Demonstration of ProvSQL Update Provenance through Temporal Databases

Albert Widiaatmaja, Pierre Senellart

École Normale Supérieure (ENS), National University of Singapore (NUS)

Introduction

ProvSQL extends the PostgreSQL database system by introducing support for (m-)semiring provenance computation for retrieval (`SELECT`) queries, implemented using provenance circuits which are arithmetic circuits where the gates correspond to the operators of (m-)semirings. In this demonstration, we further enhanced ProvSQL by enabling provenance tracking for update operations (`DELETE`, `INSERT`, `UPDATE`) and illustrated its practical utility by implementing a temporal database capable of standard temporal queries including time travel (inspecting past database states), history tracking (monitoring tuple states over time), and undo (reversing previous updates).

Implementation of Update Provenance

Provenance for updates can be supported by annotating tuples with operations that capture the type of modification [1]. In ProvSQL, we implemented these annotations through provenance circuit gates via statement-level triggers based on the following approach:

- `DELETE`: Create a $\ominus$ gate, and move the gate representing the deleted tuple to be its child.
- `INSERT`: Create a $\otimes$ gate, and move the gate representing the inserted tuple to be its child.
- `UPDATE`: Create a $\ominus$ gate, and move the gate representing the old tuple to be its child. Create a $\otimes$ gate, and move the gate representing the new tuple to be its child.

Example

Suppose we execute the following delete operation on table t shown in Table 1. a and b are ids of gates in the provenance circuit which represent their respective tuple.

```
DELETE FROM t;
```

id	provsql
0	a
1	b

Table 1: Table t

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 as shown in Figure 1.

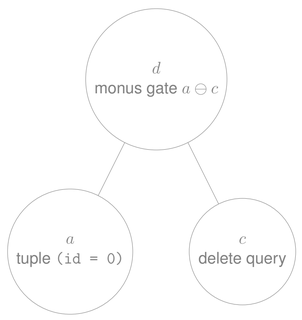

Figure 1: Provenance of tuple `(id = 0)` after deletion

Table t will be updated as shown in Table 2.

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

Table 2: Table t after deletion

Metadata of the executed query including the SQL code of the update operation, the user performing it and its timestamp are stored in a dedicated `update_provenance` metadata table.

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. In ProvSQL, `UNDO` is implemented by replacing any occurrences of c with $c \ominus u$ in the provenance circuit, where c and u represent the query to undo and the `UNDO` query respectively, as illustrated in Figure 2. `UNDO` is also a tracked operation with query metadata inserted into `update_provenance` table, allowing `UNDO` to be undone as well.

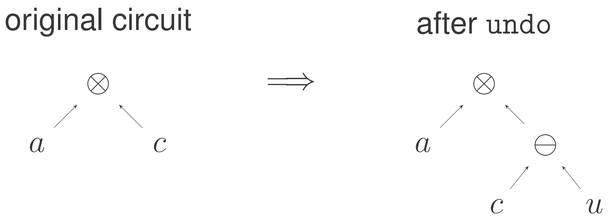

Figure 2: Replacing c with $(c \ominus u)$ in the provenance circuit

Temporal Databases through Provenance

We implemented a temporal database by interpreting the provenance circuit using the union-of-intervals semiring U, the set of finite unions of pairwise-disjoint intervals representing validity periods of database tuples. The union-of-intervals semiring is implemented in ProvSQL using a set of User-Defined Functions (UDFs) which operate on timestamp multirange fields (`tstzmultirange` in PostgreSQL).

Example

Suppose we run the query below.

```
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'
) FROM test;
```

We will obtain the result in Table 3.

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 3: `union_intervals` example

We see that `union_intervals` returns the multirange representing the valid time of each tuple. Using the valid time generated by `union_intervals`, we can support temporal functions such as:

- `get_valid_time`: returns the valid time interval of a given tuple in a table,
- `timetravel`: returns the state of a table at a specified timestamp,
- `timeslice`: returns all tuples that were valid within a given time range, and
- `history`: reveals the sequence of states of a specific tuple over time.

Demonstration Scenario

We have the following tables of France government ministers across time retrieved from Wikidata through its SPARQL endpoint: `person`, `holds`, `party`, and `person_position`.
We can run temporal queries and obtain the results below.

```
-- What were the ministers during Emmanuel Macron'
  s presidential terms?
SELECT name, validity FROM
  timeslice('person_position', '2017-05-16', NOW()
    )
  AS (name TEXT, position TEXT, validity
    tstzmultirange, provsql uuid)
  ORDER BY validity;
```

name	validity
Catherine Colonna	{["2005-06-02 00:00:00+00","2007-05-15 00:00:00+00"), "2022-05-20 00:00:00+00","2024-01-11 00:00:00+00")}
...	

Table 4: Ministers during Emmanuel Macron's presidential terms

```
-- Who were the Ministers of Justice over time?
SELECT name, validity FROM
  history('person_position', ARRAY['position'],
    ARRAY['Minister of Justice'])
  AS (name TEXT, position TEXT, validity
    tstzmultirange, provsql uuid)
  ORDER BY validity;
```

name	validity
Dominique Joseph Garat	{["1792-10-09 00:00:00+00", "1793-03-19 00:00:00+00")}
Charles Joseph Mathieu Lambrechts	{["1797-09-24 00:00:00+00", "1799-07-20 00:00:00+00")}
...	

Table 5: History of Ministers of Justice

We can track updates applied to the database.

```
-- Fire François Bayrou and replace him with
  Pierre Senellart and undo the change
DELETE FROM holds WHERE position='Prime Minister
  of France' AND id IN
  (SELECT id FROM person WHERE name='François
    Bayrou');
INSERT INTO person(id, name, gender) VALUES
  (100000, 'Pierre Senellart', 'male');
INSERT INTO holds(id, position, country) VALUES
  (100000, 'Prime Minister of France', 'FR');

-- What is the current government composition?
SELECT name, position FROM timetravel('
  person_position', NOW())
  AS tt(name TEXT, position TEXT, validity
    tstzmultirange, provsql uuid)
ORDER BY position;
```

name	position
Pierre Senellart	Prime Minister of France
Élisabeth Borne	Minister of National Education
Philippe Baptiste	research minister
Rachida Dati	Minister of Culture (France)
...	

Table 6: Composition of government after update

We can also undo an update operation, even an undo itself.

```
-- Undo the changes: Pierre Senellart is out,
  François Bayrou is in
SELECT undo(provenance()) FROM update_provenance;

-- What were the positions of François Bayrou over
   time, now he has been fired and then reinstated?
SELECT position, union_tstzintervals(provenance(),
  'time_validity_view') valid
FROM person JOIN holds ON person.id=holds.id
WHERE name='François Bayrou' order by valid;
```

position	valid	provenance
Prime Minister of France	{["1951-05-25 00:00:00+00", "2025-06-23 06:32:11.894213+00"], ["2025-06-23 06:32:18.056573+00",)}	d85871bf-...
Minister of National Education, Higher Education and Research	{["1993-03-29 00:00:00+00", "1997-06-04 00:00:00+00"]}	968c45c0-...
Minister of Justice	{["2017-05-17 00:00:00+00", "2017-06-21 00:00:00+00"]}	facb4aab-...
...		

Table 7: Positions of `François Bayrou` over time

The dataset and example ProvSQL queries are available from `https://provsql.org/temporal_demo`.

References

[1] Pierre Bourhis et al. "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*. Ed. by David Maier et al. ACM, 2020, pp. 415–429. DOI: 10.1145/3318464.3380578. URL: https://doi.org/10.1145/3318464.3380578.

Contact: w.albertariel@u.nus.edu or pierre@senellart.com