing databases or database schemes and permanently performing new evaluations on these data? In this article, we will consider evaluation queries given in an extended relational algebra. For each of the operations, we will decide whether we can compute an inverse mapping to automatically compute a (minimal) subdatabase of the original research database when only the evaluation query and the evaluation result is stored. We will distinguish between different types of inverses from exact inverses to data exchange equivalent inverses. If there is no inverse mapping, especially for aggregation operations, we will derive the necessary provenance information to be able to perform the calculation of this subdatabase. The theory behind this minimization of research data, that has to be archived to guarantee reproducible research, is based on the CHASE&BACKCHASE technique, the theory of schema mappings and their inverses, and the provenance polynomials to be used for ***how*** provenance.

Keywords: Data Provenance · Research Data Management · CHASE · Inverse Schema Mappings · Data Exchange Equivalence

1 Introduction

In several applications, an extensive amount of data has to be managed, analyzed, stored, and archived due to legal and provenance purposes. In our projects, we consider quite different applications dealing with growing amounts of data, such as research data management with measurement data and sensor data management for smart (assistive) systems aiming at the derivation of activity and intention models by means of Machine Learning algorithms. In all of these cases, the massive amount of primary measurement or sensor data has to be reduced to an important *kernel*, that has to be archived for a longer period (often ten years, due to legal reasons).

Since the database of research and sensor data and its database schema evolves over time, the researcher or smart system developer is forced to store complete snapshots of the data each time, the database or database schema changes. In most cases, the management of research or sensor data becomes difficult to handle. The problem is, how to calculate the kernel of important data when databases or database schemes are changing. This is the aim of our research project. Which parts of the original database have to be *frozen*, when evaluations of research data are performed, or when decisions of smart systems have to be recorded? We want to minimize the subdatabase that has to be stored to guarantee the reproducibility of the performed evaluation.

How to automatically compute this minimal subdatabase of the primary data? Given an evaluation query and the result of the evaluation query, is it possible to derive the minimal subdatabase simply by an inverse mapping without any additional annotations of the data and without any need to freeze the original database? Or do we have to store some additional information, e.g. provenance polynomials [13, 15] or some kernel of the original database? In the latter case, we want to automatically compute these provenance polynomials or database kernels that need to be archived, too. In contrast to [2] and [4], we will give no introduction to the three provenance types ***where***, ***why*** and ***how*** in this paper. Here, we mainly concentrate on how to derive the answer for the ***why***- and ***how***-provenance-queries called Q_{prov} by inverse mappings (from the result to the original database) on the classification of the types of inverses representing Q_{prov}.

Our use case and main application scenario for minimizing research data is a cooperation of our database research group and the Leibniz Institute for Baltic Sea Research Warnemuende (IOW). Applications, data, data integration processes, evaluation queries, and data provenance problems are described in [6] in more detail. There, we described the use of a bitemporal database solution to handle provenance management as well as data and schema evolution.

In this paper, we want to describe the first step of our research: given an evaluation query and an evaluation result, does there exist an inverse mapping that computes a subdatabase of the original database that guarantees the reproducibility of the result? If there is no such inverse mapping, can we add some annotations like provenance polynomials to do so? In our case, the evaluation query is given in an extended relational algebra (adding some basic scalar, arithmetical operations, grouping, and aggregates to the classical relational algebra). For the derivation of the inverse mapping, we will adapt the theory of inverse schema mappings from the areas of data exchange, schema evolution and schema integration [10–12]. We will use the CHASE process [19] to perform the evaluation query on the original database and the BACKCHASE process [9] to do the inverse mapping.

In the next section, we will briefly introduce some fundamental definitions and results of the CHASE&BACKCHASE process, as well as schema mappings and their inverses. In Section 3, we introduce new types of inverse mappings, the *tuple preserving relaxed CHASE-inverse* and the *result equivalent CHASE-*

inverse. In Section 4, we will determine the possible types of inverses that can be computed for a specific operator of the extended relational algebra.

2 Related Work

The idea of our work is based on the combination of three different basic techniques: CHASE&BACKCHASE, CHASE-inverse schema mappings and data provenance. In this paper, we will introduce the first two of these. For a more detailed introduction into the state of the art in data provenance, we refer to [2–4].

2.1 The CHASE

The CHASE is a tool universally applicable in database theory. It is used, for example, to incorporate dependencies within a database or a relational algebra expression, to implicate dependencies, to verify the equivalence of database schemas under given dependencies, and to handle (replace or clean) null values in databases. The idea of this algorithm can be summarized as follows: For a CHASE object $\bigcirc$ and a CHASE parameter $*$ (in most cases: a set of dependencies), the CHASE incorporates $*$ into the object $\bigcirc$, so that $*$ is implicitly contained in $\bigcirc$, and thus $\bigcirc$ satisfies the dependencies in $*$. The result can be presented as:

$$\text{chase}_*(\bigcirc) = \circledast.$$

The CHASE has its original application in database design [19], which is based on the tableau definitions in [1]. The CHASE can not only be used with functional dependencies (FDs) and join dependencies (JDs) as $*$ on tableaus as $\bigcirc$ but also with tgds/s-t tgds (a generalization of JDs, see below) and egds (a generalization of FDs, see below) as $*$ on databases as $\bigcirc$. This is described among others by Fagin et al. in [10, 12].

For a query Q and a source instance I, the CHASE returns a result instance $K = \text{chase}_{\mathcal{M}}(I)$ if the following holds:

- Q is defined as a schema mapping $\mathcal{M} = (S_1, S_2, \Sigma)$ with source and target schemas S_1 and S_2 and a set of dependencies Σ describing the semantics of the query;
- I is the source instance in S_1;
- K is target instance in S_2.

These dependencies can be tgds or s-t tgds and egds. A *tuple-generating dependency* (tgd) is a sequence of the form

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y}))$$

with conjunctions $\phi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ of atoms from $\mathbf{x}$ and $\mathbf{x}$ and $\mathbf{y}$ respectively[1]. If $\phi(\mathbf{x})$ is a conjunction of atoms over a source scheme S and $\psi(\mathbf{x}, \mathbf{y})$ a conjunction

[1] $\mathbf{x}$ and $\mathbf{y}$ are vectors of variables x_1, x_2, ... or y_1, y_2, ...

of atoms over a target scheme T, such a sequence is also called *source-to-target tuple-generating dependency* (s-t tgd). An *equality-generating dependency* (egd) is for two variables x_1, x_2 from $\mathbf{x}$ defined by

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \rightarrow (x_1 = x_2)).$$

The CHASE&BACKCHASE is an algorithm for rewriting queries using views while exploiting integrity constraints. It was introduced in [9] for queries and views expressed as conjunctive queries and integrity constraints expressed as egds and tgds. While the CHASE step incorporates the dependencies into the query, the BACKCHASE step will perform the logical or physical optimization of the query, resulting in an optimized query execution plan. In fact, the BACKCHASE can be seen as a CHASE step in the *inverse* direction.

2.2 Inverting schema mappings

A schema mapping $\mathcal{M}$ is a triple (S_1, S_2, Σ), consisting of a source schema S_1, a target schema S_2 and a set of dependencies Σ that specifies the relationship between the source and target schema. CHASE-inverses are a special form of schema mappings [12] first formulated by R. Fagin [10]. CHASE-inverses work in situations where there is information loss, due to the existence of null values and missing information in databases. Since a CHASE-inverse does not necessarily have to exist, Fagin distinguishes three different types of CHASE-inverses: *exact*, *classical* and *relaxed CHASE-inverses*. In general, the inverse of a schema mapping $\mathcal{M} : S_1 \rightarrow S_2$ is defined as a schema mapping $\mathcal{M}^* : S_2 \rightarrow S_1$ whose composition $\mathcal{M} \circ \mathcal{M}^*$ corresponds to the identity function.

Inverse mappings are used in schema evolution, for example. While an *exact CHASE-inverse* always reconstructs the original database, the *classical CHASE-inverse* only returns an instance I^* which is equivalent to the original database instance I, i.e. I^* and I are equal except for the homomorphic mapping of marked null values to other marked null values or attribute values. The *relaxed CHASE-inverse* is another version of the CHASE-inverse [12]. It does not require an equivalent relationship between the original instance I and the recovered instance $I^* = \text{chase}_{\mathcal{M}^*}(\text{chase}_{\mathcal{M}}(I))$, but *data exchange equivalence* (see Section 3) and the existence of a homomorphism between the recovered instance I^* and the source instance I.

2.3 Combination of the techniques introduced

Given an original research database over database schema S_1 and an evaluation query Q, we now want to derive the result of the (***why*** or ***how***) provenance query Q_{prov} either as some type of inverse of Q or, if not possible, by use of provenance information. To calculate this minimal subdatabase of the original research database, that has to be archived to guarantee the reproducibility of the evaluations, we will use

- st-tgds for the basic operations of the extended relational algebra, to be used to express the evaluation query Q,
- the CHASE to formally evaluate Q on the original database instance I, to produce the result instance K,
- the BACKCHASE step to formally evaluate the provenance query Q_{prov}, i.e. some kind of inverse mapping, to determine the minimal subdatabase I^* of I (or an equivalent data set) that we will call *recovered instance*,
- and the different types of inverses of mappings (see Section 3) to determine whether this inverse can be calculated with or without additional annotations like provenance polynomials (see Subsection 4.3).

The complete process is defined in [3]. We will now add new types of inverse mappings to the ones used by Fagin: the *tuple preserving relaxed CHASE-inverse* and the *result equivalent CHASE-inverse*.

3 Tuple preserving relaxed and result equivalent CHASE-inverse

The *relaxed CHASE-inverse* is a weakening of the classical CHASE-inverse. It does not require an equivalence relationship between the source instance and the recovered instance I^*, but data exchange equivalence [12] and the existence of a homomorphism from the recovered instance I^* to the source instance I.

To preserve the number of tuples of the original database we extend the definition of the relaxed CHASE-inverse and we call it *tuple preserving relaxed CHASE-inverse* (tp-relaxed). This sharpens the character of the relaxed CHASE-inverse. In the opposite, a weakened definition is a schema mapping $\mathcal{M} = (S_1, S_2, \Sigma)$, called *result equivalent CHASE-inverse*, with regard to $\mathcal{M}$ (short: $I \leftrightarrow_{\mathcal{M}} K$), where the *data exchange equivalence* is valid for two instances I and K over S, i.e. $\text{chase}_{\mathcal{M}}(I) \equiv \text{chase}_{\mathcal{M}}(K)$. The result equivalent CHASE-inverse only requires result equivalence and no additional homomorphism from I^* to I and is therefore the weakest CHASE-inverse. Overall, this results in the reduction:

$$\text{result equivalent} \preceq \text{relaxed} \preceq \text{tp-relaxed} \preceq \text{exact}.$$

This reduction forms the sufficient condition for the existence of a CHASE-inverse. The necessary condition follows from the respective definition of the CHASE-inverse. Table 1 summarizes these conditions.

If the recovered instance I^* contains tuples with (marked) null values whose remaining attribute values match the attribute values of a tuple of the source instance I (for example $(1, 3, 5) \in I$ and $(1, 3, n_1) \in I^*$), then I^* is called *section* of the instance I (short: $I^* \preceq I$). In other words, there is a homomorphism h that maps the tuples from I^* to the tuples from I. For the above example, $1 \mapsto 1, 3 \mapsto 3$ and $n_1 \mapsto 5$ applies. If I^* contains no (marked) null values and all tuples from I^* are also tuples in I, we write $I^* \subseteq I$.

With these types of inverses, CHASE-inverse schema mappings can now be found for almost all basic operations of the extended algebra, we use for

CHASE-inverse	Sufficient condition	Necessary condition
Exact	-	$I^* = I$
Classical	Exact CHASE-inverse	$I^* \equiv I$
Tp-relaxed	Exact CHASE-inverse	$I^* \preceq I$, $\mid I^* \mid = \mid I \mid$
Relaxed	Tp-relaxed CHASE-inverse	$I^* \preceq I$
Result equivalent	Relaxed CHASE-inverse	$I^* \leftrightarrow_{\mathcal{M}} I$

Table 1: Sufficient and necessary condition for the existence of CHASE-inverse

the evaluation queries in research data management or the Machine Learning algorithms in smart, sensor-based systems.

4 Basic operations and CHASE-inverse schema mappings

In this paper we discuss the following problems: Is there an inverse mapping for an evaluation query and an evaluation result that calculates a sub-database of the original database and that guarantees the reproducibility of the result? And, if there is no such inverse mapping, can we add some annotations like provenance polynomials to do so? Therefore we neglect the homomorphy demanded by R. Fagin and define the so-called result equivalent CHASE-inverse. On the other hand we extend the definition of relaxed CHASE-inverse with the additional property that the number of tuples has to be equal in the source and the result instance. For almost each of the relational basic operations one of the four CHASE-inverse types can be detected now. For this purpose, the respective operation is formulated as s-t tgd and processed using the CHASE&BACKCHASE method. By composing the basic operations, CHASE-inverse mappings can also be found for general evaluation queries such as SQL-implementations of Machine Learning algorithms.

4.1 Basic operations as s-t tgds

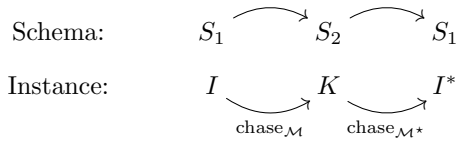
The basic operations of the extended relational algebra (copy, renaming, projection, natural join, selection, set operations and also arithmetic operations) can be written as s-t tgds based on their representations in Datalog. For aggregation and grouping, we refer to [17]. The other operations are defined in [14]. In Figure 1, we will present a small excerpt of these basic operations as s-t tgds, where r_j are relations, a_i domain variables for attributes A_i, resp., and $dom(A_i)$ the domain of A_i.

4.2 CHASE&BACKCHASE

The *recovered instance* $I^* = \text{chase}_{\mathcal{M}^*}(K) = \text{chase}_{\mathcal{M}^*}(\text{chase}_{\mathcal{M}}(I))$ is thus the result of a query Q_{prov} on the *result instance* K. I^* contains whole tuples from I or tuples restricted to certain attributes of the source schema (and filled with (marked) null values).

Copy:	$r(a_1, a_2, a_3) \rightarrow r(a_1, a_2, a_3)$
Projection:	$r(a_1, a_2, a_3) \rightarrow r(a_1, a_3)$
Natural Join:	$r_1(a_1, a_2) \wedge r_2(a_2, a_3) \rightarrow r(a_1, a_2, a_3)$
Selection:	$\exists c \in dom(A_1) : r(a_1, a_2) \wedge a_1 \theta c \rightarrow r(a_1, a_2)$ with $\theta \in \{<, \leq, =, \neq, \geq, >\}$

Figure 1: Basic operations as s-t tgds



The CHASE&BACKCHASE method for determining a CHASE-inverse schema mapping $\mathcal{M}^* = (S_2, S_1, \Sigma_2)$ to $\mathcal{M} = (S_1, S_2, \Sigma_1)$ can therefore be described in two phases:

- CHASE phase: Calculate the CHASE of I with respect to $\mathcal{M}$ as a sequence of s-t tgd and egd rules which generate new tuples (tgd) and replace (marked) null values with constants or null values with smaller index (egd).
- BACKCHASE phase: Calculate the CHASE of K regarding $\mathcal{M}^*$ as a sequence of s-t tgd and egd rules.

The second CHASE application is used here to "mapping back" the result instance K to the source instance I. However, this is usually only partially successful, so that an additional recovered instance I^* must be introduced, which is then compared with the source instance. This phase can be interpreted as the BACKCHASE phase. The idea of the CHASE&BACKCHASE method is based on the idea of Deutsch et al. [9]. However, if two schema mappings $\mathcal{M}$ and $\mathcal{M}^*$ are given instead of dependencies applied to queries as in [9], the BACKCHASE step $\text{chase}_{\mathcal{M}^*}(K)$ can be regarded as a kind of inverse schema mapping of the CHASE step $\text{chase}_{\mathcal{M}}(I)$. In other words, the mapping $\mathcal{M}^*$ is an CHASE-inverse mapping of $\mathcal{M}$.

4.3 CHASE-inverse Mapping

A general investigation of the most important basic operations on the existence of CHASE-inverse mappings forms the core of the Master's Thesis "Implementation of Provenance Queries in Big Data Analytics Environments"[2]. For an overview, we follow the lines of [4] in the next two paragraphs. Thus, with a few exceptions, an exact, tp-relaxed, relaxed or result equivalent CHASE-inverse schema mapping can be specified for the basic operations copy, renaming, projection, natural join and selection, the set operations and classic aggregate functions (**MIN**, **MAX**,

COUNT, **SUM**, **AVG**) and also for grouping and the arithmetic operations (see Table 2, column 2).

The existence of an exact CHASE-inverse (=) can only be proven for a few relational operations such as copy or one-variable arithmetic operations. While no CHASE-inverse can be found for the selection for inequality and the set difference (xxx), most operations are tp-relaxed ($\preceq_{tp}$), relaxed ($\preceq$) or result equivalent CHASE-inverses ($\leftrightarrow$).

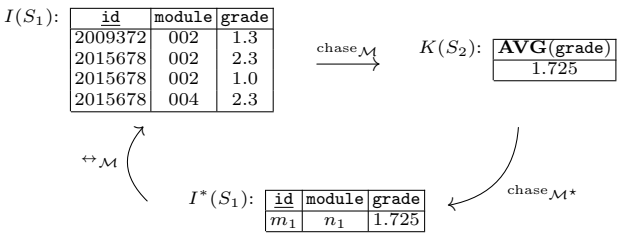


Figure 2: Example for aggregation without data provenance

The addition of provenance information allows the specification of stronger CHASE-inverse schema mappings than without (see Table 2, column 4). The provenance polynomials and (minimal) witness bases required for this correspond to the definitions in [16, 5, 15, 7]. Thus, in the case of aggregation operations **SUM** and **AVG** as well as union, an exact CHASE-inverse (=) can be constructed by using data provenance information instead of a result equivalent CHASE-inverse ($\leftrightarrow$). For other operations such as selection or **MAX** and **COUNT** the inverse type cannot be improved despite additional data provenance information. The type of these inverses is always relaxed ($\preceq$) or tp-relaxed ($\preceq_{tp}$). The operations copy, renaming, natural join with duplicates and also one-variable arithmetic operations provide an exact CHASE-inverse (=) without any additional provenance aspects. Despite provenance, no CHASE-inverse can be found for selection of inequality and set difference.

The situation is different for aggregate functions **SUM** and **AVG**. For illustration we consider the schema S_1, consisting of the attributes `id`, `module` and `grade`. The evaluation query Q performs the aggregation of the attribute `grade`. Without using data provenance, one tuple $(m_1, n_1, 1.725)$ is generated, which stores the aggregated value **AVG**(`grade`) (see also Figure 2). A result equivalent CHASE-inverse exists. By using provenance information, the exact CHASE-inverse can only be defined for the aggregated attribute `grade` itself. Therefore, the used provenance polynomial is calculated as $t = \frac{1.3 \otimes t_1 +_K 2.3 \otimes t_2 +_K 1.0 \otimes t_3 +_K 2.3 \otimes t_4}{t_1 +_K t_2 +_K t_3 +_K t_4}$. Related to the whole relation, there just can be specified a tp-relaxed CHASE-inverse mapping. In order to guarantee an exact CHASE-inverse in this case, the relation $K'(S_2')$ must be stored too (gray box in Figure 3), in addition to the evaluation query $Q =$ **AVG**(`grade`), the result database $K(S_2)$ and the provenance polyno-

mial t. The gray colored records of the recovered instance I^* are no longer null values, but can be specified concretely.

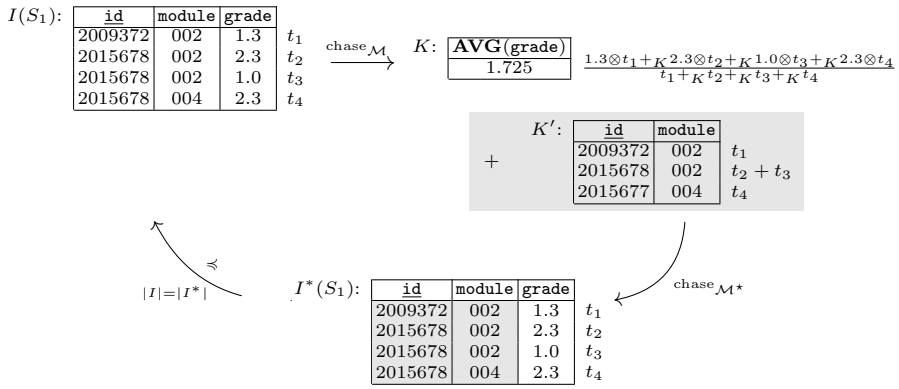


Figure 3: Example for aggregation using the provenance polynomial $t = \frac{1.3\otimes t_1 +_K 2.3\otimes t_2 +_K 1.0\otimes t_3 +_K 2.3\otimes t_4}{t_1 +_K t_2 +_K t_3 +_K t_4}$ and an additional relation K'

This additional *side table* K' is, however, in many cases not necessary or desired, for example due to data privacy aspects. A concrete traceability of the recovered result is already possible with the CHASE-inverse mapping shown in Figure 3 without the gray extension K'. In this case the `id` and `module` numbers have to be replaced by null values. The s-t tgds for this operation (aggregation with provenance polynomials) looks like

$$R(\texttt{id}, \texttt{module}, \texttt{grade}) \rightarrow R_{\text{result}}(f(\texttt{grade})) \text{ with } f(x) = \mathbf{AVG}(x)$$

for the CHASE and

$$R_{\text{result}}(\texttt{avg\_grade}) \rightarrow \exists\ \texttt{ID}, \texttt{Module} : R(\texttt{ID}, \texttt{Module}, \texttt{grade})$$

for the BACKCHASE step, where the `grade` can concretely be calculated from the result attribute value of `avg_grade` and the provenance polynomial t.

An explicit specification of the respective inverse $\mathcal{M}^*$ is also possible. Thus, the CHASE-inverse can be the identity mapping, renaming, projection, selection, the null value extension in A_i and null tuple generation, the reconstruction of lost attribute values A_i and the reconstruction of lost tuples. For relaxed schema mappings one of the above inverse always exists. The inverse can even be specified explicitly for the following operations: The natural join, one-variable arithmetic operations (multiplication, division, addition, and subtraction) as well as the renaming can be inverted by the projection, the inverse arithmetic operations (division, multiplication, subtraction, and addition, resp.) and the renaming. The inverse schema mapping of a result equivalent schema mapping is already the

identity. All realizations are represented in the third and fifth column of Table 2. In the case of aggregation, i.e. $\mathcal{M} = \mathbf{AVG}(A_i)$, this means for the inverse mapping $\mathcal{M}^*$ the reconstruction of the lost attribute value A_i and the null value extension in A_j with $i \neq j$.

For grouping, we have to distinguish three cases: an inverse type of relaxed ($\preceq$) for **MIN** and **MAX** and tp-relaxed ($\preceq_{tp}$) for **COUNT**. For **SUM** and **AVG** there exists a result equivalent CHASE-inverse ($\leftrightarrow$) or an exact CHASE-inverse ($=$) with and without using provenance information. For the inverse $\mathcal{M}^*$ this results in the identity mapping, the generation of null tuples, the null value extension in A_i and the reconstruction of lost tuples and concrete attribute values A_i (see Table 2, column 3 and 5).

4.4 Composition of basic operations

For the composition $\mathcal{M} = \mathcal{M}_1 \circ ... \circ \mathcal{M}_n$ of operations, the inverse mapping $\mathcal{M}^* = (\mathcal{M}_1 \circ ... \circ \mathcal{M}_n)^{-1} = \mathcal{M}_n^* \circ ... \circ \mathcal{M}_2^* \circ \mathcal{M}_1^*$ results in a composition of the inverse suboperations $\mathcal{M}_1^*, ..., \mathcal{M}_n^*$. The type of the CHASE-inverse $\mathcal{M}^*$ thus corresponds to the type of the weakest partial inverse $\mathcal{M}_i^*$ with $i = 1, ..., n$. If there are no CHASE-inverses to the suboperations $\mathcal{M}_i$ ($i = 1, ..., n$), there is also no CHASE-inverse to the schema mapping $\mathcal{M}$.

We use such a composition to implement more complex evaluation queries on research data such as Machine Learning algorithms, e.g. the *Hidden Markov model*. This model is one of the best-known machine learning algorithms. It describes a stochastic model in which a system is modeled by a Markov chain — a kind of "memory protocol" — with unobserved states. The future development of the process therefore depends only on the last observed state, but not on the previous states.

Based on the representation of the Hidden Markov model in the form of SQL statements [18], the required operations addition and subtraction, scalar multiplication and division as well as matrix-vector and matrix-matrix multiplication can be examined for the existence of CHASE-inverse mappings. Overall, the composition of these operations results in a result equivalent CHASE inverse.

5 Conclusion

In this paper, we investigated whether an inverse mapping exists for an evaluation query and an evaluation result, which calculates a subdatabase of the original database that guarantees the reproducibility of the result. For this purpose we have defined new CHASE-inverses and determined the possible types of inverses that can be computed for a specific operator of the extended algebra. In some cases, we used additional annotations like provenance polynomials. In Table 2, we have shown in which cases these additional annotations are necessary.

References

1. Aho, A. V.; Beeri, C.; Ullman, J. D.: *The Theory of Joins in Relational Databases.* ACM TODS, No. 3, Vol. 4, pp. 297–314 (1979)
2. Auge, T.: *Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen.* University of Rostock, Master's Thesis (2017)
3. Auge, T.; Heuer, A.: *Combining Provenance Management and Schema Evolution.* To be published in "Provenance and Annotation of Data and Processes", Proceedings of the 7th International Provenance and Annotation Workshop (IPAW), Springer LNCS Volume 11017 (2018)
4. Auge, T.; Heuer, A.: *Inverse im Forschungsdatenmanagement — Eine Kombination aus Provenance Management, Schema- und Daten-Evolution.* Proceedings of the 30th Workshop on "Grundlagen von Datenbanken". In: CEUR-WS.org Workshop Proceedings, pp. 108-113 (2018)
5. Amsterdamer, Y.; Deutch, D.; Tannen, V.: *Provenance for Aggregate Queries.* ACM PODS, pp. 153–164 (2011)
6. Bruder, I.; Klettke, M.; Möller, M.L.; Meyer, F.; Heuer, A.; Jürgensmann, S.; Feistel, S.: Daten wie Sand am Meer - Datenerhebung, -strukturierung, -management und Data Provenance für die Ostseeforschung, Datenbank-Spektrum, 17, 2, pp. 183–196, 2017
7. Buneman, P.; Khanna, S.; Tan, W. C.: *Why and Where: A Characterization of Data Provenance.* In: ICDT, Springer, Vol. 1, pp. 316–330 (2001)
8. Cheney, J.; Chiticariu, L.; Tan, W. C.: *Provenance in Databases: Why, How, and Where.* Foundations and Trends in Databases, Vol. 1, No. 4, pp. 379 – 474 (2009)
9. Deutsch, A.; Popa, L.; Tannen, V.: *Physical Data Independence, Constraints, and Optimization with Universal Plans.* Proceedings of 25th International Conference on Very Large Data Bases, pp. 459–470 (1999)
10. Fagin, R.: *Inverting Schema Mappings.* ACM TODS, No. 4, Vol. 32, pp. 25:1–25:23 (2007)
11. Fagin, R.; Kolaitis, P. G.; Popa, L.; Tan, W. C.: *Quasi-Inverses of Schema Mappings.* ACM TODS, No. 2, Vol. 33, pp. 11:1–11:52 (2008)
12. Fagin, R.; Kolaitis, P. G.; Popa, L.; Tan, W. C.: *Schema Mapping Evolution Through Composition and Inversion.* In: Schema Matching and Mapping , Springer (2011)
13. Green, T.J.; Karvounarakis, G.; Tannen, V.: *Provenance semirings.* Proceedings of the 26th ACM Symposium on PODS, pp. 31–40 (2007)
14. Greco, S.; Molinaro, C.: *Datalog and Logic Databases.* Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2015)
15. Green, T. J.; Tannen, V.: *The Semiring Framework for Database Provenance.* Proceedings of the 36th ACM Symposium on PODS, pp. 93–99 (2017)
16. Herschel, M.: *A Hybrid Approach to Answering Why-Not Questions on Relational Query Results.* J. Data and Information Quality, No. 3, Vol. 5, pp. 10:1–10:29 (2015)
17. Mohapatra, A.; Genesereth, M.: *Aggregation in Datalog Under Set Semantics* Stanford University, Techn. Report (2012)
18. Marten, D.; Heuer, A.: *Machine Learning on Large Databases: Transforming Hidden Markov Models to SQL Statements.* OJDB, Vol. 4, pp. 22–42 (2017)
19. Maier, D.; Mendelzon, A. O.; Sagiv, Y.: *Testing Implications of Data Dependencies.* ACM TODS, No. 4, Vol. 4, pp. 455–469 (1979)

Operation $\mathcal{M}$	Without Provenance		With Provenance	
	Inverse type	**Inverse $\mathcal{M}^\star$**	**Inverse type**	**Inverse $\mathcal{M}^\star$**
$r(\mathcal{R})$	$=$	identity	$=$	identity
$\beta_{A_j \leftarrow A_i}(r(\mathcal{R}))$	$=$	renaming $\beta_{A_i \leftarrow A_j}$	$=$	renaming $\beta_{A_i \leftarrow A_j}$
$\pi_{A_i}(r(\mathcal{R}))$	$\preceq_{tp}$	null value extension in A_i	$\preceq_{tp}$	null value extension in A_i
	$\asymp$	null value extension in A_i	$\preceq_{tp}$	null value extension in A_i + reconstruction of lost tuples
$r_1(\mathcal{R}_1) \bowtie r_2(\mathcal{R}_2)$	$=$	projection on attributes from $\mathcal{R}_i$	$=$	projection on attributes from $\mathcal{R}_i$
	$\asymp$	projection on attributes from $\mathcal{R}_i$	$\asymp$	projection on attributes from $\mathcal{R}_i$
$\sigma_{A_i \theta c}(r(\mathcal{R}))$	$\asymp$	identity	$\asymp$	identity
$\sigma_{A_i \theta A_j}(r(\mathcal{R}))$	$\asymp$	identity	$\asymp$	identity
with $\theta \in \{<, \leq, =, \geq, >\}$				
$\sigma_{A_i \neq c}(r(\mathcal{R}))$	xxx	xxx	xxx	xxx
$\sigma_{A_i \neq A_j}(r(\mathcal{R}))$	xxx	xxx	xxx	xxx
$r_1(\mathcal{R}_1) \cup r_2(\mathcal{R}_2)$	$\leftrightarrow$	identity	$=$	selection
$r_1(\mathcal{R}_1) \cap r_2(\mathcal{R}_2)$	$\asymp$	identity	$\asymp$	identity
$r_1(\mathcal{R}_1) - r_2(\mathcal{R}_2)$	xxx	xxx	xxx	xxx
$\mathbf{MAX}_{A_i}(r(\mathcal{R}))$ / $\mathbf{MIN}(r(\mathcal{R}))$	$\asymp$	identity	$\asymp$	null value extension in A_j, $i \neq j$
$\mathbf{COUNT}_{A_i}(r(\mathcal{R}))$	$\preceq_{tp}$	null tuple generation	$\preceq_{tp}$	null tuple generation
$\mathbf{SUM}_{A_i}(r(\mathcal{R}))$	$\leftrightarrow$	identity	$=$	reconstruction of lost attribute values in A_i
				+ null value extension in A_j, $i \neq j$
$\mathbf{AVG}_{A_i}(r(\mathcal{R}))$	$\leftrightarrow$	identity	$=$	reconstruction of lost attribute values in A_i
				+ null value extension in A_j, $i \neq j$
$\gamma_{G_i;F_j(A_j)}(r(\mathcal{R}))$	$\asymp$	identity	$\asymp$	identity
	$\preceq_{tp}$	null tuple generation	$\preceq_{tp}$	null value extension in A_i + reconstruction of lost tuples
	$\leftrightarrow$	identity	$=$	reconstruction of lost tuples and attribute values in A_i
$r(\mathcal{R})\ \theta\ \alpha$ with $\theta \in \{+, -, \cdot, /\}$	$=$	$- \ / \ + \ / \ : \ / \ \cdot$	$=$	$- \ / \ + \ / \ : \ / \ \cdot$

Table 2: Basic operations and their exact ($=$), tp-relaxed ($\preceq_{tp}$), relaxed ($\asymp$) or result equivalent ($\leftrightarrow$) CHASE-inverse with restriction to the attribute A_i for **COUNT**, **SUM** and **AVG**