Implementing “AS OF”-queries in PostgreSQL

HANS-JÜRGEN SCHÖNIG

02.2019 / Category: How To / Tags: query | sql help

Over the years many people have asked for “timetravel” or “AS OF”-queries in PostgreSQL. Oracle has provided this kind of functionality for quite some time already. However, in the PostgreSQL world “AS OF timestamp” is not directly available. The question now is: How can we implement this vital functionality in user land and mimic Oracle functionality?

Table of Contents

IMPLEMENTING “AS OF” AND TIMETRAVEL IN USER LAND

Let us suppose we want to version a simple table consisting of just three columns: id, some_data1 and some_data2. To do this we first have to install the btree_gist module, which adds some valuable operators we will need to manage time travel. The table storing the data will need an additional column to handle the validity of a row. Fortunately, PostgreSQL supports “range types”, which allow to store ranges in an easy and efficient way. Here is how it works:

```
CREATE EXTENSION IF NOT EXISTS btree_gist;

CREATE TABLE t_object
(
    id      int8,
    valid       tstzrange,
    some_data1  text,
    some_data2  text,
    EXCLUDE USING gist (id WITH =, valid WITH &&)
);
```

Attention!

Mind the last line here: “EXLUDE USING gist” will ensure that if the “id” is identical the period (“valid”) must not overlap. The idea is to ensure that the same “id” only has one entry at a time. PostgreSQL will automatically create a Gist index on that column. The feature is called “exclusion constraint”. If you are looking for more information about this feature consider checking out the official documentation (https://www.postgresql.org/docs/current/ddl-constraints.html).

How can I filter?

If you want to filter on some_data1 and some_data2 consider creating indexes. Remember, missing indexes are in many cases the root cause of bad performance:

```
CREATE INDEX idx_some_index1 ON t_object (some_data1);
CREATE INDEX idx_some_index2 ON t_object (some_data2);
```

By creating a view, it should be super easy to extract data from the underlying tables:

```
CREATE VIEW t_object_recent AS
    SELECT  id, some_data1, some_data2
    FROM    t_object
    WHERE   current_timestamp <@ valid;

SELECT * FROM t_object_recent;
```

For the sake of simplicity I have created a view, which returns the most up to date state of the data. However, it should also be possible to select an old version of the data. To make it easy for application developers I decided to introduce a new GUC (= runtime variable), which allows users to set the desired point in time. Here is how it works:

```
SET timerobot.as_of_time = '2018-01-10 00:00:00';
```

Then you can create a second view, which returns the old data:

```
CREATE VIEW t_object_historic AS
    SELECT  id, some_data1, some_data2
    FROM    t_object
    WHERE   current_setting('timerobot.as_of_time')::timestamptz <@ valid;
SELECT * FROM t_object_historic;
```

It is of course also possible to do that with just one view. However, the code is easier to read if two views are used (for the purpose of this blog post). Feel free to adjust the code to your needs.

If you are running an application you usually don't care what is going on behind the scenes - you simply want to modify a table and things should take care of themselves in an easy way. Therefore, it makes sense to add a trigger to your t_object_current table, which takes care of versioning. Here is an example:

```
CREATE FUNCTION version_trigger() RETURNS trigger AS
$
BEGIN
    IF TG_OP = 'UPDATE'
    THEN
        IF NEW.id <> OLD.id
        THEN
            RAISE EXCEPTION 'the ID must not be changed';
        END IF;

        UPDATE  t_object
        SET     valid = tstzrange(lower(valid), current_timestamp)
        WHERE   id = NEW.id
            AND current_timestamp <@ valid;

        IF NOT FOUND THEN
            RETURN NULL;
        END IF;
    END IF;

    IF TG_OP IN ('INSERT', 'UPDATE')
    THEN
        INSERT INTO t_object (id, valid, some_data1, some_data2)
            VALUES (NEW.id,
                tstzrange(current_timestamp, TIMESTAMPTZ 'infinity'),
                NEW.some_data1,
                NEW.some_data2);

        RETURN NEW;
    END IF;

    IF TG_OP = 'DELETE'
    THEN
        UPDATE  t_object
        SET     valid = tstzrange(lower(valid), current_timestamp)
        WHERE id = OLD.id
            AND current_timestamp <@ valid;

        IF FOUND THEN
            RETURN OLD;
        ELSE
            RETURN NULL;
        END IF;
    END IF;
END;
$ LANGUAGE plpgsql;

CREATE TRIGGER object_trig
    INSTEAD OF INSERT OR UPDATE OR DELETE
    ON t_object_recent
    FOR EACH ROW
    EXECUTE PROCEDURE version_trigger();
```

The trigger will take care that INSERT, UPDATE, and DELETE is properly taken care of.

FINALLY: TIMETRAVEL MADE EASY

It is obvious that versioning does have an impact on performance. You should also keep in mind that UPDATE and DELETE are more expensive than previously. However, the advantage is that things are really easy from an application point of view. Implementing time travel can be done quite generically and most applications might not have to be changed at all. What is true, however, is that foreign keys will need some special attention and might be easy to implement in general. It depends on your applications whether this kind of restriction is in general a problem or not.

6 responses to “Implementing “AS OF”-queries in PostgreSQL”

1. **Ravi Krishna** says:
   February 5, 2019 at 3:35 pm

   "Oracle has provided this kind of functionality for quite some time already"
   Also DB2 LUW. DB2's implementation of bi temporal tables is probably the best.

   Reply

2. **Дмитрий Долгов** says:
   February 8, 2019 at 9:35 am

   There was an interesting discussion in the past about implementing similar stuff in PostgreSQL https://www.postgresql.org/message-id/flat/78aadf6b-86d4-21b9-9c2a-51f1efb8a499@postgrespro.ru

   Reply

3. **Thierry Moraty** says:
   September 7, 2019 at 5:20 pm

   First versions of Postgres used to include this feature too

   Reply

4. **Mike** says:
   September 10, 2019 at 3:35 pm

   Hi,
   I tried to implement this solution but trigger doesn't works on run PostgreSQL said: tables cannot have triggers INSTEAD OF
   Other question what is correct solution to implement Validation-Time time travel?

   I need that my users to be able to edit field value from now to infinte or from dataA to dateB and split the record if already existing A o B in record range date
   Ex
   in DB there is a record with
   valid = ['2019-01-01', 'infinity')

name = "pippo";

user do an update with:
name = "pluto"
valid = ['2019-06-01', '2019-08-01')

DB must have this 3 records
valid = ['2019-08-01', 'infinity') name = "pippo";
valid = ['2019-06-01', '2019-08-01') name = "pluto";
valid = ['2019-08-01', '2019-06-01') name = "pippo";

Thanks a lot if someone helps me

Reply

5. **Ronny Timmermans** says:
July 14, 2024 at 4:11 pm

Hans-Jürgen, does this time travel also support joins (ie given time correct results when joining tables on a "past date"?

Reply

- **Laurenz Albe** says:
  July 16, 2024 at 7:58 pm

  Well, sure: you time travel in both tables, then join the result.

  Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment*

Name*

Email*

Website

☐

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Articles by our PostgreSQL Experts

Hans-Jürgen Schönig
CEO, Founder of CYBERTEC & PostgreSQL Visionary
285 Posts >

Laurenz Albe
Senior Consultant & Support Engineer
101 Posts >

Pavlo Golub
PostgreSQL Expert and Developer
45 Posts >

Florian Nadler
GIS Consultant
16 Posts >

Julian Markwort
PostgreSQL Consultant and Developer
6 Posts >

Ants Asma
Senior Developer & Consultant
4 Posts >

Christoph Berg
Senior PostgreSQL Engineer
8 Posts >

Stay tuned with our
NEWSLETTER

Your Email

your@email.com

☐ I accept the terms and conditions and read the privacy policy. We use Google reCAPTCHA.

SUBMIT

Blog Tags

// administration

// development

// sql help

// tuning

// monitoring

// postgis

// conference

// community

Hans-Jürgen Schönig

Founder & CEO of CYBERTEC

Hans-Jürgen Schönig has worked with PostgreSQL since the 90's. He is the CEO and technical lead of CYBERTEC PostgreSQL International, a market leader in the field. He's served countless customers around the globe since the year 2000. He is also the author of the well-received "Mastering PostgreSQL" book series, as well as several other books about PostgreSQL replication and administration.

IMPRINT