ProSA: A provenance system for reproducing query results

Tanja Auge*
tanja.auge@ur.de
University of Regensburg
Germany

ABSTRACT

Good scientific work requires comprehensible, transparent and reproducible research. One way to ensure this is to include all data relevant to a study or evaluation when publishing an article. This data should be at least aggregated or anonymized, at best compact and complete, but always resilient.

In this paper we present ProSA, a system for calculating the minimal necessary data set, called *sub-database*. For this, we combine the Chase — a set of algorithms for transforming databases — with additional provenance information. We display the implementation of provenance guided by the ProSA pipeline and show its use to generate an optimized sub-database. Furhter, we demonstrate how the ProSA GUI looks like and present some applications and extensions.

KEYWORDS

Chase, Data Provenance, Research Data Management, System Demo

ACM Reference Format:
Tanja Auge. 2023. ProSA: A provenance system for reproducing query results. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3543873.3587563

1 INTRODUCTION

One of the basic requirements for scientific work is to obtain comprehensible, transparent and reproducible results. This implies storing and keeping all data involved in a research project/process, i.e., primary data, secondary data, validated and un-validated data, intermediate data, published results, etc. However, a complete reconstruction of all data is often not possible.

In this article, we focus on a (published) research result and its corresponding source data. For this, we demonstrate a technique to verify such results. Our system ProSA automatically determines the (minimal) data set, called *sub-database*, necessary for reproduction or replication. By archiving this sub-database, the reproducibility and replicability of the result (defined after the ACM Policy *Artifact Review and Badging*) should be guaranteed.

Applying the original query to the sub-database will produce a (query) result that is the same or at least similar to the original (query) result. In this way, ProSA completely reconstructs the results of queries with low information loss and at least provides a plausibility check in the case of high information loss.

For a given SQL query and a database instance, ProSA generates a sub-database containing all data needed for the reconstruction of the query result. Since some queries cannot be inverted in the "classical" sense, a complete reconstruction of the source database is not always possible. Classical means that there is no loss of information due to inversion of the query. In some cases, this loss of information may be intentional for privacy reasons, as described in [7]. In other cases, this loss must be accepted because of storage space problems.

To calculate an optimized sub-database we combine the Chase — a set of algorithms for transforming databases — with additional provenance information. Provenance allows us to reconstruct duplicates and dangling tuples that can occur when evaluating conjunctive queries. Additional side tables allow us to store attribute values of highly selective and summarizing queries. This means that ProSA has to fulfill two requirements that at first glance appear to be contradictory:

- minimal respectively privacy aspects;
- maximal respectively reproducibility and replicability.

Thus, to maximize the sub-database we use witness basis and additional side tables. To minimize the sub-database we offer the possibility of a final anonymization. In addition, we take advantage of the "natural" loss of information of many evaluation operators. In this article, we will focus on maximizing the sub-database.

Contribution. In this demo, we present a system for generating the sub-database necessary to reconstruct the result of a conjunctive query. We focus on the integration of provenance and present the ProSA system itself.

Prior Work. Executing the Chase is outsourced to the Maven project ChaTEAU. The tool implements a generalized version of the Standard Chase [5] that abstracts instances and queries to a general Chase object and Chase parameter as described in [4].

The first concept of ProSA is given in [2]. In [1] and [3] we present the ProSA pipeline.

Artifact availability. The source code and a demo video of ProSA are available online[1] along with some real-world examples. An update will be available in time for a live demo. The code of ChaTEAU is available online[2] as well.

Structure. First, we provide some information including the Chase in Section 2. In Section 3, we outline our provenance concept based on the ProSA pipeline. Finally, we conclude with a demonstration of our ProSA system in Section 4.

*The implementation of ProSA was done during my time at the University of Rostock.

WWW '23 Companion, April 30-May 4, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9419-2/23/04.
https://doi.org/10.1145/3543873.3587563

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

2 BASICS AND RELATED WORK

All database evaluations, that are performed in ProSA, are executed by the Chase, a family of algorithms used in a number of data management tasks. For details we refer to [5, 8]. In our case, the Chase evaluates a query Q on a given database instance I and returns the query result $Q(I)$ [1]. The query Q has to be formalized as a set of (s-t) tgds Σ_Q (defined below). Note: Q and Σ_Q describe the same evaluation query; the former as a SQL query and the latter as a set of (s-t) tgds.

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 $\phi(x)$, $\psi(x, y)$ conjunctions of atoms (over a source schema S and/or a target schema T), called *body* and *head*. An *equality generating dependency* (egd) is a dependency that generates equalities. It is a formula of the form $\forall x(\phi(x) \rightarrow x_1 = x_2)$ with x tuple of variables and $\phi(x)$ conjunction of atoms.

Applying the Chase (*chasing*) to a query defined as a set of (s-t) tgds and a database instance I is equal to evaluating a SQL query over I. Since the Chase operates on the relational algebra, chasing (s-t) tgds can lead to duplicates, which can be deleted by additional egds. Chasing egds cleans the database by replacing null values η_i (by other null values η_j or constants c_j). For more details we refer again to [5, 8].

Applying the Chase twice calculates an instance I^* similar to I. We first Chase the source instance I with a query Q and then Chase its result $Q(I)$ with the inverse query Q^{-1} to obtain I^*. Again, the two queries must be formalized as two sets of (s-t) tgds Σ_Q and Σ_Q^{-1}. The resulting instance I^* we call *sub-database*. Furthermore, we say that a sub-database I^* is *correctly constructed* if there is a homomorphism from I^* into the original database I (in short, $I^* \preccurlyeq I$).

To optimize the sub-database we extend the Chase with additional provenance information: ***where***, ***why*** and ***how***. While ***why*** [6, 7] specifies a witness basis that identifies the tuples involved in the query result, ***how*** uses provenance polynomials to specify a calculation rule [9]. In contrast to [6], we define ***where*** as a list of all relevant tuple IDs and their corresponding relation names.

3 PROSA

The core ideas of ProSA can be summarized as follows [1–3]:

1. evaluating a given query Q over a source instance I,
2. inverting Q to Q^{-1},
3. evaluating Q^{-1} over the query result $Q(I)$.

Steps 1 and 3 are done by the Chase [4]. To do so, the SQL query must be parsed, inverted and extended by additional provenance information. Applying the Chase twice we get the sub-database I^*, which is finally checked for correctness, i.e. $I^* \preccurlyeq I$. Additional provenance allows us to optimize the reconstructed sub-database I^* by restoring duplicates, lost attribute values, and dangling tuples.

ProSA is implemented as a Maven project in Java 11. It provides a user interface, part of which is shown in Figure 2. The user formulates a SQL query — conjunctive queries extended by simple aggregate functions — which is internally mapped into a set of (s-t) tgds. To extend our application capabilities, we extend our approach with schema evolution and anonymization [1–3]. Note the additional tabs (`Evolution` and `Anonymizer`) in Figure 2. In this paper, however, we focus on the integration of provenance.

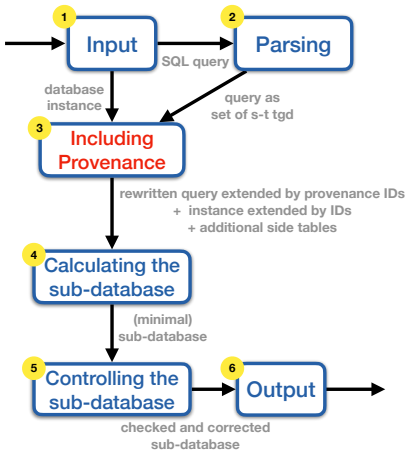

Figure 1: Extraction of the ProSA pipeline

3.1 The ProSA pipeline

The most important steps of the ProSA pipeline are summarized in Figure 1: First, we establish a connection to the database (1). Then, the SQL query Q is parsed into a set of (s-t) tgds (2). After that the query Q and the database instance I are extended by additional provenance information (Q_P and I_P, (3)). Evaluating the query ($Q_P(I_P)$) and recalculating the sub-database I^* are done in step (4) by applying the Chase to Q_P respectively its inverse Q_P^{-1} as described in [4]. Finally ProSA controls the calculated sub-database I_P^* ((5)) and prints it in a separate file ((6)).

3.2 Including Provenance

For integrating provenance we extend each tuple $\mathtt{R}(x_{i_1}, ..., x_{i_n})$ of I with a globally unique *provenance ID* p_i, which consists of the relation name (or its first letter) with the suffix "id" and a continuous integer number $i \in \mathbb{N}$. So, each tuple of I_P has the form $\mathtt{R}(x_{i_1}, ..., x_{i_n}, \mathtt{R}_{\mathtt{id}_i})$ as described in Algorithm 1. Analogously we extend the atoms of the bodies and heads of a dependency such as $\mathtt{R}(x_{i_1}, ..., x_{i_n}) \wedge \mathtt{R'}(x_{j_1}, ..., x_{j_m}) \rightarrow \mathtt{result}(r_1, ..., r_k)$ by adding the provenance IDs to $\mathtt{R}(x_{i_1}, ..., x_{i_n}, \mathtt{R}_{\mathtt{id}_i}) \wedge \mathtt{R'}(x_{j_1}, ..., x_{j_m}, \mathtt{R'}_{\mathtt{id}_j}) \rightarrow \mathtt{result}(r_1, ..., r_k, \mathtt{R}_{\mathtt{id}_i}, \mathtt{R'}_{\mathtt{id}_j})$.

By extending the source tuples in I and the query Q with additional provenance IDs to I_P and Q_P, we are able to calculate the ***where***-, ***why***- and ***how***-provenance of a query result (by chasing Q_P with I_P). After that, we use the provenance results to calculate of the sub-database I_P^* (by chasing Q_P^{-1} with $Q_P(I_P)$).

Internally, I always corresponds to I_P and Q to Q_P and the sub-database I_P^* is calculated using the ***why***-provenance. In addition, the query result $Q(I)$ and the sub-database I^* are also computed without additional provenance information. To display them, click on the button *Without ProSA provenance*.

Evaluating the inverse query Q_P^{-1} may create redundant null values and/or tuples, which can be eliminated by egds such as $\mathtt{R}(x_{i_1}, ..., x_{i_n}, \mathtt{R}_{\mathtt{id}}) \wedge \mathtt{R}(x_{j_1}, ..., x_{j_n}, \mathtt{R}_{\mathtt{id}}) \rightarrow x_{i_1} = x_{j_1}, ..., x_{i_n} = x_{j_n}$. The equality generating dependencies (egds) describe the key properties of the provenance IDs. They are determined automatically by ProSA and do not need to be defined by the user.

Algorithm 1 Integrating provenance to ProSA

Require: set of dependencies Σ, a database instance I
Ensure: extended instance I_P, extended dependency set Σ_P

for all $\mathsf{R} \in I$ **do**
  extend $\mathsf{R}(x_{i_1}, ..., x_{i_n})$ to $\mathsf{R}(x_{i_1}, ..., x_{i_n}, \mathsf{R}_{\mathrm{id}_i}) \in I_P$
for all $\sigma \in \Sigma$ **do**
  if $\mathsf{R}(x_{i_1}, ..., x_{i_n}, \mathsf{R}_{\mathrm{id}_i})$ exists in I_P **then**
    extend each atom $\mathsf{R}(x_{i_1}, ..., x_{i_n})$ in the body of σ to $\mathsf{R}(x_{i_1}, ..., x_{i_n}, \mathsf{R}_{\mathrm{id}_i}) \in \Sigma_P$
    extend $\mathtt{result}(r_1, ..., r_k)$ in the head of σ to $\mathtt{result}(r_1, ..., r_k, \mathsf{R}_{\mathrm{id}_i})$ **or**
    extend $\mathtt{result}(r_1, ..., r_k, \mathsf{R}_{\mathrm{id}_j})$ in the head of σ to $\mathtt{result}(r_1, ..., r_k, \mathsf{R}_{\mathrm{id}_j}, \mathsf{R}_{\mathrm{id}_i})$

3.3 Evaluating the Provenance Results

By applying the Chase on I_P and Q_P (4), ProSA generates three types of provenance results for each tuple of $Q_P(I_P)$: a list of provenance IDs and their corresponding relation names (***where***), a witness basis (***why***) and a provenance polynomial (***how***). The result tuple $\mathtt{result}(r_1, ..., r_k, \mathsf{R}_{\mathrm{id}_i}, \mathsf{R'}_{\mathrm{id}_j})$ has, for example, the following provenance results:

- a list of provenance IDs: $P_{\textbf{\textit{where}}_{\text{tuple}}} = \{\mathsf{R}_{\mathrm{id}_i}, \mathsf{R'}_{\mathrm{id}_j}\}$;
- a list of relation names: $P_{\textbf{\textit{where}}_{\text{relation}}} = \{\mathsf{R}, \mathsf{R'}\}$;
- a witness basis: $P_{\textbf{\textit{why}}} = \{\{\mathsf{R}_{\mathrm{id}_i}, \mathsf{R'}_{\mathrm{id}_j}\}\}$;
- a provenance polynomial: $P_{\textbf{\textit{how}}} = \mathsf{R}_{\mathrm{id}_i} \cdot \mathsf{R'}_{\mathrm{id}_j}$.

They are calculated as follows: Let $\phi(x, p_x) \rightarrow \exists y : \psi(x, y, p)$ be formulated as (s-t) tgd and extended by additional provenance information. To answer ***where***, we store all provenance IDs of the result in a set P, i.e. $P_{\textbf{\textit{where}}_{\text{tuple}}} = \{p \mid p \text{ is provenance ID in } \psi(x, y, p)\}$. Additionally, we list the corresponding relation names in a second list $P_{\textbf{\textit{where}}_{\text{relation}}} = \{R \mid R \text{ is relation name to provenance ID } p\}$. For the witness basis of the ***why***-provenance and the provenance polynomials of the ***how***-provenance we first have to determine the witnesses w_x. They consist of one combination of provenance IDs p_x for all attributes $x \in \psi(x, y, p)$ such that $x_i \in x$ and $p_{x_i} \in \phi(x, p_x)$. The witness basis itself consists of all witnesses belonging to x. The provenance polynomial is equal to the sum of the witnesses, whereby each witness corresponds to the product of its individual provenance IDs. The procedure is summarized in Algorithm 2.

For determining the sub-database I_P^* in (4) we only need one type of provenance result. Since the additional information of the provenance polynomials is lost during the final anonymization, we chose the witness basis of the ***why***-provenance as provenance concept for determining I_P^*. Furthermore, the storage and processing of provenance polynomials is much more complicated than that of witness basis. If concrete attribute values have to be recorded, we save them separately. In this way, we can save storage space and keep the provenance calculation fast and simple.

For queries with highly condensed information such as aggregate functions or joins with dangling tuples, we need to store concrete attribute values in a separate *side table*. An entry of a side table consists of the attribute value and a provenance ID: $\mathsf{R}(x_i, \mathsf{R}_{\mathrm{id}_i})$. Side tables are calculated by evaluating SQL queries which can not be formalized as a single s-t tgd such as `UNION` or aggregation.

Algorithm 2 ***where***-, ***why***- and ***how***-provenance in ProSA

Require: a set of dependencies Σ_P, a database instance I_P both extended by provenance
Ensure: provenance results for ***where***, ***why*** and ***how***

for all $\sigma \in \Sigma_P$ **do**
  if $\sigma : \phi(x, p_x) \rightarrow \exists y : \psi(x, y, p)$ is an (s-t) tgd **then**
    if ***where*** **then**
      $P_{\textbf{\textit{where}}} = \{p \mid p \text{ is provenance ID in } \psi(x, y, p)\}$
    if ***why*** **then**
      for all $x \in \psi(x, y, p)$ **do**
        witness $w_x = \{p_{x_i} \mid x_i \in x \wedge p_{x_i} \in \phi(x, p_x)\}$
      $P_{\textbf{\textit{why}}} = \{w_x \mid w_x \text{ is witness of } x\}$
    if ***how*** **then**
      for all $p_x \in w_x$ **do**
        addends $a_x = \Pi_i p_{x_i}$
      $P_{\textbf{\textit{how}}} = \sum_x a_x$

In summary, with ***why***-provenance and side tables ProSA is able to reconstruct lost duplicates, lost attribute values as well as dangling tuples. This maximizes the sub-database in terms of reproducibility and replicability. For minimizing the sub-database regarding privacy aspects we refer to [1].

3.4 Evaluating Aggregate Functions

ProSA also evaluates SQL queries with simple aggregate functions such as `MAX`/`MIN`, `COUNT`, `SUM` and `AVG`. For this, we use provenance IDs t to order the tuples of a source relation $\mathsf{R}(a, b, t)$ in a second, extended relation $\mathsf{R_S}(x, a, b, t)$. Based on this, we can, for example, sum up a column by storing the intermediate results in additional side tables $\mathsf{H}(c, x)$. The auxiliary relation $\mathsf{S}(t)$ ensures that each tuple is added only once. In summary, this can be formalized in a set of five (s-t) tgds. The most important one is: $\mathsf{R_S}(x, a, b, t) \wedge \neg\mathsf{S}(t) \wedge \mathsf{H}(c, x-1) \rightarrow \mathsf{S}(t) \wedge \mathsf{H}(f(b, c), x) \wedge f(b, c) = b + c$.

3.5 Applications and Extensions of ProSA

ProSA enables us to calculated a sub-database I^* including all relevant information to replicate or reproduce a given query result $Q(I)$. The reconstructed query result $Q(I^*)$ can always be mapped homomorphically to the original query result $Q(I)$. I.e. $Q(I)$ can either be reconstructed in its entirety, or at least checked for plausibility.

ProSA already handles SQL queries with aggregate functions. A corresponding provenance concept also exists. However, their concrete implementation is still in progress.

4 DEMONSTRATION

In our demonstration, we will show how to calculate the sub-database for simple SQL queries. In particular, we will focus on the provenance questions ***where***, ***why*** and ***how*** and how this affects the resulting sub-database. To do so, we will display the different features of the ProSA system using various examples. Furthermore, we demonstrate the challenges of calculating the sub-database.

Interacting with ProSA is possible in several ways: to start the process, a database connection to a PostgreSQL database is required. For integrating schema evolution and anonymization ProSA excepts additional input files. Examples can be found online[1].

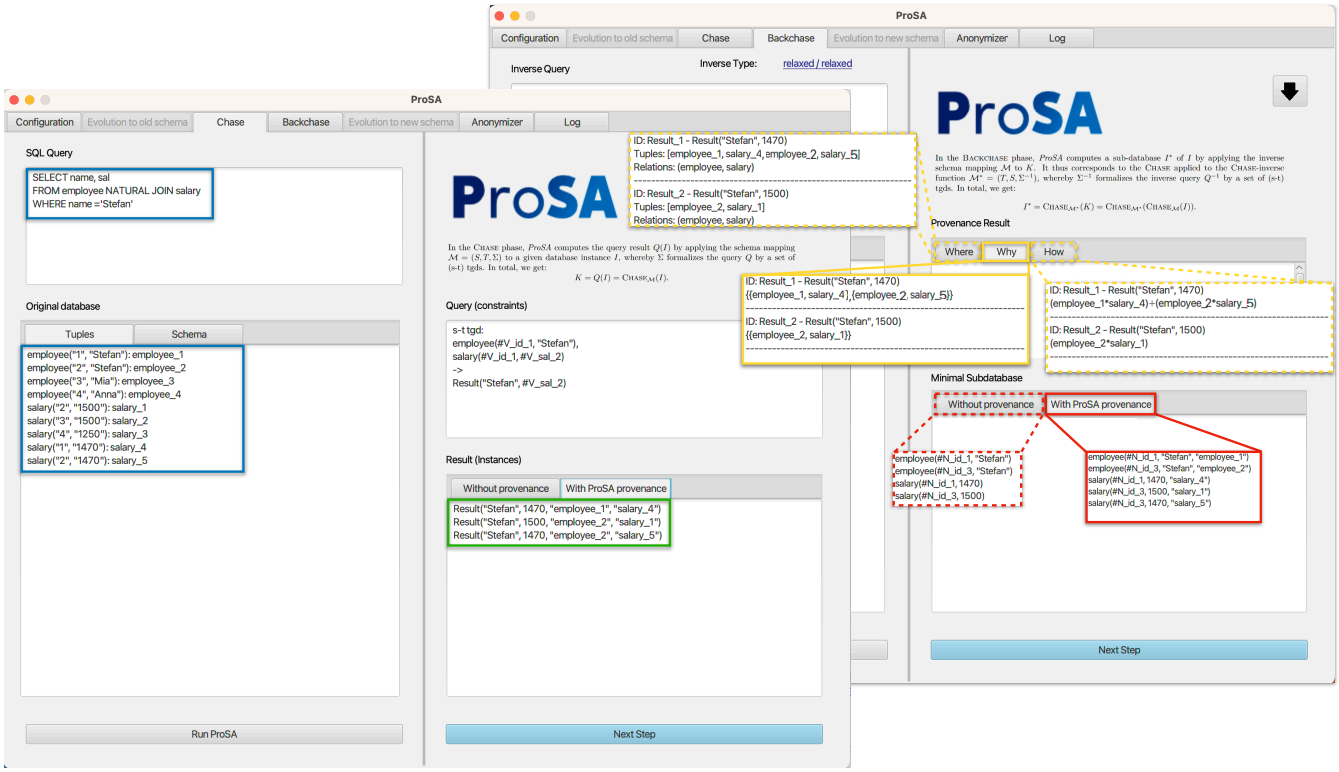

Figure 2: Chase- and Backchase-tab of the ProSA GUI for calculating the sub-database. Input: SQL query and source instance (blue). Intermediate: query result (green). Output: sub-database (red) and additional provenance information (yellow).

The evaluation queries are defined and evaluated in the Chase- and Backchase-tab of the ProSA GUI, shown in Figure 2. The other tabs serve the evolution and anonymization as well as the input and logging of the ProSA process. Each window (except the first and the last) is separated into two parts: the input on the left sight — SQL query (blue), original database (blue), and inverse query (not seen in Figure 2) — and the output on the right sight — parsed query, query result (green), provenance results (yellow) and (minimal) sub-database (red).

Let, for example, I be the database instance given in Figure 2 and Q the SQL query `SELECT name, sal FROM employee NATURAL JOIN salary WHERE name='Stefan'`. Inverting the result tuple `('Stefan',1470)` with witness basis {{`employee_1`, `salary_4`}, {`employee_2`,`salary_5`}} calculates the three tuples `employee(#N_id_1,'Stefan')`, `salary(#_id_1,1470)` and `salary(#_id_3, 1470)`, whereby the latter can only be reconstructed based on provenance. See the two (dashed or solid) red sub-databases in Figure 2. Finally, the results can be downloaded via the "download" button.

5 CONCLUSION AND FUTURE WORK

Given conjunctive queries including simple aggregate functions, our system ProSA determines the (minimal) data set required to reproduce or replicate the query result. ProSA can be extended to arbitrary queries by implementing an appropriate parser, provided that a transformation into a set of (s-t) tgds is possible.

ACKNOWLEDGMENTS

The implementation of ProSA was done during my time at the University of Rostock. Many thanks to Andreas Heuer from the University of Rostock for supervising this project and to all the students involved in the implementation of 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.

REFERENCES

[1] Tanja Auge, Moritz Hanzig, and Andreas Heuer. 2022. ProSA Pipeline: Provenance Conquers the Chase. In *ADBIS (Short Papers) (CCIS, Vol. 1652)*. Springer, 89–98.
[2] Tanja Auge and Andreas Heuer. 2019. ProSA — Using the CHASE for Provenance Management. In *ADBIS (LNCS, Vol. 11695)*. Springer, 357–372.
[3] Tanja Auge and Andreas Heuer. 2021. Tracing the History of the Baltic Sea Oxygen Level. In *BTW (LNI, Vol. P-311)*. GI, Bonn, 337–348.
[4] Tanja Auge, Nic Scharlau, Andreas Görres, Jakob Zimmer, and Andreas Heuer. 2022. ChaTEAU: A Universal Toolkit for Applying the Chase. (2022). https://arxiv.org/pdf/2206.01643.pdf
[5] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *PODS*. ACM, 37–52.
[6] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. 2001. Why and Where: A Characterization of Data Provenance. In *ICDT (LNCS, Vol. 1973)*. Springer, 316–330.
[7] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Found. Trends Databases* 1, 4 (2009), 379–474.
[8] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. 2011. Schema Mapping Evolution Through Composition and Inversion. In *Schema Matching and Mapping*. Springer, 191–222.
[9] Todd J. Green and Val Tannen. 2017. The Semiring Framework for Database Provenance. In *PODS*. ACM, 93–99.