Demonstration of ProvSQL Update Provenance through Temporal Databases

Albert Widiaatmaja
National University of Singapore
Singapore, Singapore
w.albertariel@u.nus.edu

Belkis Djeffal
CRIStAL, Univ. Lille, Inria, CNRS, Centrale Lille
Lille, France
belkis.djeffal@inria.fr

Ashish Dandekar
National University of Singapore
Singapore, Singapore
dcsashi@nus.edu.sg

Pierre Senellart
DI ENS, ENS, PSL University, CNRS, Inria & IUF
& CNRS@CREATE & IPAL, CNRS
Paris, France & Singapore
pierre@senellart.com

Abstract

ProvSQL extends the PostgreSQL database management system by integrating advanced support for *(m-)semiring* provenance and uncertainty management in the form of a PostgreSQL extension. In this demonstration, we further enhance ProvSQL by enabling provenance tracking for update operations (`DELETE`, `INSERT`, `UPDATE`). We illustrate the practical utility of update provenance by implementing a temporal database capable of standard operations, including time travel (inspecting past database states), history tracking (monitoring tuple states over time), and undo (reversing previous updates). These features rely on a provenance formalism based on the *union-of-intervals* m-semiring. Additionally, we emphasize a key advantage of using semiring-based provenance model: its generality allows the same semiring structure to seamlessly support various applications, such as probabilistic databases, by simply modifying the semiring definition.

CCS Concepts

• **Theory of computation → Data provenance**; • **Information systems → Temporal data**.

Keywords

Provenance, temporal databases, semiring provenance, updates

ACM Reference Format:
Albert Widiaatmaja, Belkis Djeffal, Ashish Dandekar, and Pierre Senellart. 2025. Demonstration of ProvSQL Update Provenance through Temporal Databases. In *ProvenanceWeek (PW' 25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3736229.3736253

1 Introduction

Data provenance helps address meta-questions related to query results, such as why a particular result was obtained, tracing the origins of values within the result, and explaining the transformation process that led to the final output [15]. A *semiring* is an algebraic structure which provides a formalism of data provenance in relational databases [8, 15]. Semirings can be applied to annotate positive relational algebra operations in relational databases: selection, renaming, projection, union, and cross product (or join) [15]. Semirings with monus (also known as *m-semirings*) [6] have been introduced to capture non-monotone relational algebra operations such as set difference, extending beyond positive operations [15, 16].

The ProvSQL project [14, 17, 18] supports (m-)semiring provenance computation for retrieval (`SELECT`) queries in SQL, implemented using provenance circuits which are arithmetic circuits where the gates correspond to the operators of (m-)semirings [15]. In this demonstration, we extend ProvSQL to keep track of provenance for updates, enabling various real-world use cases [4]: it enhances explainability, supports the evaluation of hypothetical scenarios, enables verification of modifications, and facilitates access control and trust management, among others.

We also showcase how to use provenance for updates to implement a temporal database within the ProvSQL system. Temporal databases manage data that changes over time by associating records with their *valid time* – timestamp ranges that indicate when the data were considered true – allowing temporal queries such as retrieving past states or tracking data changes over time [9, 12]. However, most existing temporal database systems do not natively capture provenance beyond storing historical states. Provenance adds an additional layer of insight by not only recording what data existed at a given time, but also how it came to be – through sequences of inserts, updates, and deletes [11]. In this work, we achieve this by extending ProvSQL to support provenance for updates and evaluating the provenance circuits in the *union-of-intervals* semiring. This approach enables both temporal queries and undo functionality within a formal provenance framework. Additionally, this approach offers generality: we can model alternative applications by simply substituting different semirings, such as probabilistic databases using the Boolean semiring [15].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
PW' 25, Berlin, Germany
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1941-7/2025/06
https://doi.org/10.1145/3736229.3736253

2 Related Work

Several provenance management systems have been proposed, including Trio [1], DBNotes [3], Mondrian [5], Perm [7], Orchestra [10], and Smoke [13]. These systems typically formalize provenance using semiring annotations (or, for older systems, some provenance framework that can be seen as semiring annotations) and relational encodings, and compute it through techniques such as query rewriting and instrumentation. To date, GProM [2] is the only usable system that implements provenance capture for updates. GProM is a provenance middleware that uses query rewriting over algebraic representations to compute provenance for queries, updates, transactions, and cross-transaction operations. It introduces the framework of multi-version semirings (MV-semirings), which extend conventional semiring-based provenance models by incorporating version annotations. These annotations record how tuples are derived and modified through sequences of insertions, updates, and deletions. GProM reconstructs update provenance on demand by translating past operations into annotated queries, using audit logs and temporal queries. GProM also supports provenance for transactions, but its approach is based on modeling and tracking updates as the fundamental units through which provenance is captured and propagated across transactional executions.

3 Methodology

We describe our methodology for update tracking. The corresponding code is available as open source as part of ProvSQL [17].

3.1 Implementation of Update Provenance

Based on the approach proposed in [4], provenance for updates can be supported by annotating tuples with operations that capture the type of modification. In the ProvSQL system, we implemented these annotations through provenance circuit gates (see [14] for a detailed description of provenance circuits in ProvSQL) as follows:

- `DELETE`: For every tuple that is deleted, we create a monus ($\ominus$) gate and move the deleted tuple to be a child of this gate. No tuples are removed from the table.
- `INSERT`: For every tuple that is inserted, we create a times ($\otimes$) gate and move the inserted tuple to be a child of this gate.
- `UPDATE`: For each old tuple, we create a monus ($\ominus$) gate and move the old tuple to be a child of this gate and for each new tuple, we create a times ($\otimes$) gate and move the new tuple to be a child of this gate. Again, no tuples are removed from the table.

Each times or monus gate corresponding to an update operation has two children: one for the affected tuple and one for the update operation. The unique provenance token associated with each update operation is recorded in a `update_provenance` table, which also stores metadata including the query type, the executing user, and a timestamp. We implement this approach via statement-level triggers on `DELETE`, `INSERT`, and `UPDATE` queries, described in more detail below.

Example 3.1. Consider the table t in Table 1. In the provenance circuit, a and b are gates which represent tuples with id 0 and 1 respectively. Suppose we execute the following delete operation:

id	provenance
0	a
1	b

Table 1: Table t

Algorithm 1 Trigger Statement for `DELETE` Query

1: Generate a provenance token `c` representing this operation.
2: Store query metadata in the `update_provenance` table.
3: **for all** deleted tuples t **do**
4: Replace the gate representing t with a monus gate.
5: The monus gate has two children:
- Left child: the gate representing the original tuple t.
- Right child: the gate representing the delete query.

6: **end for**

```
DELETE FROM t;
```

We generate a provenance token c to represent this `DELETE` operation and store it in a dedicated `update_provenance` metadata table, as shown in Table 2. Metadata includes the SQL code of the update operation, the user performing it, its timestamp, etc.

provenance	query_type	other_metadata
c	delete	...

Table 2: `update_provenance` table after deletion

Each deleted tuple will be updated as in Algorithm 1, yielding the updated table t in Table 3. In the provenance circuit, gates a and b are replaced by d, a monus gate with left child a and right child c, and e, a monus gate with left child b and right child c, respectively.

id	provenance
0	$d = a \ominus c$
1	$e = b \ominus c$

Table 3: Table t after deletion

Example 3.2. Suppose now we execute the following insert operation on an empty table t:

```
INSERT INTO t(id) VALUES (0), (1);
```

We generate a provenance token c to represent this `INSERT` and store it in the `update_provenance`. Each inserted tuple will be updated as in Algorithm 2, yielding the updated table t in Table 4. In the provenance circuit, gates a and b are replaced by d, a times gate with left child a and right child c, and e, a times gate with left child b and right child c, respectively.

Example 3.3. Suppose we execute the following update operation on table t in Table 1.

```
UPDATE t SET id = 2 WHERE id = 0;
```

We generate a provenance token c for this `UPDATE` and record it in the `update_provenance` table. Each affected old and new tuple will be updated as in Algorithm 3, yielding the updated table t in Table 5. In the provenance circuit, gate a is replaced by d, a monus gate with left child a and right child c, and e, a times gate with left child b and right child c, respectively.

Algorithm 2 Trigger Statement for `INSERT` Query

1: Generate a provenance token `c` representing this operation.
2: Store query metadata in the `update_provenance` table.
3: **for all** inserted tuples t **do**
4: Replace the gate representing t with a times gate.
5: The times gate has two children:
- Left child: the gate representing the original tuple t.
- Right child: the gate representing the insert query.

6: **end for**

id	provenance
0	$d = a \otimes c$
1	$e = b \otimes c$

Table 4: Empty Table t after insertion

Algorithm 3 Trigger Statement for `UPDATE` Query

1: Generate a provenance token `c` representing this operation.
2: Store query metadata in the `update_provenance` table.
3: **for all** tuples t to be updated **do**
4: *// Handle the removal of old tuples*
5: Replace the gate representing t with a *monus* gate.
6: The monus gate has two children:
- Left child: the gate representing the original tuple t.
- Right child: the gate representing the update.

7: *// Handle the insertion of new tuples*
8: Generate the updated tuple t'.
9: Replace the gate representing t' with a *times* gate.
10: The times gate has two children:
- Left child: the gate representing the original tuple t.
- Right child: the gate representing the update.

11: **end for**

id	provenance
0	$d = a \ominus c$
1	b
2	$e = a \otimes c$

Table 5: Table t after update

3.2 Implementation of `Undo`

By supporting provenance for updates, we are also able to implement an `undo` operation, which reverts the effect of a previous update. How `undo` is implemented in ProvSQL is described in Algorithm 4. The key step is the `replace_the_circuit` rewriting. This function traverses the provenance circuit of a row and replaces any occurrences of c with $c \ominus u$. Figure 1 illustrates how a single occurrence of c in the provenance circuit is replaced by $c \ominus u$.

3.3 Temporal Databases through Provenance

We implement a temporal database by using the union-of-intervals semiring to interpret the provenance circuit. This union-of-intervals semiring U is the set of finite unions of pairwise-disjoint intervals, representing validity periods of database tuples. Formally, the union-of-intervals semiring is defined as the algebraic structure $(U, \cup, \cap, \emptyset, \{]-\infty, +\infty[\})$, where:

Algorithm 4 `Undo`

Require: c: the provenance token of the update to undo
1: Generate a new provenance token `u` representing this undo operation.
2: Store query metadata in the `update_provenance` table.
3: **for all** tables T in the database with provenance **do**
4: **for all** rows r in T **do**
5: $x \leftarrow r.$`provsql`
6: $x' \leftarrow$ `replace_the_circuit`(x, c, u)
7: Update $r.$`provsql` in T from x to x'.
8: **end for**
9: **end for**

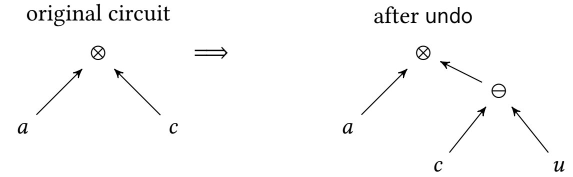

Figure 1: Replacing c with $(c \ominus u)$ in the provenance circuit

- **Plus ($\oplus$)** The $\oplus$ operation is the union of unions of intervals. Given two elements $A, B \in U$, we have: $A \oplus B = A \cup B$. Its neutral element is the empty set $\emptyset$.
- **Times ($\otimes$)** The $\otimes$ operation is the intersection of unions of intervals. Given two elements $A, B \in U$, we have: $A \otimes B = A \cap B = \cup_{a \in A, b \in B} a \cap b$. Its neutral element is the universal singleton interval $\{]-\infty, +\infty[\}$, spanning all possible time instants.

We extend this semiring structure by including a **monus operation** ($\backslash$) defined as: $A \setminus B = \cup_{a \in A, b \in b} a \cap \overline{b}$ where $\overline{b} =]-\infty, +\infty[\setminus b$. The resulting structure $(U, \cup, \cap, \setminus, \emptyset, \{]-\infty, +\infty[\})$ forms an **m-semiring**.

The union-of-intervals semiring is implemented in ProvSQL using a set of User-Defined Functions (UDFs), which enable provenance tracking for temporal queries. These functions operate on timestamp multirange fields (`tstzmultirange` in PostgreSQL). In a provenance circuit context, we interpret the validity of a gate corresponding to a tuple (in any table) without any other annotation as $\{]-\infty, +\infty[\}$ (the neutral element for the $\otimes$ operation in the semiring) and a gate representing an update as $\{[t, +\infty[\}$ where t is the timestamp of the update operation recorded in `update_provenance`.

The following example illustrates how the temporal database implementation tracks the validity intervals of database tuples through a sequence of insertions, deletions, and updates.

Suppose we run the following queries:

```
CREATE TABLE test (id INT);
SELECT add_provenance('test');

INSERT INTO test VALUES (1), (2), (3);
DELETE FROM test WHERE id = 2;
UPDATE test SET id = 4 WHERE id = 3;

SELECT *, union_intervals(
  provenance(),
  'time_validity'
```

id	union_intervals	provsql
1	{["2025-01-31 07:22:41.074735+00",)}	6bb1090e-...
2	{["2025-01-31 07:22:41.074735+00", "2025-01-31 07:23:53.652126+00")}	19014d56-...
3	{["2025-01-31 07:22:41.074735+00", "2025-01-31 07:24:23.929575+00")}	8c09ee82-...
4	{["2025-01-31 07:24:23.929575+00",)}	222faf52-...

Table 6: `union_intervals` example

```
) FROM test;
```

We obtain the result in Table 6; the content of the `provsql` attribute is (abbreviated) unique identifiers referencing the gates of the provenance circuit. We see that `union_intervals` returns the multirange representing the valid time of each tuple. We observe: (1) the tuple of id 1 is valid from its insertion timestamp onwards, with no subsequent deletion or update, (2) the tuple of id 2 was valid from insertion until its deletion at timestamp `2025-01-31 07:23:53.652126+00`, (3) the tuple of id 3 was valid from insertion until it was updated at `2025-01-31 07:24:23.929575+00` and (4) the tuple of id 4 was created through an update of the tuple of id 3, thus becoming valid from the update timestamp onward.

3.4 Temporal Database Operations

Using the valid time generated by `union_intervals`, we can support the following temporal query functions, standard in temporal databases:

- `get_valid_time` returns the valid time interval of a given tuple in a table, indicating when the tuple was logically present in the database.
- `timetravel` returns the entire state of a table at a specified timestamp, allowing users to inspect past versions.
- `timeslice` returns all tuples that were valid within a given time range, supporting queries over historical windows.
- `history` reveals the sequence of changes applied to a specific tuple, showing how its values evolved over time.

Additionally, we can support `undo` functionality which reverts the effect of an update.

4 Demonstration Scenario

We demonstrate the use of provenance for keeping tracks of updates and implementing temporal data support on an example database of government ministers per country over time. We retrieve information from Wikidata through its SPARQL endpoint[1] about government ministers in different countries (e.g., France, Singapore), along with some meta-information about them (date of birth and death, gender, political party, title of their position) along with the validity period of every appointment as a minister. The dataset and example ProvSQL queries are available from https://provsql.org/temporal_demo.

The user is shown the structure of the database, and how provenance is kept track of in the special `provsql` attribute of each relation and query result. Then, the result of evaluating the provenance in the unions-of-intervals semiring is demonstrated: it can be used, for instance, to determine all political appointments of a given individual. For example, the following SQL query:

```
SELECT position, union_intervals(provenance(),
  'time_validity') valid
FROM person JOIN holds ON person.id=holds.id
WHERE name LIKE '% Bayrou' ORDER BY valid;
```

returns all government positions held by François Bayrou, along with the time intervals when he held such positions.

We then showcase more advanced features of temporal databases:

- with `timetravel`, it is possible to have the composition of the government of a country at any date in the past (provided the information is available in Wikidata);
- with `timeslice`, one can ask for the list of persons having had a ministry within a parliamentary or presidential cycle;
- with `history`, one can have a list of persons having had the same title over time.

We also simulate a government change by a series of update operations simulating the resigning of selected individuals and appointment of new individuals. Thanks to the tracking of update operations, without needing to set any explicit temporal validity for the new information or to explicitly change the temporal validity of old information, all previous temporal operations will return correct results. We can now test the `undo` operation: by undoing these update operations, we get back the composition of the government before they occurred, with time validity properly set.

Finally, we move away from the temporal m-semiring and consider other applications of ProvSQL: if we assign a probabilistic confidence value to each update (e.g., depending on who was the author of the update, reflecting the confidence in this source), ProvSQL can compute the probability of the output of any query that takes into account the probability on update operations.

5 Conclusion

We demonstrated how ProvSQL can be flexibly extended to support tracking of update operations, inspired by the formalism developed in [4]. A natural and powerful application of update tracking is temporal databases, which are supported by simply evaluating provenance in the union-of-intervals semiring.

There are two main limitations in ProvSQL's current provenance tracking, which we intend to cover in future work. The first is that only *hyperplane update queries*, the formalism of [4], are supported; updates whose provenance depends on external tuples are not currently supported. Second, there is no support for transactions and concurrency control, which is a major limitation for specific applications.

Acknowledgments

This research is part of the program DesCartes (https://descartes.cnrsatcreate.cnrs.fr/) and is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) program.

[1] https://query.wikidata.org/

References

[1] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. 2006. Trio: A System for Data, Uncertainty, and Lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim (Eds.). ACM, 1151–1154. http://dl.acm.org/citation.cfm?id=1164231

[2] Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. 2018. GProM - A Swiss Army Knife for Your Provenance Needs. *IEEE Data Eng. Bull.* 41, 1 (2018), 51–62. http://sites.computer.org/debull/A18mar/p51.pdf

[3] Deepavali Bhagwat, Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya. 2005. An annotation management system for relational databases. *VLDB J.* 14, 4 (2005), 373–396. doi:10.1007/S00778-005-0156-6

[4] Pierre Bourhis, Daniel Deutch, and Yuval Moskovitch. 2020. Equivalence-Invariant Algebraic Provenance for Hyperplane Update Queries. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 415–429. doi:10.1145/3318464.3380578

[5] Floris Geerts, Anastasios Kementsietsidis, and Diego Milano. 2006. *i*MONDRIAN: A Visual Tool to Annotate and Query Scientific Databases. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3896)*, Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Böhm, Alfons Kemper, Torsten Grust, and Christian Böhm (Eds.). Springer, 1168–1171. doi:10.1007/11687238_84

[6] Floris Geerts and Antonella Poggi. 2010. On database query languages for K-relations. *J. Appl. Log.* 8, 2 (2010), 173–185. doi:10.1016/J.JAL.2009.09.001

[7] Boris Glavic, Renée J. Miller, and Gustavo Alonso. 2013. Using SQL for Efficient Generation and Querying of Provenance Information. In *In Search of Elegance in the Theory and Practice of Computation - Essays Dedicated to Peter Buneman (Lecture Notes in Computer Science, Vol. 8000)*, Val Tannen, Limsoon Wong, Leonid Libkin, Wenfei Fan, Wang-Chiew Tan, and Michael P. Fourman (Eds.). Springer, 291–320. doi:10.1007/978-3-642-41660-6_16

[8] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, Leonid Libkin (Ed.). ACM, 31–40. doi:10.1145/1265530.1265535

[9] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and Richard Thomas Snodgrass. 1992. A glossary of temporal database concepts. *SIGMOD Rec.* 21, 3 (09 1992), 35–43. doi:10.1145/140979.140996

[10] Grigoris Karvounarakis, Todd J. Green, Zachary G. Ives, and Val Tannen. 2013. Collaborative data sharing via update exchange and provenance. *ACM Trans. Database Syst.* 38, 3 (2013), 19. doi:10.1145/2500127

[11] Ekaterina Kuleshova. 2011. Provenance in Temporal Databases. Facharbeit in Informatik, University of Zürich. Supervised by Prof. Dr. M. Böhlen and A. Dignös..

[12] Gultekin Özsoyoglu and Richard T. Snodgrass. 1995. Temporal and Real-Time Databases: A Survey. *IEEE Trans. Knowl. Data Eng.* 7 (1995), 513–532. https://api.semanticscholar.org/CorpusID:14140377

[13] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-grained Lineage at Interactive Speed. *Proc. VLDB Endow.* 11, 6 (2018), 719–732. doi:10.14778/3184470.3184475

[14] Aryak Sen, Silviu Maniu, and Pierre Senellart. 2025. ProvSQL: A General System for Keeping Track of the Provenance and Probability of Data. arXiv:2504.12058 [cs.DB] https://arxiv.org/abs/2504.12058

[15] Pierre Senellart. 2017. Provenance and Probabilities in Relational Databases. *SIGMOD Rec.* 46, 4 (2017), 5–15. doi:10.1145/3186549.3186551

[16] Pierre Senellart. 2024. On the Impact of Provenance Semiring Theory on the Design of a Provenance-Aware Database System. In *The Provenance of Elegance in Computation - Essays Dedicated to Val Tannen, Tannen's Festschrift, May 24-25, 2024, University of Pennsylvania, Philadelphia, PA, USA (OASIcs, Vol. 119)*, Antoine Amarilli and Alin Deutsch (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:10. doi:10.4230/OASICS.TANNEN.9

[17] Pierre Senellart et al. 2025. ProvSQL. https://github.com/PierreSenellart/provsql.

[18] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and Probability Management in PostgreSQL. *Proc. VLDB Endow.* 11, 12 (2018), 2034–2037. doi:10.14778/3229863.3236253