DATABASE ADMINISTRATION & MONITORING

Temporal tables for PostgreSQL 15?

26.08.2021 by **Daniel Westermann**

One of the features which is currently missing in PostgreSQL is Temporal Tables. Other database systems have that since a long time and many people want to have in PostgreSQL as well. If you don't know what it is, **here** is a short overview. Basically you can ask for a row as it was at a specific point in time. The **patch** which implements this, is currently in status **"Ready for Committer"**, which does not guarantee that it actually will be committed, but it should be in shape to test the functionality that comes with it.

Having applied the patch and re-compiled PostgreSQL the feature can be tested. The most easy way to enable the feature for a table is this:

```
postgres=# create table t1 ( a int primary key generated alway
                           , b text )
                           with system versioning;
CREATE TABLE
postgres=# d t1
                                        Table "public.t1"
  Column   |           Type           | Collation | Nullable
-----------+--------------------------+-----------+----------
 a         | integer                  |           | not null
 b         | text                     |           |
 starttime | timestamp with time zone |           | not null
```

```
 endtime   | timestamp with time zone |           | not null
Indexes:
    "t1_pkey" PRIMARY KEY, btree (a, endtime)
```

Two additional columns have been added automatically, starttime and endtime. If you don't like the column names you can also do it like this:

```
postgres=# create table t2 ( a int primary key generated alway
                           , b text
                           , mystart timestamptz generated alw
                           , myend timestamptz generated alway
CREATE TABLE
postgres=# d t2
                                    Table "public.t2"
 Column  |           Type           | Collation | Nullable |
---------+--------------------------+-----------+----------+-
 a       | integer                  |           | not null | g
 b       | text                     |           |          |
 mystart | timestamp with time zone |           | not null | g
 myend   | timestamp with time zone |           | not null | g
Indexes:
    "t2_pkey" PRIMARY KEY, btree (a, myend)
```

Note that the two columns need to be defined as **timestamptz**:

```
postgres=# create table t3 ( a int primary key generated alway
ERROR:  data type of row start time must be timestamptz
```

To see the feature in action lets add some data, fire an update and have a look at the history:

```
postgres=# insert into t1 (b) select md5(a::text) from generat
INSERT 0 10
postgres=# update t1 set b = random()::text;
UPDATE 10
postgres=# select * from t1 for system_time from '-infinity' t
 a  |                b                 |           starttime
----+----------------------------------+----------------------
  1 | c4ca4238a0b923820dcc509a6f75849b | 2021-08-24 16:16:25.6
  1 | 0.42111793538855835              | 2021-08-24 16:16:55.4
  2 | c81e728d9d4c2f636f067f89cc14862c | 2021-08-24 16:16:25.6
  2 | 0.5479128803753532               | 2021-08-24 16:16:55.4
  3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 | 2021-08-24 16:16:25.6
  3 | 0.5512468293024142               | 2021-08-24 16:16:55.4
  4 | a87ff679a2f3e71d9181a67b7542122c | 2021-08-24 16:16:25.6
  4 | 0.4112741522472554               | 2021-08-24 16:16:55.4
  5 | e4da3b7fbbce2345d7772b0674a318d5 | 2021-08-24 16:16:25.6
```

```
  5 | 0.46017420469036807               | 2021-08-24 16:16:55.4
  6 | 1679091c5a880faf6fb5e6087eb1b2dc | 2021-08-24 16:16:25.6
  6 | 0.3495216613664702                | 2021-08-24 16:16:55.4
  7 | 8f14e45fceea167a5a36dedd4bea2543 | 2021-08-24 16:16:25.6
  7 | 0.2657576876373895                | 2021-08-24 16:16:55.4
  8 | c9f0f895fb98ab9159f51fd0297e236d | 2021-08-24 16:16:25.6
  8 | 0.9808748465536858                | 2021-08-24 16:16:55.4
  9 | 45c48cce2e2d7fbdea1afc51c7c6ad26 | 2021-08-24 16:16:25.6
  9 | 0.4533070845652887                | 2021-08-24 16:16:55.4
 10 | d3d9446802a44259755d38e6d163e820 | 2021-08-24 16:16:25.6
 10 | 0.20914767879762408               | 2021-08-24 16:16:55.4
(20 rows)
```

The rows with “endtime = infinity” are the current ones, the ones with an actual endtime are historical versions. Doing the almost same thing again makes it more clear:

```
postgres=# update t1 set b = 'xxxxx';
UPDATE 10
postgres=# select * from t1 for system_time from '-infinity' t
 a  |                b                 |           starttime
----+----------------------------------+----------------------
  1 | c4ca4238a0b923820dcc509a6f75849b | 2021-08-24 16:16:25.6
  1 | 0.42111793538855835               | 2021-08-24 16:16:55.4
  1 | xxxxx                             | 2021-08-24 16:58:24.7
  2 | c81e728d9d4c2f636f067f89cc14862c | 2021-08-24 16:16:25.6
  2 | 0.5479128803753532                | 2021-08-24 16:16:55.4
  2 | xxxxx                             | 2021-08-24 16:58:24.7
  3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 | 2021-08-24 16:16:25.6
  3 | 0.5512468293024142                | 2021-08-24 16:16:55.4
  3 | xxxxx                             | 2021-08-24 16:58:24.7
  4 | a87ff679a2f3e71d9181a67b7542122c | 2021-08-24 16:16:25.6
  4 | 0.4112741522472554                | 2021-08-24 16:16:55.4
  4 | xxxxx                             | 2021-08-24 16:58:24.7
  5 | e4da3b7fbbce2345d7772b0674a318d5 | 2021-08-24 16:16:25.6
  5 | 0.46017420469036807               | 2021-08-24 16:16:55.4
  5 | xxxxx                             | 2021-08-24 16:58:24.7
  6 | 1679091c5a880faf6fb5e6087eb1b2dc | 2021-08-24 16:16:25.6
  6 | 0.3495216613664702                | 2021-08-24 16:16:55.4
  6 | xxxxx                             | 2021-08-24 16:58:24.7
  7 | 8f14e45fceea167a5a36dedd4bea2543 | 2021-08-24 16:16:25.6
  7 | 0.2657576876373895                | 2021-08-24 16:16:55.4
  7 | xxxxx                             | 2021-08-24 16:58:24.7
  8 | c9f0f895fb98ab9159f51fd0297e236d | 2021-08-24 16:16:25.6
  8 | 0.9808748465536858                | 2021-08-24 16:16:55.4
  8 | xxxxx                             | 2021-08-24 16:58:24.7
  9 | 45c48cce2e2d7fbdea1afc51c7c6ad26 | 2021-08-24 16:16:25.6
  9 | 0.4533070845652887                | 2021-08-24 16:16:55.4
  9 | xxxxx                             | 2021-08-24 16:58:24.7
```

```
 10 | d3d9446802a44259755d38e6d163e820 | 2021-08-24 16:16:25.6
 10 | 0.20914767879762408              | 2021-08-24 16:16:55.4
 10 | xxxxx                            | 2021-08-24 16:58:24.7
(30 rows)
```

Without specifying any time frame you get the current version of rows, of course:

```
postgres=# select * from t1 order by a;
 a  |   b   |           starttime           | endtime
----+-------+-------------------------------+----------
  1 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  2 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  3 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  4 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  5 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  6 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  7 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  8 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
  9 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
 10 | xxxxx | 2021-08-24 16:58:24.799322+02 | infinity
```

Asking for a specific point in time works as well:

```
postgres=# select * from t1 for system_time as of '2021-08-24
 a  |                b                 |           starttime
----+----------------------------------+----------------------
  1 | c4ca4238a0b923820dcc509a6f75849b | 2021-08-24 16:16:25.6
  2 | c81e728d9d4c2f636f067f89cc14862c | 2021-08-24 16:16:25.6
  3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 | 2021-08-24 16:16:25.6
  4 | a87ff679a2f3e71d9181a67b7542122c | 2021-08-24 16:16:25.6
  5 | e4da3b7fbbce2345d7772b0674a318d5 | 2021-08-24 16:16:25.6
  6 | 1679091c5a880faf6fb5e6087eb1b2dc | 2021-08-24 16:16:25.6
  7 | 8f14e45fceea167a5a36dedd4bea2543 | 2021-08-24 16:16:25.6
  8 | c9f0f895fb98ab9159f51fd0297e236d | 2021-08-24 16:16:25.6
  9 | 45c48cce2e2d7fbdea1afc51c7c6ad26 | 2021-08-24 16:16:25.6
 10 | d3d9446802a44259755d38e6d163e820 | 2021-08-24 16:16:25.6
```

... or asking like this:

```
postgres=# select * from t1 for system_time between '2021-08-2
 a  |          b          |           starttime           |
----+---------------------+-------------------------------+---
  1 | 0.42111793538855835 | 2021-08-24 16:16:55.417076+02 | 20
  1 | xxxxx               | 2021-08-24 16:58:24.799322+02 | in
  2 | 0.5479128803753532  | 2021-08-24 16:16:55.417076+02 | 20
  2 | xxxxx               | 2021-08-24 16:58:24.799322+02 | in
  3 | 0.5512468293024142  | 2021-08-24 16:16:55.417076+02 | 20
  3 | xxxxx               | 2021-08-24 16:58:24.799322+02 | in
  4 | 0.4112741522472554  | 2021-08-24 16:16:55.417076+02 | 20
```

```
 4 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
 5 | 0.46017420469036807 | 2021-08-24 16:16:55.417076+02 | 20
 5 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
 6 | 0.3495216613664702  | 2021-08-24 16:16:55.417076+02 | 20
 6 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
 7 | 0.2657576876373895  | 2021-08-24 16:16:55.417076+02 | 20
 7 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
 8 | 0.9808748465536858  | 2021-08-24 16:16:55.417076+02 | 20
 8 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
 9 | 0.4533070845652887  | 2021-08-24 16:16:55.417076+02 | 20
 9 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
10 | 0.20914767879762408 | 2021-08-24 16:16:55.417076+02 | 20
10 | xxxxx               | 2021-08-24 16:58:24.799322+02 | ir
```

Really looks promising. Thanks to all involved.

Post Views: 2,718

by

Daniel Westermann

No comments yet.

Leave a Reply:

Email

name@example.com

Username

Comment

BE SHARING

Related blog articles

DATABASE ADMINISTRATION & MONITORING

Uyuni, an open-source configuration and infrastructure management solution for software-defined infrastructure (2) – Adding a client

29.12.2022 by **Daniel Westermann**

DATABASE ADMINISTRATION & MONITORING

Uyuni, an open-source configuration and infrastructure management solution for software-defined infrastructure (1) – The server

28.12.2022 by **Daniel Westermann**

DATABASE ADMINISTRATION & MONITORING

Fast setup of a two node repmgr cluster with auto failover

23.12.2022 by **Karsten Lenz**

DATABASE ADMINISTRATION & MONITORING

Applying RU on Enterprise Manager 13c R5

16.12.2022 by **Mouhamadou Diaw**

ABOUT

dbi services is a company specialized in IT consulting and services. We are experts in innovative and efficient data infrastructures and platforms. Tailor-made solutions is what we offer to our customers thanks to our consultants, whose skills and knowledge are constantly evolving thanks to continuous training.

Delémont >

Nyon >

Bâle >

Berne >

Zurich >

WE ARE CLOSE TO YOU

Delémont >

Nyon >

Bâle >

Berne >

Zurich