ProSA Pipeline: Provenance Conquers the Chase

Tanja Auge(✉), Moritz Hanzig, and Andreas Heuer

University of Rostock, Rostock, Germany
{tanja.auge,andreas.heuer}@uni-rostock.de

Abstract. One of the main problems in data minimization is the determination of the relevant data set. Combining the Chase—a universal tool for transforming databases—and data provenance, a (anonymized) minimal sub-database of an original data set can be calculated. To ensure reproducibility, the evaluations performed on the original data set must be feasible on the sub-database, too. For this, we extend the Chase&Backchase with additional ***why***-provenance to handle lost attribute values, null tuples, and duplicates occurring during the query evaluation and its inversion. In this article, we present the ProSA pipeline, which describes the method of data minimization using the Chase&Backchase extended with additional provenance.

Keywords: Research data management · Data minimization · Data provenance · Chase&Backchase

1 Introduction

Research institutions around the world produce research data in large quantities. Collecting, evaluating, analyzing, archiving and publishing these data are major tasks of research data management. The effort required to manage and store these data volumes is often underestimated, especially when certain criteria, such as the traceability or reproducibility of a published result, are to be guaranteed. This can be supported substantially by adding additional provenance.

Let's imagine the following situation: We are writing an article for a journal or conference in which we have evaluated a larger data set. Since the conference or journal promotes reproducibility and traceability of new research results to be published, we are asked to publish our underlying data as well. Open data, as demanded in our example, is desirable in the interest of good science, but sometimes not feasible. Some reasons for not publishing data in detail are of privacy-related, economic, financial or military nature. We aim to limit the amount of data to the part that is relevant for publication. To determine this data set ProSA (**Pro**venance Management using **S**chema Mappings with **A**nnotations) uses a version of the Chase&Backchase. Additional provenance information such as ***why***-provenance and additional side tables allows to reconstruct as much information as possible. However, we try to keep the amount of data to be stored as small as possible.

© Springer Nature Switzerland AG 2022
S. Chiusano et al. (Eds.): ADBIS 2022, CCIS 1652, pp. 89–98, 2022.
https://doi.org/10.1007/978-3-031-15743-1_9

Systems for determining ***why***-provenance are already existing. A list of these can be found in [3], where we tested the best-known data provenance systems for their functionality and effectiveness. Most of them determine witness basis [7] based on a given SQL query. However, we provide a better suited approach. In ProSA we use the determined provenance in a subsequent step to classify not only the IDs of the original tuples. Instead, we determine the entire instance and then anonymize it as necessary. Some early ideas can be found in [1].

The foundation for our approach is a variant of the Chase, a family of algorithms for reasoning with constraints [5]. It modifies a database instance I by incorporating a set of dependencies Σ represented as *(source-to-target) tuple generating dependencies* ((s-t) tgds) or *equality generating dependencies* (egds). While chasing (s-t) tgds creates new tuples, chasing egds "cleans up" the database by substituting null values. We regard in the following the application of the Chase as chasing. We take advantage of this and use the Chase to evaluate queries on a given database instance.

For evaluating queries, we define a query Q as a set of (s-t) tgds Σ. Then $\mathcal{M} = (S, T, \Sigma)$ with S source schema, T target schema, and Σ set of database dependencies specifying the relationship between S and T formalizes a *schema mapping* which can be processed by the Chase. Thus, the Chase can be used for evaluating conjunctive queries or simple SQL queries, such as `SELECT n,m FROM R1, R2 WHERE R1.m = R2.m`. Even aggregate functions such as `MAX` and evolution operators such as `PARTITION Table`, which can be formulated as sets of (s-t) tgds, can also be processed with our approach.

Applying the Chase twice we call Chase&Backchase. This technique is used among others by [8] for query optimization or query reformulation. In these two cases, provenance supports also the optimization process of queries. We, on the other hand, do not use the Chase&Backchase for optimization but for evaluating queries and generating the minimal sub-database. In addition to evaluating queries, the Chase can be used to evolve databases. A corresponding algorithm can be found in [2].

Consider again our research data management use case from the very beginning. Given a query Q and a database instance I, ProSA uses the Chase&Backchase to determine the minimal sub-database I^* relevant for publication. However, an exact inverse cannot always be computed without additional information. Thus, we extend the Chase&Backchase with ***why***-provenance and additional side tables. But first, let's consider this with a concrete example. For this, let I be a database instance containing of two relations `employee`(id,name) and `salary`(id,sal), defined in Table 1a and 1b. Let further Q_1 and Q_2 be two SQL queries defined as:

Query Q_1

```
SELECT name, sal
FROM employee NATURAL JOIN salary
WHERE name = "Stefan";
```

Query Q_2

```
SELECT MAX(sal)
FROM salary
```

Then ProSA calculates in a first step, the Chase *phase*, the query result $Q(I)$ and in a second step, the Backchase *phase*, the reconstructed instance I^*. We call I^* *sub-database* of I if there is a homomorphism from I^* to I that maps all null values to corresponding constants in I. Furthermore, we call the sub-database I^* *minimal* if it is free of duplicates and solves the optimization problem *min* $|I^*|$ under the constraint $\text{Chase}_{\mathcal{M}}(I^*) = \text{Chase}_{\mathcal{M}}(I)$, i.e. the number of tuples of I^* is minimal respect to $Q(I) = Q(I^*)$.

After generating the minimal sub-database ProSA allows to perform a final anonymization, if necessary. For this, we generalize the sub-database based on user-defined domain generalization hierarchies. In doing so, we ensure k-anonymity and l-diversity, the most common anonymization measurements. The generated anonymized minimal sub-database we identify as I^*_{anon}.

Our Contribution. The ProSA pipeline describes a way to minimize data without loosing reproducibility of its corresponding evaluations. Our approach is particularly suitable for research data management, which more and more often requires the publication of data. For this, we combine the Chase&Backchase with additional provenance information such as ***why***-provenance and side tables. We not only determine the relevant source IDs, as in some known systems, but also determine and anonymize the resulting minimal sub-database.

In Sect. 2, we introduce the necessary definitions regarding the Chase and data provenance. In Sect. 3.2, we present our ProSA pipeline. The theory behind we introduce in Section and close with a short description of our implementation in Sect. 3.3.

2 Basics and Related Work

Chase. The Chase is a technique used in a number of data management tasks such as data exchange, data cleaning or answering queries using views [5,6]. It takes a set of dependencies Σ and an instance I as input and returns an instance J that satisfies all dependencies of Σ represented as (s-t) tgds and egds.

A *source-to-target tuple generating dependency* (s-t tgd) is a formula of the form $\forall \mathbf{x} : (\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y}))$, with $\mathbf{x}$,$\mathbf{y}$ tuples of variables, and $\phi(\mathbf{x})$, $\psi(\mathbf{x}, \mathbf{y})$ conjunctions of atoms, called *body* and *head*. A s-t tgd can be seen as inter-database dependency, representing a constraint within a database. An intra-database dependency is called *tuple generating dependency* (tgd) and a dependency with equality atoms in the head *equality generating dependency* (egd). Specifically, an egd is a formula of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow (x_1 = x_2))$ with $\mathbf{x}$ tuple of variables and $\phi(\mathbf{x})$ conjunction of atoms.

This means chasing (s-t) tgds create new tuples and chasing egds clean the database by replacing null values η_i (by other null values η_j or constants c_j). The Chase can be used to compute a target instance J from a source instance I via a schema mapping $\mathcal{M}$ [9]. A *schema mapping* is a triple $\mathcal{M} = (S, T, \Sigma)$ with S source schema, T target schema and Σ set of constraints describing the relationship between S and T. It is semantically identified with the binary

Table 1. Example relations with associated provenance: source instance I with relations `employee` (a) and `salary` (b), query result $Q_1(I)$ resp. $Q_2(I)$ with relation `result` (c) resp. (d), minimal sub-database I^* with relations `employee` (e) and `salary` (f) resp. (h) as well as side table `interim` (g).

(a) `employee`

id	name	
1	Stefan	e_{id_1}
2	Stefan	e_{id_2}
3	Mia	e_{id_3}
4	Anna	e_{id_4}

(b) `salary`

id	sal	
2	1500	s_{id_1}
3	1500	s_{id_2}
4	1250	s_{id_3}
1	1470	s_{id_4}
2	1470	s_{id_5}

(c) `result` of S_1

name	sal	
Stefan	1500	$\{\{e_{\mathrm{id}_2}, s_{\mathrm{id}_1}\}\}$
Stefan	1470	$\{\{e_{\mathrm{id}_1}, s_{\mathrm{id}_4}\}, \{e_{\mathrm{id}_2}, s_{\mathrm{id}_5}\}\}$

(d) `result` of S_2

sal	
1470	$\{\{s_{\mathrm{id}_1}\}, \{s_{\mathrm{id}_2}\}\}$

(e) `employee`

id	name	
η_1	Stefan	e_{id_1}
η_2	Stefan	e_{id_2}

(f) `salary`

id	sal	
η_2	1500	s_{id_1}
η_1	1470	s_{id_4}
η_2	1470	s_{id_5}

(g) `interim`

name	
1250	e_{id_3}
1470	e_{id_4}
1470	e_{id_5}

(h) `salary`

id	sal	
η_1	1500	s_{id_1}
η_2	1500	s_{id_2}

relation: $\mathrm{Inst}(\mathcal{M}) = \{(I, J) \mid (I, J) \models \Sigma\}$, whereby I is S-instance, J is T-instance and Σ defined as set of (s-t) tgds and egds.

The Chase produces a target instance J, denoted as $\textsc{Chase}_{\mathcal{M}}(I)$, as follows: For every s-t tgd $\forall \mathbf{x} : (\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y}))$ in Σ and for every tuple t_1 of constants of I we add to $\textsc{Chase}_{\mathcal{M}}(I)$ all facts in $\psi(t_1, \eta)$ where η is a tuple of new, distinct labeled nulls interpreting the existential quantified variables $\mathbf{y}$. However, if there is a tuple t_2 of constants c_i or labeled null values η_i such that $\psi(t_1, t_2)$ already exists in J, no new tuple is created.

Let's take a look at a concrete example. Let S be source schema with two relations `employee`(id,name) and `salary`(id,sal) and T target schema with one relation `result`(name,sal). Let further I be the source instance defined in Table 1a and 1b. Let $\Sigma = \{\texttt{employee}(\mathrm{id}, \text{“Stefan”}) \wedge \texttt{salary}(\mathrm{id}, \mathrm{sal}) \rightarrow \texttt{result}(\text{“Stefan”}, \mathrm{sal})\}$ be formalization of Q_1. Then the corresponding schema mapping is defined as triple $\mathcal{M} = (S, T, \Sigma)$. Let `employee`$(1, \text{Stefan})$ and `salary`$(1, 1470)$ be two tuples of I. The Chase of I respectively $\mathcal{M}$ will then produces a target instance $J = \textsc{Chase}_{\mathcal{M}}(I)$ that consists the fact `result`$(1, 1470)$.

Provenance. As described in [11], "provenance generally refers to any information that describes the production process of an end product, which can be anything from a piece of data to a physical object." For this, let I be a database instance and Q conjunctive query. It describes (1) ***where*** a result tuple $r \in Q(I)$ comes from, (2) ***why*** and (3) ***how*** r exists in the result $Q(I)$. In this paper, we focus on ***why***-provenance [7], which specifies a witness basis identifying the tuples involved in the calculation of r. This tuple-based basis contains all IDs necessary for reconstructing r.

For example, let `employee`$(1, \text{Stefan}, e_{\mathrm{id}_1})$, `employee`$(2, \text{Stefan}, e_{\mathrm{id}_2})$, `salary` $(1, 1470, s_{\mathrm{id}_1})$ and `salary`$(2, 1470, s_{\mathrm{id}_2})$ be four tuples of I defined in Table 1a and 1b. The corresponding witness basis $\{\{e_{\mathrm{id}_1}, s_{\mathrm{id}_4}\}, \{e_{\mathrm{id}_2}, s_{\mathrm{id}_5}\}\}$ consists of two witnesses $\{e_{\mathrm{id}_1}, s_{\mathrm{id}_4}\}$ and $\{e_{\mathrm{id}_2}, s_{\mathrm{id}_5}\}$. The first duplicate uses the tuples e_{id_1} and s_{id_e}, while the second uses the tuples e_{id_2} and s_{id_5}.

3 The ProSA-Pipeline

As seen in Fig. 1, the ProSA pipeline consists of ten steps. The Chase& Backchase is executed in step 4 (evaluating query Q) and step 7 (evaluating inverse query Q^{-1}), outsourced in ChaTEAU [4]. The specification of the inverse query Q^{-1} is done by the Inverter in step 5. However, an *exact inverse*—a function f with $f \circ f = \mathrm{id}$—cannot always be computed without additional information. This is especially the case for highly selective and summarizing queries. But provenance can solve this problem.

Preparing provenance takes place in the Provenancer in step 3. Here provenance IDs and additional side tables are introduced. The calculation of ***why***-provenance is part of the Chase phase and its evaluation is considered in the Backchase phase. Subsequently, the Controller in step 8 verifies the correctness of the calculated minimal sub-database before it is anonymized in the Anonymizer in step 9.

3.1 The Theory Behind ProSA

Chase&Backchase. In this article, we use the Chase to evaluate queries on instances. Let Q be a query formalized as a set of (s-t) tgds Σ and I over S a given database instance. Let further be $Q(I)$ query result over T. Then the corresponding schema mapping is defined as $\mathcal{M} = (S, T, \Sigma)$ and applying the Chase yields $\text{Chase}_{\mathcal{M}}(I) = I(Q)$. On the other hand, let Q^{-1} be the query inverse to Q, which is also formalized as a set of (s-t) tgds Σ^{-1}. Let further be $\mathcal{M}^* = (T, S, \Sigma^{-1})$ the corresponding schema mapping with I^* instance over S. Then applying the Chase yields $I^* = \text{Chase}_{\mathcal{M}^*}(Q(I))$. Altogether the Chase&Backchase yields: $\text{Chase}_{\mathcal{M}^*}(\text{Chase}_{\mathcal{M}}(I)) = \text{Chase}_{\mathcal{M}^*}(Q(I)) = I^*$. For simplicity, we denote Σ instead of $\mathcal{M}$ and Σ^{-1} instead of $\mathcal{M}^*$ in the following. Additionally, we use Σ_P and Σ_P^{-1} to refer to the dependency set extended by provenance annotations p. For the Chase& Backchase with additional provenance this means $\text{Chase}_{\Sigma_P^{-1}}(\text{Chase}_{\Sigma_P}(I)) = \text{Chase}_{\Sigma_P^{-1}}(Q(I)) = I^*$.

If one extends both the database instance I and the query Q with additional provenance annotations (step 3), these are automatically integrated into the query result $Q(I)$ in the Chase phase. In the Backchase phase, this information is used to reconstruct the minimal sub-database. Again, this is done automatically, provided that the inverse query has been extended by provenance annotations before (step 5).

Combining* Chase *and Provenance. To include additional provenance information in the Chase, we require globally unique provenance IDs. These we construct using the relation name (or its first letter) with the suffix `id` and a continuous integer number. Specifically, the i-th tuple $R(x_{i_1}, ..., x_{i_n})$ of relation R receives the provenance ID R_{id_i}, which is stored as an additional, new attribute in R. So, each tuple of I has the form $R(x_{i_1}, ..., x_{i_n}, R_{\mathrm{id}_i})$.

Additionally, we modify the tgds and s-t tgd by adding provenance IDs to the dependencies body and head. Analogous to the tuples of I, the body's relations receive additional IDs R_{id_i}. These new attributes are also appended to

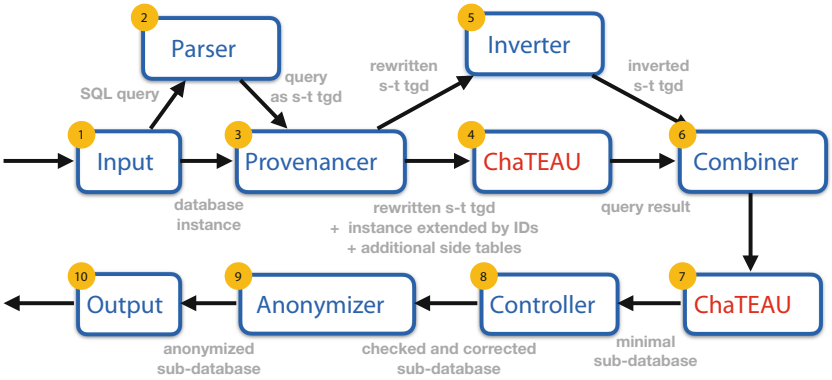

Fig. 1. ProSA pipeline

the result tuple in the head. In summary, this results in dependencies such as $R(x_{i_1}, ..., x_{i_n}, R_{\mathrm{id}_i}) \wedge R'(x_{j_1}, ..., x_{j_m}, R'_{\mathrm{id}_j}) \rightarrow \texttt{result}(r_1, ..., r_k, R_{\mathrm{id}_i}, R'_{\mathrm{id}_j})$. So, the ID attributes correspond to the witness set $\{R_{\mathrm{id}_i}, R'_{\mathrm{id}_j}\}$ [7], which is created by joining the two relations R and R'. Due to the attribute-wise storage of the provenance IDs in the result tuple, duplicates need not be considered.

By extending the source tuples in I and the dependencies with additional provenance IDs, the calculation of witness sets is outsourced to the Chase phase. The evaluation of the ***why***-provenance, i.e. the integration of additional information based on witness sets, can also be outsourced to the Chase. This information is used to reconstruct lost attribute values, null tuples and duplicates.

When the query is inverted, redundant null values and tuples may be created in the Backchase phase. These are subsequently eliminated by global variable substitution. For this, we make use of the key properties of the Provenance IDs internally defined by equality generating dependencies (egds) such as $R(x_{i_1}, ..., x_{i_n}, R_{\mathrm{id}}) \wedge R(x_{j_1}, ..., x_{j_n}, R_{\mathrm{id}}) \rightarrow x_{i_1} = x_{j_1}, ..., x_{i_n} = x_{j_n}$. No further input is required from the user. The egds perform global data cleaning internally. At the end, ProSA returns a minimal sub-database I^* of I, which is computed using the Chase&Backchase extended by additional provenance information.

For queries with highly condensed information, we need to store concrete attribute values. This values are stored in a separate side table, which consists of the attribute value itself as well as the corresponding provenance ID: $R(x_i, R_{\mathrm{id}_i})$. The tables are calculated during the Chase phase and used in the Backchase phase to reconstruct the lost attribute values. The extension is done by the Provenancer by $R(x_{i_1}, ..., x_{i_n}, R_{\mathrm{id}_i}) \rightarrow \texttt{result}(r_1, ..., r_k, R_{r_{\mathrm{id}_1}}) \wedge \texttt{side}(x_{i_j}, R_{\mathrm{id}_i})$. The combination of side tables and ***why***-provenance is sufficient to check the results of SPJU queries extended by aggregation, grouping and more.

Chase *Execution by ChaTEAU.* The Chase is a technique used in a number of data management tasks such as data exchange, data cleaning or answering queries using views [5]. The versatile applicability of the Chase is due to the

fact that one can pass different types of objects and parameters as input. As described in [4], this object can be an instance as needed here, or alternatively a query. ChaTEAU (**Cha**se for **T**ransforming, **E**volving, and **A**dapting databases and queries, **U**niversal Approach) abstracts instances and queries to a general Chase object and parameter. It thus provides a multi-purpose implementation of the Chase. In ProSA, we use ChaTEAU for processing the Chase as well as the Backchase phase.

Anonymizing the Minimal Sub-database. In the ProSA pipe-line, it is feasible to anonymize at different points in time: It can be done in the input, i.e. the source instance I or the query Q can be anonymized. Anonymizing the source database is theoretically possible, but the effort involved is disproportionate. However, anonymizing the query violates reconstructability and can lead to problems in the ProSA pipeline. Anonymizing the intermediate ProSA steps, i.e., the provenance annotations Σ_P as well as the query result $I(Q)$, is not very useful, too. These would make using the ***why***-provenance obsolete. We therefore choose to anonymize the minimal sub-database I^*. As anonymization measure, we use k-anonymity [13] as well as l-diversity, and as anonymization method, generalization using domain generalization hierarchies based on [12].

Limitations of ProSA. The key component of ProSA is the Chase. Hence we are limited to processing (s-t) tgds and/or egds. Thus, queries that cannot be formulated as a set of (s-t) tgds cannot be evaluated with ProSA.

3.2 The Steps of the ProSA Pipeline

1 + **10** ***Input and Output.*** The input—a database instance I and a SQL query Q—and output—the minimal sub-database I^*_{anon}—is organized via a *GUI*. It is the central interface between the user and ProSA. It is composed of five tabs: First of all, we need a connection to the database to be minimized. This can be established via the *DB Configuration* tab. The calculation of the minimal sub-database is then done in the next three tabs (Chase, Backchase and *Anonymizer*). Finally, the *Log* offers additional information about Chase execution such as the results of the single Chase steps after applying an (s-t) tgd or egd. The single ProSA steps can be tracked here, too.

2 ***Parsing SQL Queries.*** ProSA uses the Chase for evaluating queries. However, the Chase deals with dependencies such as egds, tgds or s-t tgds and not with SQL queries, which we want to use as input for a better user experience. Hence, the *Parser's* main task is to transform these SQL queries into a set of (s-t) tgds. It transforms conjunctive SPJU queries, aggregate functions such as `Min/Max`, `Count`, `Sum` and `Avg`, nested queries as well as grouping.

For rendering simple conjunctive queries, an s-t tgd is usually required. So, a SQL query `SELECT` $\psi(\mathbf{x})$ `FROM` $\phi(\mathbf{x})$ corresponds to an s-t tgd $\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$ with $\psi(\mathbf{x})$ query result. However, there are queries formalized as two s-t tgds. This includes, for example, union, nested queries or aggregate functions. For example, the query (`SELECT` $\psi(\mathbf{x})$ `FROM` $\phi_1(\mathbf{x})$) `UNION` (`SELECT` $\psi(\mathbf{x})$ `FROM` $\phi_2(\mathbf{x})$) corresponds to $\phi_1(\mathbf{x}) \rightarrow \psi(\mathbf{x}), \phi_2(\mathbf{x}) \rightarrow \psi(\mathbf{x})$. In the case of Q_1 we get:

$$\texttt{employee}(\text{id}, \text{“Stefan”}) \wedge \texttt{salary}(\text{id}, \text{sal}) \longrightarrow \texttt{result}(\text{“Stefan”}, \text{sal}).$$

3 ***Including Provenance.*** The *Provenancer* prepares the input of the CHASE phase in such a way that provenance information can be collected. For this, he adds global unique provenance IDs to the (s-t) tgds of Σ as well as to the source tuples of I. We thus obtain Σ_P and I_P as provenance extensions of Σ and I. If necessary, additional side tables are created here, too. Afterwards, the Provenances rewrites the (s-t) tgds in Σ_P as described in Sect. 3.1.

In the case of Q_2, the SQL query is transformed into two dependencies. The tgd creates an additional side table `interim`, which stores all intermediate results (see Table 1g). The s-t tgd provides the maximum salaries using the side table as a filter. After that, the Provenancer adds global unique provenance IDS (highlighted in purple) to the s-t tgd of Σ as well as to the source tuples of I. In summary, the associated rewritten query $Q_{2,P}$ is:

$$\texttt{salary}(\text{id}_1, \text{sal}_1, s_{\text{id}_1}) \wedge \texttt{salary}(\text{id}_2, \text{sal}_2, s_{\text{id}_2}) \wedge \text{sal}_1 < \text{sal}_2 \longrightarrow \texttt{interim}(\text{sal}_1, s_{\text{id}_1}), \quad \text{(tgd)}$$

$$\texttt{salary}(\text{id}, \text{sal}, s_{\text{id}}) \wedge \neg\, \texttt{interim}(\text{sal}, s_{\text{id}}) \longrightarrow \texttt{result}(\text{sal}, s_{\text{id}}). \quad \text{(s-t tgd)}$$

And I is extended to $I_P = \{\texttt{salary}(2, 1500), s_{\text{id}_1}), \ldots, \texttt{salary}(2, 1470), s_{\text{id}_5})\}$.

4 ***ChaTEAU* (Chase *Phase).*** In the CHASE phase, *ChaTEAU* determines the query result $Q(I)$ by incorporating the dependencies from Σ_P into the database instance I, i.e. $Q(I) = \text{CHASE}_{\Sigma_P}(I)$. In addition to the query result extended by ***why***-provenance, the CHASE phase provides the results of ***where***- and ***how***-provenance. The provenance types are calculated for the sake of completeness, but are no longer used in the following.

The query results can be found in Table 1c and Table 1d. Note that a result tuple such as `result`(“Stefan”,1470) $\in Q_{1,P}(I_P)$ is stored internally as `result`(Stefan,1470,e_{id_1},s_{id_4}). In this way, duplicates are retained, but not displayed to the user. The same is done for the tuples of $Q_{2,P}(I_P)$.

5 ***Inverting s-t tgds***. In this module, a parsed query Q is inverted. The inverse query Q^{-1} always corresponds to one s-t tgd, which then be processed in the BACKCHASE phase. For inverting the query, we extend the *maximum-extended-recovery-algorithm* of [10] by grouping the inverted dependencies into a minimal number of s-t tgds. In the case of $Q^{-1}_{1,P}$, for example, the Inverter yields:

$$\texttt{result}(\text{“Stefan”}, \text{sal}, e_{\text{id}}, s_{\text{id}}) \longrightarrow \exists\ \text{Id} : \texttt{employee}(\text{Id}, \text{“Stefan”}, e_{\text{id}}) \wedge \texttt{salary}(\text{Id}, \text{sal}, s_{\text{id}}).$$

6 ***Preparing the* Backchase.** The *Combiner* combines the inverse Q_P^{-1} with the query result $Q_P(I_P)$ to obtain a valid input for the BACKCHASE phase.

7 ***ChaTEAU* (Backchase *Phase).*** The technical design of the BACKCHASE phase does not differ from the CHASE phase. Only the input differs, i.e. Σ_P^{-1} instead of Σ_P and $Q(I)$ instead of I. Here, *ChaTEAU* determines the minimal sub-database I^* by incorporating the dependencies from Σ_P^{-1} into the query result $Q(I)$, i.e. $I^* = \text{CHASE}_{\Sigma_P^{-1}}(Q(I))$ with Σ_P^{-1} formalization of Q^{-1}. In doing so, the provenance information P is automatically integrated into I^*.

By applying the Chase, lost attribute values are filled with null values and duplicates are reconstructed. Thus I^* contains regarding to Q_1 the tuples `employee`(η_1,Stefan, e_{id_1}), `employee`(η_2,e_{id_2}), and `employee`(η_3,Stefan, e_{id_2}).

8 ***Proving the Calculated Sub-database and Correcting Duplicates.*** The controller checks whether the calculated minimal sub-database I^* is correct concerning reproducibility of the query result $Q(I_P)$. This is important, because only then the calculated sub-database satisfies the requirements of reproducibility at all and can be co-published along with the research result $Q(I_P)$. For this, the controller calculates $Q(I^*)$ and compares it with $Q(I_P)$ calculated in the Chase phase. If the check fails, the calculated minimal sub-database has been incorrectly determined and an error message is returned.

In addition, the controller eliminates redundant tuples, which can be created in the Backchase phase. For this, the key properties of the provenance IDs are exploited as described in Sect. 3.1 by a directly implemented global variable substitution. So, replacing null values in I^* resp. Q_1 leads to the deletion of the superfluous and wrong tuple `employee`(η_3,Stefan,e_{id_2}) as shown in Table 1e by

$$\texttt{employee}(\mathrm{id}_1, \mathrm{name}_1, e_{\mathrm{id}}) \wedge \texttt{employee}(\mathrm{id}_2, \mathrm{name}_2, e_{\mathrm{id}}) \rightarrow \mathrm{id}_1 = \mathrm{id}_2 \wedge \mathrm{name}_1 = \mathrm{name}_2.$$

9 ***Anonymizing the Sub-database.*** ProSA guarantees an k-anonym and l-diverse anonymized minimal sub-database I^*_{anon}. For this, we anonymize I^* by generalization using previously defined domain generalization hierarchies. The calculation of possible quasi-identifiers is done by ProSA itself.

3.3 Implementing ProSA

ProSA is implemented as a Maven project in Java 11. It provides a user interface that allows connecting a database and answering simple conjunctive queries as well as any query represented as a set of (s-t) tgds. For processing the Chase, ProSA calls ChaTEAU, an implementation of the Chase [4]. ProSA itself performs query parsing, adding provenance information, preparing the Chase execution and interpreting the Chase result. In addition, the minimal sub-database is checked for correctness and then be anonymized by generalization in compliance with k-anonymity [13] and l-diversity. The software, examples, and further information about ChaTEAU and ProSA are available at our Git repositories[1,2].

[1] ChaTEAU repository: https://git.informatik.uni-rostock.de/ta093/chateau-demo.
[2] ProSA repository: https://git.informatik.uni-rostock.de/ta093/prosa-demo.

4 Conclusion

In this article, we introduced the ProSA pipeline for minimizing research data before publishing. Therefore, we combine a variant of the Chase&Backchase with additional provenance such as ***why***-provenance and side tables to reconstruct lost attribute values, null tuples and duplicates. Subsequently we anonymize the calculated sub-database to satisfy possibly occurring privacy aspects.

Acknowledgments. We thank all students involved in implementing ProSA: Leonie Förster, Melinda Heuser, Ivo Kavisanczki, Judith-Henrike Overath, Tobias Rudolph, Nic Scharlau, Tom Siegl, Dennis Spolwind, Anne-Sophie Waterstradt, Anja Wolpers, and Marian Zuska. Also, thanks to Bertram Ludäscher for comments and suggestions.

References

1. Auge, T., Heuer, A.: ProSA—using the CHASE for provenance management. In: Welzer, T., Eder, J., Podgorelec, V., Kamišalić Latifić, A. (eds.) ADBIS 2019. LNCS, vol. 11695, pp. 357–372. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28730-6_22
2. Auge, T., Heuer, A.: Tracing the history of the Baltic sea oxygen level. In: BTW, LNI, vol. P-311, pp. 337–348. Gesellschaft für Informatik, Bonn (2021)
3. Auge, T., Heuer, A.: Testing provenance systems. Technical report CS 01-22, Computer Science Division, University of Rostock (2022)
4. Auge, T., Scharlau, N., Görres, A., Zimmer, J., Heuer, A.: ChaTEAU: a universal toolkit for applying the Chase. https://arxiv.org/abs/2206.01643
5. Benedikt, M., et al.: Benchmarking the chase. In: PODS, pp. 37–52. ACM (2017)
6. Benczúr, A., Kiss, A., Márkus, T.: On a general class of data dependencies in the relational model and its implication problems. Comput. Math. Appl. **21**(1), 1–11 (1991)
7. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: why, how, and where. Found. Trends Databases **1**(4), 379–474 (2009)
8. Deutsch, A., Hull, R.: Provenance-directed chase&backchase. In: Tannen, V., Wong, L., Libkin, L., Fan, W., Tan, W.-C., Fourman, M. (eds.) In Search of Elegance in the Theory and Practice of Computation. LNCS, vol. 8000, pp. 227–236. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41660-6_11
9. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Schema mapping evolution through composition and inversion. In: Bellahsene, Z., Bonifati, A., Rahm, E. (eds.) Schema Matching and Mapping. Data-Centric Systems and Applications. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-16518-4_7
10. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Reverse data exchange: coping with nulls. ACM Trans. Database Syst. **36**(2), 11:1–11:42 (2011)
11. Herschel, M., Diestelkämper, R., Ben Lahmar, H.: A survey on provenance - what for? what form? what from? VLDB J. **26**(6), 881–906 (2017)
12. Han, J., Cai, Y., Cercone, N.: Data-driven discovery of quantitative rules in relational databases. IEEE Trans. Knowl. Data Eng. **5**(1), 29–40 (1993)
13. Samarati, P.: Protecting respondents' identities in microdata release. IEEE Trans. Knowl. Data Eng. **13**(6), 1010–1027 (2001)