

Time Travelling PostgreSQL

PostgreSQL Conference Europe 2026
Stefanie Janine Stölting





About Me



Owner of Tallinn/Estonia based ProOpenSource OÜ

Website: www.ProOpenSource.eu

Blog: www.ProOpenSource.it

Mastodon: <https://digitalcourage.social/@sjstoelting>

Telegram: <https://t.me/postgreschat>



About Me

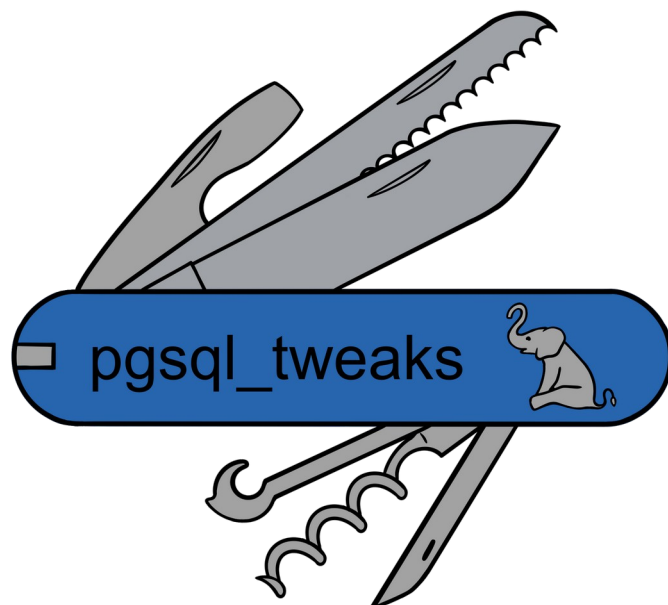


Maintainer of `pgsql_tweaks`, a PostgreSQL extension

Sources: https://codeberg.org/pgsql_tweaks

Website: <https://pgsql-tweaks.org/>

Docu: <https://rtfm.pgsql-tweaks.org/>





PostgreSQL Event Calendars



<https://proopensource.it/conference-calendar/>
(Based on ProOpenSource Nextcloud)



<https://ics.postgresql.life/>
(Based on Google Calendar)





About Me



Member of the **PGEU Diversity Committee**



This year the conference survey will have some diversity related questions. Please support the committee by filling it out.
Thank you



What is Time Travelling





Timetravel Definition



Being able to return data as it has been at a certain point in time.

That can obviously only be data saved in the past.

And we need dates and times stored with data.



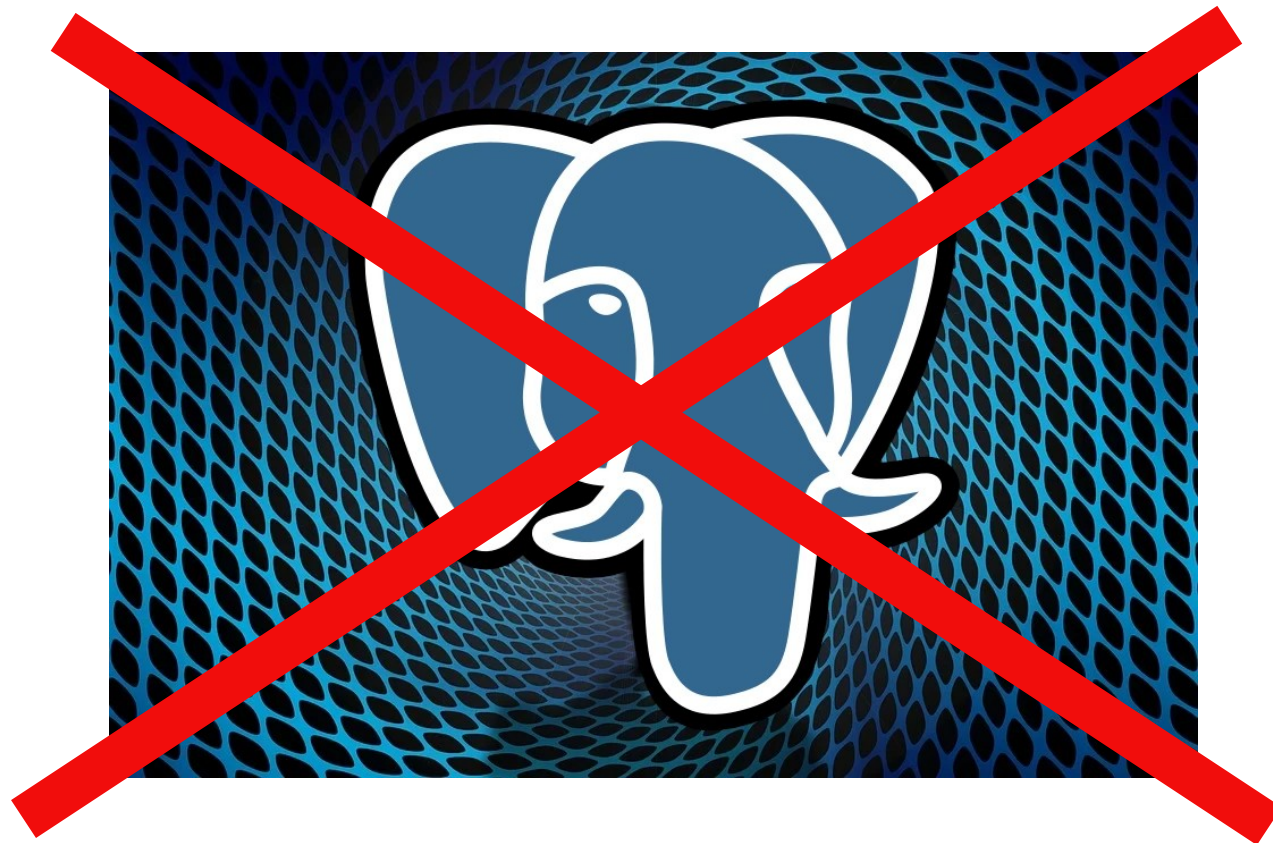
Photo by [Tim Green](#)



PostgreSQL History



PostgreSQL supported time travel as it was implemented in its core up to version 6.2. It has been removed in [PostgreSQL 6.3](#).





Other Ideas



Hans-Jürgen Schönig from CYBERTEC has written about implementing time travelling in his blog post in 2019 about [AS OF-queries](#).

I have tested them, but they seem not to work with PostgreSQL 17, 18, 19.

The creation of the trigger fails.



Other Ideas



The problem here is:

```
CREATE FUNCTION version_trigger() RETURNS trigger AS
$
...
...
END;
$ LANGUAGE plpgsql;
```

The solution with PostgreSQL >= 17 is:

```
CREATE FUNCTION version_trigger()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
AS
$$
...
...
END;
$$
```



Other Ideas - E-Maj



E-Maj is a PostgreSQL extension which is available by source code and with **PGXN**.

It enables you to set marks for a group of tables and to roll back to a mark.

This is great for tests against databases and spares you to install backups.



What is covered?



Table partitioning by list

Primary key with partitioned tables

Updating records with the possibility to access the previous records as they have been at a certain point in time

Deleting records with the possibility to access the previous records as they have been at a certain point in time



Let's start



Photo by [Wim van 't Einde](#) on [Unsplash](#)



Primary Key - Partitioning



PostgreSQL does not support unique indexes or primary keys over partitioned tables.

But there is a solution to solve the primary key problem without any extensions in PostgreSQL.





Unique Identifiers



The solution is relatively easy to solve with Unique Identifiers.

Since PostgreSQL 14 there is the version 7 UUID available without the need of an extension: uuidv7.



Preparation



I stated we do not need an extension.

Well, we need one to get speedy access to the data: `btree_gist`.

`btree_gist` is an index that is part of the contrib package and available on all supported platforms.

```
CREATE EXTENSION IF NOT EXISTS btree_gist;
```



Time Travel Table



```
/**
 * Creates the partitioned table which will store all versions of records
 *
 * TSTZRANGE is a TIMESTAMP WITH TIME ZONE range Type
 * https://www.postgresql.org/docs/current/rangetypes.html
 */
CREATE TABLE timetravel (
    timetravelid UUID NOT NULL DEFAULT uuidv7(),
    changed TSTZRANGE NOT NULL DEFAULT
        tstzrange(clock_timestamp(), 'INFINITY', '[]'),
    data_text TEXT,
    data_json JSONB,
    p_key INTEGER NOT NULL DEFAULT ((to_char(now(), 'YYYYMM'))::INTEGER),
    deleted BOOLEAN NOT NULL DEFAULT FALSE,
    CONSTRAINT timetravel_pk PRIMARY KEY (timetravelid, changed, p_key)
) PARTITION BY LIST (p_key)
;
```



Indexes



```
CREATE INDEX timetravelid_idx
ON timetravel
USING btree
(timetravelid, p_key)
;
```

```
CREATE INDEX timetravel_changed_idx
ON timetravel
USING gist
(changed)
;
```

```
CREATE INDEX timetravel_not_deleted_idx
ON timetravel
USING btree
(deleted)
WHERE NOT deleted
;
```



Partition Management



A function will be used to create partitions with unique names.

The function will be executed manually.

In real life I would use a cron job, for example with `pg_cron`.



Show Time



Showing slides with SQL is a bit boring.





Schema, Searchpath



-- Drops the Schema used in this script with all objects

DROP SCHEMA IF EXISTS timetravel CASCADE;

CREATE SCHEMA IF NOT EXISTS timetravel;

SET SEARCH_PATH TO timetravel, public;



Index Type



```
/**  
 * Creates the btree GIST index which will be used for the column changed  
 * https://www.postgresql.org/docs/current/btree-gist.html  
 */  
CREATE EXTENSION IF NOT EXISTS btree_gist;
```



Timetravel Table



```
/**
 * Creates the partitioned table which will store all versions of records
 *
 * TSTZRANGE is a TIMESTAMP WITH TIME ZONE range Type
 * https://www.postgresql.org/docs/current/rangetypes.html
 */
CREATE TABLE timetravel (
    timetravelid UUID NOT NULL DEFAULT uuidv7(),
    changed TSTZRANGE NOT NULL DEFAULT
        tstzrange(clock_timestamp(), 'INFINITY', '[]'),
    data_text TEXT,
    data_json JSONB,
    p_key INTEGER NOT NULL DEFAULT ((to_char(now(), 'YYYYMM'))::INTEGER),
    deleted BOOLEAN NOT NULL DEFAULT FALSE,
    CONSTRAINT timetravel_pk PRIMARY KEY (timetravelid, changed, p_key)
) PARTITION BY LIST (p_key)
;
```



Table Timetravel Indexes



```
-- Creating indexes
CREATE INDEX timetravelid_idx
  ON timetravel
  USING btree
  (timetravelid, p_key)
;

CREATE INDEX timetravel_changed_idx
  ON timetravel
  USING gist
  (changed)
;

CREATE INDEX timetravel_not_deleted_idx
  ON timetravel
  USING btree
  (deleted)
  WHERE NOT deleted
;
```



Functions



```
/**FUNCTIONS*****/

/**
 * Function to handle partitions
 *
 * It uses system tables to check for existing partitions
 * Partitions are done for months
 * Returns the name of the created partition
 */
```



FUNCTION timetravel_partition



```
CREATE OR REPLACE FUNCTION timetravel_partition (  
    in_changed TIMESTAMPTZ,  
    in_schema TEXT,  
    in_table TEXT  
)  
    RETURNS TEXT  
    LANGUAGE PLPGSQL  
    AS  
$$  
DECLARE  
    in_range BOOLEAN = FALSE;  
    p_key INTEGER;  
    part_name TEXT;
```



FUNCTION timetravel_partition



BEGIN

```
-- Calculate the partition time
SELECT (to_char($1, 'YYYYMM'))::INTEGER
INTO p_key
;

-- Partition name
part_name := 'timetravel_' || p_key::TEXT;

-- Check the changed date to be in an existing partition
EXECUTE 'SELECT count(*) > 0
FROM pg_inherits
INNER JOIN pg_class AS parent
    ON pg_inherits.inhparent = parent.oid
INNER JOIN pg_class AS child
    ON pg_inherits.inhrelid = child.oid
INNER JOIN pg_namespace AS nmsp_parent
    ON nmsp_parent.oid = parent.relnamespace
WHERE parent.relname = $1
    AND nmsp_parent.nspname = $2
    AND child.relname = $3'
INTO in_range
USING in_table, in_schema, part_name
;
```



FUNCTION timetravel_partition



```
IF (NOT in_range) THEN
    -- Create a new partition
    EXECUTE 'CREATE TABLE ' || part_name ||
    ' PARTITION OF timetravel FOR VALUES IN ('' ' || p_key::text || '' ');

    RAISE NOTICE 'Partition % created.', part_name;
END IF;

RETURN part_name;
END;
$$
;
```



Creating the Partitions



-- Create the partitions for this some months

```
SELECT timetravel_partition (  
    now(),  
    'timetravel',  
    'timetravel'  
);
```

```
SELECT timetravel_partition (  
    now() - INTERVAL '1 Month',  
    'timetravel',  
    'timetravel'  
);
```

```
SELECT timetravel_partition (  
    now() - INTERVAL '2 Months',  
    'timetravel', 'timetravel'  
);
```



Autovacuum Off



```
/**
 * Turning autovacuum off for some tests, has to be done for each partition
 *
 *** DON'T DO THAT OTHERWISE, IT WILL HARM YOU! ***
 */

DO $$
DECLARE
    table_name TEXT;
    i SMALLINT;
    months TEXT;
BEGIN
    FOR i IN 0..2 LOOP
        months := i::TEXT || ' Month';
        EXECUTE 'SELECT timetravel_partition (now() - INTERVAL ''' || months ||
        ''' ,
                ''timetravel'', ''timetravel'')'
        INTO table_name;

        EXECUTE 'ALTER TABLE timetravel.' || table_name || ' SET
        (autovacuum_enabled = false)';
    END LOOP;
END;
$;
```



Trigger Function INSERT/UPDATE



```
-- Trigger function for inserts and updates
CREATE OR REPLACE FUNCTION trigger_timetravel_in_upd ()
    RETURNS TRIGGER
    LANGUAGE PLPGSQL
    AS
$$
BEGIN
    -- Setting default values
    NEW.deleted = false;
    NEW.changed = tstzrange(clock_timestamp(), 'INFINITY', '[]');
```



Trigger Function INSERT/UPDATE



```
CASE WHEN TG_OP = 'UPDATE' THEN

  IF NEW.timetravelid <> OLD.timetravelid THEN
    RAISE EXCEPTION 'The identity column timetravelid can not be changed!';
  END IF;

  IF NEW.p_key <> OLD.p_key THEN
    RAISE EXCEPTION 'The partition key can not be changed!';
  END IF;

  IF NOT OLD.deleted THEN
    IF upper(OLD.changed) = 'infinity' THEN
      -- Updated the latest version of a record
      INSERT INTO timetravel (timetravelid, data_text, data_json, changed)
      SELECT NEW.timetravelid
            , NEW.data_text
            , NEW.data_json
            , NEW.changed
      ;

      -- Only the range for the old record is changed, it has an end now
      NEW.data_text = OLD.data_text;
      NEW.data_json = OLD.data_json;
      NEW.changed = tstzrange(lower(OLD.changed), lower(NEW.changed));

      RETURN NEW;
    ELSE
      -- It is not the newest version, therefore there is nothing to do
      RETURN NULL;
    END IF;
  ELSE
    -- An already deleted record cannot be changed
    RETURN NULL;
  END IF;
```



Trigger Function INSERT/UPDATE



```
CASE WHEN TG_OP = 'UPDATE' THEN

    IF NEW.timetravelid <> OLD.timetravelid THEN
        RAISE EXCEPTION 'The identity column timetravelid can not be changed!';
    END IF;

    IF NEW.p_key <> OLD.p_key THEN
        RAISE EXCEPTION 'The partition key can not be changed!';
    END IF;

    IF NOT OLD.deleted THEN
        IF upper(OLD.changed) = 'infinity' THEN
            -- Updated the latest version of a record
            INSERT INTO timetravel (timetravelid, data_text, data_json, changed)
            SELECT NEW.timetravelid
                   , NEW.data_text
                   , NEW.data_json
                   , NEW.changed
            ;

            -- Only the range for the old record is changed, it has an end now
            NEW.data_text = OLD.data_text;
            NEW.data_json = OLD.data_json;
            NEW.changed = tstzrange(lower(OLD.changed), lower(NEW.changed));

            RETURN NEW;
        ELSE
            -- It is not the newest version, therefore there is nothing to do
            RETURN NULL;
        END IF;
    ELSE
        -- An already deleted record cannot be changed
        RETURN NULL;
    END IF;
WHEN TG_OP = 'INSERT' THEN
    RETURN NEW;
ELSE
    RETURN NULL;
END CASE;

END;
$$;
```



Trigger Function INSERT/UPDATE



```
-- Attach the trigger function for inserts
CREATE OR REPLACE TRIGGER timetravel_insert
  BEFORE INSERT
  ON timetravel
  FOR EACH ROW
  WHEN (pg_trigger_depth() < 1)
  EXECUTE PROCEDURE trigger_timetravel_in_upd()
;

-- Attach the trigger function for updates
CREATE OR REPLACE TRIGGER timetravel_update
  BEFORE UPDATE
  ON timetravel
  FOR EACH ROW
  EXECUTE PROCEDURE trigger_timetravel_in_upd()
;
```



Trigger Function DELETE



```
/**
 * The trigger inserts a records with the old data but with an end
 * timestamp in column changed and with deleted = TRUE but end timestamp
 * is INFINITY
 *
 * https://www.postgresql.org/docs/current/datatype-datetime.html#DATATYPE-DATETIME-SPECIAL-VALUES
 */
CREATE OR REPLACE FUNCTION trigger_timetravel_del ()
    RETURNS TRIGGER
    LANGUAGE PLPGSQL
    AS
$$
DECLARE
    ts timestamp with time zone;
BEGIN
    -- When a record has already been deleted, an error is raised and no data is changed
    IF OLD.deleted THEN
        RAISE EXCEPTION 'Timetravel record with the timetravelid % is already deleted.', OLD.timetravelid;
    END IF;

    IF upper(OLD.changed) = 'infinity' THEN
        -- Only the latest version has to be taken care off
        ts = clock_timestamp();

        INSERT INTO timetravel (timetravelid, changed, data_text, data_json, deleted)
        VALUES (
            OLD.timetravelid,
            tstzrange(ts, 'INFINITY'),
            OLD.data_text,
            OLD.data_json,
            TRUE
        );

        RETURN OLD;
    ELSE
        -- All other records stay as they are
        RETURN NULL;
    END IF;
END;
$*
```



VIEWS



```
/**
 * The view does return only the current and undeleted records
 *
 * It contains the column rec_version, that has the information
 * of the iterations of the records
 */
CREATE OR REPLACE VIEW timetravel_v AS
WITH rec_v AS
(
    SELECT t.timetravelid
    , count(*) OVER (PARTITION BY t.timetravelid) AS rec_version
    FROM timetravel AS t
)
SELECT DISTINCT ON (t.timetravelid)
    t.timetravelid
    , rec_v.rec_version
    , t.data_text
    , t.data_json
    , min(lower(t.changed)) OVER (PARTITION BY t.timetravelid) AS created_at
    , t.changed
    , lower(t.changed) AS valid_from
    , upper(t.changed) AS valid_until
FROM timetravel AS t
INNER JOIN rec_v
    ON t.timetravelid = rec_v.timetravelid
WHERE NOT deleted
    AND upper(t.changed) = 'infinity'::TIMESTAMPTZ
;
```



PostgreSQL

-- Querying view

SELECT *

FROM timetravel_v

ORDER BY timetravelid

;

VIEWS



Grid Text	timetravelid	123 rec_version	Az data_text	{ } data_json	⌚ created_at	☰ changed	⌚ valid_from	⌚ valid_until



PostgreSQL

VIEWS



```
-- Querying table
```

```
SELECT *  
    , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(changed)) AS  
  rec_version  
FROM timetravel AS t  
ORDER BY timetravelid  
    , rec_version  
;
```

	timetravelid	changed	data_text	data_json	p_key	deleted	rec_version



VIEWS



```
/**
 * Statistics information will be used several times,
 * therefore a view makes more sense
 */
CREATE VIEW timetravel_stats AS
WITH rec_v AS
(
    SELECT t.timetravelid
           , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY
lower(t.changed)) AS rec_version
    FROM timetravel AS t
)
SELECT count(t.timetravelid) AS recordcount
       , count(DISTINCT t.timetravelid) AS unique_records
       , count(DISTINCT t.timetravelid) - count(DISTINCT t.timetravelid) FILTER
(WHERE t.deleted) AS active_records
       , count(DISTINCT t.timetravelid) FILTER (WHERE t.deleted) AS deleted_records
       , min(rec_v.rec_version) AS min_version_number
       , max(rec_v.rec_version) AS max_version_number
FROM timetravel AS t
INNER JOIN rec_v
    ON t.timetravelid = rec_v.timetravelid
;
```



PostgreSQL

-- Query the statistic view

SELECT *

FROM timetravel_stats

;

VIEWS



	123 recordcount ▼	123 unique_records ▼	123 active_records ▼	123 deleted_records ▼	123 min_version_number ▼	123 max_version_number ▼
1	0	0	0	0	[NULL]	[NULL]



Insert Data



```
/**TEST*DATA*****
```

```
-- Insert records into several partitions
```

```
ALTER TABLE timetravel ALTER COLUMN p_key  
SET DEFAULT to_char(now() - INTERVAL '2 Months', 'YYYYMM'::TEXT)::INTEGER;
```

```
INSERT INTO timetravel (data_text, data_json)  
SELECT substr(md5(random()::text), 1, 25) AS data_text  
      , to_jsonb(substr(md5(random()::text), 1, 25)) AS data_json  
FROM generate_series(1, 100000) s(i)  
;
```

```
ALTER TABLE timetravel ALTER COLUMN p_key  
SET DEFAULT to_char(now() - INTERVAL '1 Month', 'YYYYMM'::TEXT)::INTEGER;
```

```
INSERT INTO timetravel (data_text, data_json)  
SELECT substr(md5(random()::text), 1, 25) AS data_text  
      , to_jsonb(substr(md5(random()::text), 1, 25)) AS data_json  
FROM generate_series(1, 100000) s(i)  
;
```

```
ALTER TABLE timetravel ALTER COLUMN p_key  
SET DEFAULT to_char(now(), 'YYYYMM'::TEXT)::INTEGER;
```

```
INSERT INTO timetravel (data_text, data_json)  
SELECT substr(md5(random()::text), 1, 25) AS data_text  
      , to_jsonb(substr(md5(random()::text), 1, 25)) AS data_json  
FROM generate_series(1, 100000) s(i)  
;
```



Insert Data



-- Querying table

```
SELECT *  
    , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(changed)) AS  
    rec_version  
FROM timetravel AS t  
ORDER BY timetravelid  
    , rec_version  
;
```

	timetravelid	changed	data_text	data_json	p_key	deleted	rec_version
1	01a01464-040e-78f0-8dd6-c0c75ad4f307	["2026-08-18 12:21:34.095692+02",infinity)	9934db0f66dcae4736b503847	"e82f7ee1e93768378adf67f1f"	202,606	[]	1
2	01a01464-0410-725b-b956-6198700bbf0a	["2026-08-18 12:21:34.09617+02",infinity)	deefa34cf60c35aad1ca0032a	"c608d62fe5750f7d31b553d57"	202,606	[]	1
3	01a01464-0410-7334-aba8-592a3f7312c5	["2026-08-18 12:21:34.096212+02",infinity)	e4f811127d07dfd645aa43ae3	"aefd213a146ae157728652e16"	202,606	[]	1
4	01a01464-0410-73b5-9024-32f451f36c53	["2026-08-18 12:21:34.096241+02",infinity)	30dcde446697ca3b54d681eb8	"89fcc2067490f985e903c3089"	202,606	[]	1



Update Data



-- Update existing records

WITH t1 AS

```
(  
    SELECT row_number() OVER () AS id  
        , timetravelid  
    FROM timetravel  
)
```

, t2 AS

```
(  
    SELECT floor(random() * (300000-1+1) + 1)::bigint AS id  
        , substr(md5(random()::text), 1, 25) AS data_text  
        , to_jsonb(substr(md5(random()::text), 1, 25)) AS data_json  
    FROM generate_series(1, 300000) s(i)  
)
```

, t3 AS

```
(  
    SELECT DISTINCT ON (t1.timetravelid)  
        t1.timetravelid  
        , t2.data_text  
        , data_json  
    FROM t1  
    INNER JOIN t2  
        ON t1.id = t2.id  
)
```

UPDATE timetravel **SET**

```
    data_text = t3.data_text,  
    data_json = t3.data_json
```

FROM t3

WHERE timetravel.timetravelid = t3.timetravelid

;



Update Data



Statistics 1 ×	
Name	Value
Updated Rows	164364
Execute time	21s



Updated Data



-- Querying table

```
SELECT *  
    , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(changed)) AS  
    rec_version  
FROM timetravel AS t  
ORDER BY timetravelid  
    , rec_version  
;
```

	timetravelid	changed	data_text	data_json	p_key	deletec	rec_version
1	01a01464-040e-78f0-8dd6-c0c75ad4f307	["2026-08-18 12:21:34.095692+02","2026-08-18 12:25:20.173526+02")	9934db0f66dcae	"e82f7ee1e93768"	202,606	[]	1
2	01a01464-040e-78f0-8dd6-c0c75ad4f307	["2026-08-18 12:25:20.173526+02",infinity)	1c5451ebb5b2fc	"02fe9f6a20ea6c"	202,608	[]	2
3	01a01464-0410-725b-b956-6198700bbf0a	["2026-08-18 12:21:34.09617+02",infinity)	deefa34cf60c35	"c608d62fe5750f"	202,606	[]	1
4	01a01464-0410-7334-aba8-592a3f7312c5	["2026-08-18 12:21:34.096212+02","2026-08-18 12:25:20.174281+02")	e4f811127d07df	"aefd213a146ae1"	202,606	[]	1



PostgreSQL

Delete Data



```
-- Delete some records
```

```
WITH t1 AS
(
    SELECT row_number() OVER () AS id
           , timetravelid
    FROM timetravel
    LIMIT 6000
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (99, 654, 5698)
)
DELETE FROM timetravel
USING t2
WHERE timetravel.timetravelid = t2.timetravelid
;
```

Statistics 1 ×	
Name	Value
Updated Rows	3
Execute time	0.202s



PostgreSQL

-- Table statistics

```
SELECT *  
FROM timetravel_stats  
;
```



Data Statistics

	123 recordcount ▾	123 unique_records ▾	123 active_records ▾	123 deleted_records ▾	123 min_version_number ▾	123 max_version_number ▾
1	2,472,317	300,000	299,997	3	1	4



Current Data from View



-- Get current data

```
SELECT *  
FROM timetravel_v AS tv  
ORDER BY tv.timetravelid  
;
```

	timetravelid	rec_version	data_text	data_json	created_at	changed	valid_from	valid_until
1	01a01464-040e-78f0-8dd6-c0c75ad4f307	4	d2d15e97e20d6	"b83beccbcea3b0c	2026-08-18 12:31:41.316 +0200	["2026-08-18 12:31:41.316555+02",infinity)	2026-08-18 12:31:41.316 +0200	infinity
2	01a01464-0410-725b-b956-6198700bbf0a	3	83b111b7ca607	"c802c4d999220fd	2026-08-18 12:31:41.316 +0200	["2026-08-18 12:31:41.316797+02",infinity)	2026-08-18 12:31:41.316 +0200	infinity
3	01a01464-0410-7334-aba8-592a3f7312c5	4	01593193b5e1b	"9e6f44844af1cf4a	2026-08-18 12:31:41.316 +0200	["2026-08-18 12:31:41.316954+02",infinity)	2026-08-18 12:31:41.316 +0200	infinity
4	01a01464-0410-73b5-9024-32f451f36c53	2	06e10e1ae575c	"286dda482e61c9	2026-08-18 12:30:01.286 +0200	["2026-08-18 12:30:01.286789+02",infinity)	2026-08-18 12:30:01.286 +0200	infinity



Deleted Data



-- Get all records of a deleted one

WITH t1 AS

```
(  
    SELECT timetravelid  
    FROM timetravel  
    WHERE deleted  
)
```

SELECT count(*) **FILTER** (WHERE deleted) **OVER** () **AS** deleted_records

```
, t.timetravelid  
, ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY t.timetravelid,  
lower(t.changed)) AS rec_version  
, t.data_text  
, t.data_json  
, t.deleted  
, t.changed  
, lower(t.changed) AS rec_changed
```

FROM timetravel **AS** t

INNER JOIN t1

ON t.timetravelid = t1.timetravelid

ORDER BY t.timetravelid

```
, lower(t.changed)
```

;

 123 deleted_records	 timetravelid	123 rec_version	Az data_text	{ } data_json	 deleted	 changed	 rec_changed
1	3 01a01464-0417-7c8d-ba4b-cfc59016664d	1	b63133ebae9c8 "3f1643e28f08f	[v]	[v]	["2026-08-18 12:28:27.704995+02",infinity)	2026-08-18 12:28:27.704 +0200
2	3 01a01464-044a-7366-b292-19a562fb92f4	1	5f61830d6eb3a "d44aabdb0c0a	[v]	[v]	["2026-08-18 12:28:27.705367+02",infinity)	2026-08-18 12:28:27.705 +0200
3	3 01a01464-060d-708c-b695-c60475cfbb08	1	f3097f11ab96f4 "d094656b622a	[v]	[v]	["2026-08-18 12:28:27.706524+02",infinity)	2026-08-18 12:28:27.706 +0200



Data for a Point in Time



```
/**
 * Result for a certain point in time
 *
 * Using CONTAINS which is <@,
 * see https://www.postgresql.org/docs/current/functions-range.html
 */
SELECT t.timetravelid
      , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(t.changed))
AS rec_version
      , t.data_text
      , t.data_json
      , min(lower(t.changed)) OVER (PARTITION BY t.timetravelid)
      , t.changed
      , lower(t.changed) AS valid_from
      , upper(t.changed) AS valid_until
      , t.deleted
FROM timetravel AS t
WHERE '2026-08-18 12:28:27.704 +0200'::timestamptz <@ t.changed
ORDER BY t.timetravelid
      , lower(t.changed)
;
```

	timetravelid	rec_version	data_text	data_json	min	changed	valid_from
1	01a01464-040e-78f0-8dd6-c0c75ad4f307	1	1c5451ebb5b2fdc	"02fe9f6a20ea6c08	2026-08-18 12:25:20.173 +0200	["2026-08-18 12:25:20.173526+02","2026-08-18 12:30:01.286149+02")	2026-08-18 12:25:20.1
2	01a01464-0410-725b-b956-6198700bbf0a	1	deefa34cf60c35aa	"c608d62fe5750f7d	2026-08-18 12:21:34.096 +0200	["2026-08-18 12:21:34.09617+02","2026-08-18 12:30:01.28639+02")	2026-08-18 12:21:34.0
3	01a01464-0410-7334-aba8-592a3f7312c5	1	ee0a616fbad6b85	"879fdc6f42df2b33	2026-08-18 12:25:20.174 +0200	["2026-08-18 12:25:20.174281+02","2026-08-18 12:30:01.286621+02")	2026-08-18 12:25:20.1
4	01a01464-0410-73b5-9024-32f451f36c53	1	30dcde446697ca3	"89fcc2067490f985	2026-08-18 12:21:34.096 +0200	["2026-08-18 12:21:34.096241+02","2026-08-18 12:30:01.286789+02")	2026-08-18 12:21:34.0



Trigger for View



```
-- One trigger to bind them all ;-)  
CREATE OR REPLACE FUNCTION trigger_timetravel_view_in_upd_del ()  
  RETURNS TRIGGER  
  LANGUAGE PLPGSQL  
  AS  
  $$  
  BEGIN  
    CASE  
      WHEN TG_OP = 'INSERT' THEN  
        EXECUTE 'INSERT INTO timetravel (data_text, data_json) VALUES ($1, $2)'  
        USING NEW.data_text, NEW.data_json  
        ;  
      WHEN TG_OP = 'UPDATE' THEN  
        EXECUTE 'UPDATE timetravel SET data_text = $1, data_json = $2 WHERE  
timetravelid = $3'  
        USING NEW.data_text, NEW.data_json, OLD.timetravelid  
        ;  
      WHEN TG_OP = 'DELETE' THEN  
        EXECUTE 'DELETE FROM timetravel WHERE timetravelid = $1'  
        USING OLD.timetravelid  
        ;  
    ELSE  
      RAISE EXCEPTION 'Operation not supported.';  
    END CASE;  
  
    RETURN NEW;  
  
  END;  
  $$  
  ;
```



Trigger for View



-- Attach the trigger function to the view

CREATE OR REPLACE TRIGGER timetravel_v_trigger

INSTEAD OF INSERT OR UPDATE OR DELETE

ON timetravel_v

FOR EACH ROW

EXECUTE PROCEDURE trigger_timetravel_view_in_upd_del ()

;



Query the View



-- Query the view

```
SELECT *  
FROM timetravel_v  
;
```

	timetravelid	rec_version	data_text	data_json	created_at	changed	valid_from	valid_until
1	01a01464-040e-78f0-8dd6-c0c75ad4f307	4	d2d15e97e20d6	"b83becbcea3b"	2026-08-18 12:31:41.316 +0200	["2026-08-18 12:31:41.316555+02",infinity)	2026-08-18 12:31:41.316 +0200	infinity
2	01a01464-0410-725b-b956-6198700bbf0a	3	83b111b7ca607	"c802c4d999220"	2026-08-18 12:31:41.316 +0200	["2026-08-18 12:31:41.316797+02",infinity)	2026-08-18 12:31:41.316 +0200	infinity
3	01a01464-0410-7334-aba8-592a3f7312c5	4	01593193b5e1b	"9e6f44844af1cf"	2026-08-18 12:31:41.316 +0200	["2026-08-18 12:31:41.316954+02",infinity)	2026-08-18 12:31:41.316 +0200	infinity
4	01a01464-0410-73b5-9024-32f451f36c53	2	06e10e1ae575c	"286dda482e61c"	2026-08-18 12:30:01.286 +0200	["2026-08-18 12:30:01.286789+02",infinity)	2026-08-18 12:30:01.286 +0200	infinity



Data Query Before Deletion



-- Query before deletion

```
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
           , timetravelid
    FROM timetravel
    LIMIT 120
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (118)
)
SELECT t.*
FROM timetravel_v AS t
INNER JOIN t2
    ON t.timetravelid = t2.timetravelid
ORDER BY rec_version
;
```

	timetravelid	rec_version	data_text	data_json	created_at	changed	valid_from	valid_until
1	01a01464-0418-7ee4-97ee-c02128d8a2b7	3	e3e7ee3204f9f7	"6ca168d44ce12	2026-08-18 12:31:41.341 +0200	["2026-08-18 12:31:41.341982+02",infinity)	2026-08-18 12:31:41.341 +0200	infinity



PostgreSQL

Delete a Record from View



-- Deleting from the view

```
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
        , timetravelid
    FROM timetravel
    LIMIT 120
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (118)
)
DELETE
FROM timetravel_v
USING t2
WHERE timetravel_v.timetravelid = t2.timetravelid
;
```



Deleted Records



-- Query the deleted records

WITH t1 AS

```
(  
    SELECT timetravelid  
    FROM timetravel  
    WHERE deleted  
)
```

SELECT count(*) FILTER (WHERE deleted) OVER () AS deleted_records

```
, t.timetravelid  
, ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY t.timetravelid,  
lower(t.changed)) AS rec_version  
, t.data_text  
, t.data_json  
, t.deleted  
, t.changed  
, lower(t.changed) AS rec_changed
```

FROM timetravel **AS** t

INNER JOIN t1

ON t.timetravelid = t1.timetravelid

ORDER BY t.timetravelid

, **lower**(t.changed)

;

	123 deleted_records	timetravelid	123 rec_version	AZ data_text	{ } data_json	deleted	changed
1	4	01a01464-0417-7c8d-ba4b-cfc59016664d	1	b63133ebae9c80a39a064da01	"3f1643e28f08f09a149c98d32"	[v]	["2026-08-18 12:28:27.704995+02",infinity)
2	4	01a01464-0418-7ee4-97ee-c02128d8a2b7	1	cae53860cfd821462c0bc3a19	"34df222d2334bd68cad412eb3"	[]	["2026-08-18 12:21:34.104939+02","2026-08-18 12:30:01.
3	4	01a01464-0418-7ee4-97ee-c02128d8a2b7	2	ed4432eb36ebb9e52ccc61c09	"8a65f1f2c9bb0264960fb6a03"	[]	["2026-08-18 12:30:01.317521+02","2026-08-18 12:31:41.
4	4	01a01464-0418-7ee4-97ee-c02128d8a2b7	3	e3e7ee3204f9f7ff2874bb017	"6ca168d44ce12ce1034fbfd15"	[v]	["2026-08-18 12:48:51.869685+02",infinity)
5	4	01a01464-044a-7366-b292-19a562fb92f4	1	5f61830d6eb3af013953fd21d	"d44aabd0c0a59c0634154f68"	[v]	["2026-08-18 12:28:27.705367+02",infinity)
6	4	01a01464-060d-708c-b695-c60475cfbb08	1	f3097f11ab96f4dbc746e8547	"d094656b622a18cc71832557e"	[v]	["2026-08-18 12:28:27.706524+02",infinity)



Data Before Update



-- Check records before update

```
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
           , timetravelid
    FROM timetravel
    LIMIT 330
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (324)
)
SELECT t.timetravelid
       , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(t.changed)) AS
rec_version
       , t.data_text
       , t.data_json
       , t.deleted
       , t.changed
       , lower(t.changed) AS valid_from
FROM timetravel AS t
INNER JOIN t2
    ON t.timetravelid = t2.timetravelid
ORDER BY lower(t.changed) DESC
;
```

	timetravelid	rec_version	data_text	data_json	deleted	changed	valid_from
1	01a01464-042d-7ba7-b53a-d8a0e20f2a71	1	88eddd1ac1d9aebb96865fdb4	"7daf993dcd0b561f186236a90"	[]	["2026-08-18 12:21:34.125737+02",infinity)	2026-08-18 12:21:34.125 +0200



Update Data With View



```
-- Update a record
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
           , timetravelid
    FROM timetravel
    LIMIT 330
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (324)
)
UPDATE timetravel_v SET
    data_text = substr(md5(random()::text), 1, 25),
    data_json = to_jsonb(substr(md5(random()::text), 1, 25))
FROM t2
WHERE timetravel_v.timetravelid = t2.timetravelid
;
```



Data After Update



-- After some updates of the same record

```
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
           , timetravelid
    FROM timetravel
    LIMIT 330
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (324)
)
SELECT now() - valid_from AS changed_diff
     , t.timetravelid
     , t.rec_version
     , t.data_text
     , t.data_json
     , t.changed
     , lower(t.changed) AS valid_from
FROM timetravel_v AS t
INNER JOIN t2
    ON t.timetravelid = t2.timetravelid
ORDER BY lower(t.changed) DESC
;
```

	timetravelid	rec_version	data_text	data_json	deleted	changed	valid_from
1	01a01464-042d-73e9-a51e-6567f3881084	5	08df558ab2869 "18ae9e6dd41fde7635435a370"		[]	["2026-08-18 15:22:50.536113+02",infinity)	2026-08-18 15:22:50.536 +0200
2	01a01464-042d-73e9-a51e-6567f3881084	4	2095db4bf8e8 "113031517c18f8d34f3e48905"		[]	["2026-08-18 15:22:27.032929+02","2026-08-18 15:22:50.536113+02")	2026-08-18 15:22:27.032 +0200
3	01a01464-042d-73e9-a51e-6567f3881084	3	98dab70dbccbf "311c11f325c5c187a42dbfaf4"		[]	["2026-08-18 12:31:41.393083+02","2026-08-18 15:22:27.032929+02")	2026-08-18 12:31:41.393 +0200
4	01a01464-042d-73e9-a51e-6567f3881084	2	013603a93d9bf "b565110b1b7bd98917003a31e"		[]	["2026-08-18 12:30:01.38209+02","2026-08-18 12:31:41.393083+02")	2026-08-18 12:30:01.382 +0200
5	01a01464-042d-73e9-a51e-6567f3881084	1	5a765d0ae949 "510779e0d2c0b2c4761b8dee4"		[]	["2026-08-18 12:21:34.125252+02","2026-08-18 12:30:01.38209+02")	2026-08-18 12:21:34.125 +0200



Data After Update



```
-- The latest one from the view, or the current active one
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
           , timetravelid
    FROM timetravel
    LIMIT 330
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (324)
)
SELECT t.timetravelid
       , t.rec_version
       , t.data_text
       , t.data_json
       , t.changed
       , lower(t.changed) AS valid_from
FROM timetravel_v AS t
INNER JOIN t2
    ON t.timetravelid = t2.timetravelid
ORDER BY lower(t.changed) DESC
;
```

	timetravelid	rec_version	data_text	data_json	changed	valid_from
1	01a01464-042d-73e9-a51e-6567f3881084	5	08df558ab2869d0f3a989a531	"18ae9e6dd41fde7635435a370"	["2026-08-18 15:22:50.536113+02",infinity)	2026-08-18 15:22:50.536 +0200



Data Statistics



```
-- Table statistics
```

```
SELECT *
```

```
FROM timetravel_stats
```

```
;
```

	123 recordcount ▼	123 unique_records ▼	123 active_records ▼	123 deleted_records ▼	123 min_version_number ▼	123 max_version_number ▼
1	2,472,339	300,000	299,996	4	1	5



Add Another Record



-- Added another entry

```
WITH t1 AS
(
    SELECT row_number() OVER (ORDER BY timetravelid) AS id
        , timetravelid
    FROM timetravel
    LIMIT 330
)
, t2 AS
(
    SELECT timetravelid
    FROM t1
    WHERE id IN (324)
)
INSERT INTO timetravel_v
SELECT t.*
FROM timetravel_v AS t
INNER JOIN t2
    ON t.timetravelid = t2.timetravelid
;
```



PostgreSQL

Statistics Before/After Another Record



```
-- Table statistics
```

```
SELECT *
```

```
FROM timetravel_stats
```

```
;
```

	¹²³ recordcount ▼	¹²³ unique_records ▼	¹²³ active_records ▼	¹²³ deleted_records ▼	¹²³ min_version_number ▼	¹²³ max_version_number ▼
1	2,472,339	300,000	299,996	4	1	5

	¹²³ recordcount ▼	¹²³ unique_records ▼	¹²³ active_records ▼	¹²³ deleted_records ▼	¹²³ min_version_number ▼	¹²³ max_version_number ▼
1	2,472,340	300,001	299,997	4	1	5



A Deleted Record



-- Remember we have deleted records?

WITH *t1* **AS**

```
(  
    SELECT row_number() OVER () AS id  
        , timetravelid  
    FROM timetravel  
    WHERE deleted  
    LIMIT 1  
)
```

```
SELECT t.timetravelid  
    , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(t.changed))  
AS rec_version  
    , t.data_text  
    , t.data_json  
    , t.deleted  
    , t.changed  
    , lower(t.changed) AS valid_from  
FROM timetravel AS t  
INNER JOIN t1  
    ON t.timetravelid = t1.timetravelid  
ORDER BY lower(t.changed) DESC  
;
```

	timetravelid	rec_version	data_text	data_json	deleted	changed	valid_from
1	01a01464-0417-7c8d-ba4b-cfc59016664d	1	b63133ebae9c80a39a064da01	"3f1643e28f08f09a149c98d32"	[v]	["2026-08-18 12:28:27.704995+02",infinity)	2026-08-18 12:28:27.704 +0200



Update Deleted Record



```
-- Update an already deleted record with the view
WITH t1 AS
(
    SELECT row_number() OVER () AS id
           , timetravelid
    FROM timetravel
    WHERE deleted
    LIMIT 1
)
UPDATE timetravel_v SET
    data_text = substr(md5(random()::text), 1, 25),
    data_json = to_jsonb(substr(md5(random()::text), 1, 25))
FROM t1
WHERE timetravel_v.timetravelid = t1.timetravelid
;
```

Name	Value
Updated Rows	0
Execute time	0.412s



Check Deleted Record



-- Check for changes

WITH *t1* **AS**

```
(  
    SELECT row_number() OVER () AS id  
        , timetravelid  
    FROM timetravel  
    WHERE deleted  
    LIMIT 1  
)
```

```
SELECT t.timetravelid  
    , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(t.changed))  
AS rec_version  
    , t.data_text  
    , t.data_json  
    , t.deleted  
    , t.changed  
    , lower(t.changed) AS valid_from  
FROM timetravel AS t  
INNER JOIN t1  
    ON t.timetravelid = t1.timetravelid  
ORDER BY lower(t.changed) DESC  
;
```

	timetravelid	rec_version	data_text	data_json	deleted	changed	valid_from
1	01a01464-0417-7c8d-ba4b-cfc59016664d	1	b63133ebae9c80a39a064da01	"3f1643e28f08f09a149c98d32"	[v]	["2026-08-18 12:28:27.704995+02",infinity)	2026-08-18 12:28:27.704 +0200



Check Deleted Record in Table



```
-- Update an already deleted with the table
WITH t1 AS
(
    SELECT row_number() OVER () AS id
           , timetravelid
    FROM timetravel
    WHERE deleted
    LIMIT 1
)
UPDATE timetravel SET
    data_text = substr(md5(random()::text), 1, 25),
    data_json = to_jsonb(substr(md5(random()::text), 1, 25))
FROM t1
WHERE timetravel.timetravelid = t1.timetravelid
;
```

Name	Value
Updated Rows	0
Execute time	0.039s



PostgreSQL

Check Deleted Record in Table



-- Result: Nothing changed

WITH *t1* **AS**

```
(
    SELECT row_number() OVER () AS id
           , timetravelid
    FROM timetravel
    WHERE deleted
    LIMIT 1
)
```

```
SELECT t.timetravelid
       , ROW_NUMBER() OVER (PARTITION BY t.timetravelid ORDER BY lower(t.changed))
AS rec_version
       , t.data_text
       , t.data_json
       , t.deleted
       , t.changed
       , lower(t.changed) AS valid_from
FROM timetravel AS t
INNER JOIN t1
    ON t.timetravelid = t1.timetravelid
ORDER BY lower(t.changed) DESC
;
```

	timetravelid	rec_version	data_text	data_json	deleted	changed	valid_from
1	01a01464-0417-7c8d-ba4b-cfc59016664d	1	b63133ebae9c80a39a064da01	"3f1643e28f08f09a149c98d32"	[v]	["2026-08-18 12:28:27.704995+02",infinity)	2026-08-18 12:28:27.704 +0200



Check Bloat



```
/**
 * Index and table bloat
 *
 * pg_table_bloat and pg_index_bloat are from https://pgsql-tweaks.org/
 */
SELECT 'table' AS type
, schema_name
, table_name
, '' AS index_name
, table_owner
, tuple_count
, dead_tuple_count
, dead_tuple_percent
, pg_size_pretty(table_length_in_bytes) AS object_size
FROM pg_table_bloat
UNION
SELECT 'index' AS type
, schema_name
, table_name
, index_name
, '' AS table_owner
, tuple_count
, dead_tuple_count
, dead_tuple_percent
, pg_size_pretty(table_length_in_bytes) AS object_size
FROM pg_index_bloat
;
```





Solve Bloat with REPACK



```
/**  
 * PostgreSQL 19 at least became a REPACK function!  
 *  
 * https://www.postgresql.org/docs/19/sql-repack.html  
 */
```

```
REPACK (CONCURRENTLY) timetravel;
```

```
/*  
 * We could run  
 REPACK timetravel;  
 * But that is not concurrently  
 */
```



SQL Error [0A000]: ERROR: REPACK (CONCURRENTLY) is not supported for partitioned tables
Hint: Consider running the command on individual partitions.



Solve Bloat with REPACK



```
/**
 * As we can not execute REPACK in a function or procedure,
 * we just return the statements, that have to be executed.
 */
CREATE OR REPLACE FUNCTION timetravel_repack()
RETURNS TEXT
LANGUAGE PLPGSQL
AS
$$
DECLARE
    -- We could also use the timetravel_part_vals table for this
    part_names CURSOR FOR
        SELECT child.relname AS p_name
        FROM pg_inherits
        INNER JOIN pg_class AS parent
            ON pg_inherits.inhparent = parent.oid
        INNER JOIN pg_class AS child
            ON pg_inherits.inhrelid = child.oid
        INNER JOIN pg_namespace AS nmsp_parent
            ON nmsp_parent.oid = parent.relnamespace
        WHERE parent.relname = 'timetravel'
        AND nmsp_parent.nspname = 'timetravel'
    ;
    part_name_rec RECORD;
    res TEXT DEFAULT '';
BEGIN

    OPEN part_names;

    LOOP
        FETCH NEXT FROM part_names INTO part_name_rec;
        EXIT WHEN NOT FOUND;

        res := res || 'REPACK (CONCURRENTLY) ' || part_name_rec.p_name || ';' || chr(10);

    END LOOP;

    CLOSE part_names;

    RETURN res;
END
$;
```



Solve Bloat with REPACK



```
SELECT timetravel_repack();
```

	○ A-Z timetravel_repack
1	REPACK (CONCURRENTLY) timetravel_202608; REPACK (CONCURRENTLY) timetravel_202607; REPACK (CONCURRENTLY) timetravel_202606;



Solve Bloat with REPACK



```
REPACK (CONCURRENTLY) timetravel_202610;  
REPACK (CONCURRENTLY) timetravel_202609;  
REPACK (CONCURRENTLY) timetravel_202608;
```

Name	Value
Updated Rows	0
Execute time	7.702s

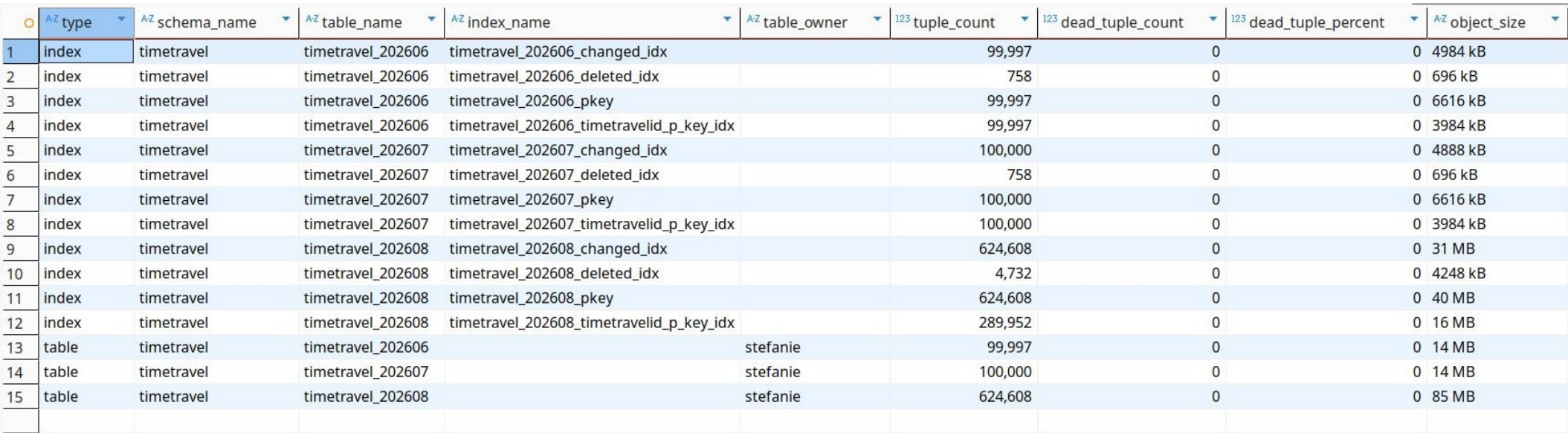


Check Bloat



-- Check bloat again

```
SELECT 'table' AS type
    , schema_name
    , table_name
    , '' AS index_name
    , table_owner
    , tuple_count
    , dead_tuple_count
    , dead_tuple_percent
    , pg_size_pretty(table_length_in_bytes) AS object_size
FROM pg_table_bloat
UNION
SELECT 'index' AS type
    , schema_name
    , table_name
    , index_name
    , '' AS table_owner
    , tuple_count
    , dead_tuple_count
    , dead_tuple_percent
    , pg_size_pretty(table_length_in_bytes) AS object_size
FROM pg_index_bloat
;
```





Record Count at Timestamps



-- Record count at different points in time

```
WITH tmx AS
(
    SELECT min(lower(t.changed)) AS min_changed
          , min(lower(t.changed)) + INTERVAL '20 Seconds' AS interval_changed
          , max(lower(t.changed)) AS max_changed
    FROM timetravel AS t
)
SELECT 'at min time' AS TYPE
     , tmx.min_changed AS changed
     , count(t.*) AS row_count
FROM tmx
INNER JOIN timetravel AS t
    ON tmx.min_changed <@ t.changed
GROUP BY tmx.min_changed
UNION
SELECT 'at interval time' AS TYPE
     , tmx.interval_changed
     , count(t.*) AS row_count
FROM tmx
INNER JOIN timetravel AS t
    ON tmx.interval_changed <@ t.changed
GROUP BY tmx.interval_changed
UNION
SELECT 'at max time' AS type
     , tmx.max_changed
     , count(t.*) AS row_count
FROM tmx
INNER JOIN timetravel AS t
    ON tmx.max_changed <@ t.changed
GROUP BY tmx.max_changed
ORDER BY 2
;
```



Record Count at Timestamps



	A-Z type	🕒 changed	123 row_count
1	at min time	2026-08-18 12:21:34.095 +0200	1
2	at interval time	2026-08-18 12:21:54.095 +0200	199,997
3	at max time	2026-08-18 15:39:20.122 +0200	300,002



PostgreSQL

Active Records and Statistics



-- Comparing the view with active records with the statistics

```
SELECT 'table statistics' AS data_source
```

```
, *
```

```
FROM timetravel_stats
```

```
UNION
```

```
SELECT 'view statistics' AS data_source
```

```
, count(*)
```

```
, count(*)
```

```
, count(*)
```

```
, 0
```

```
, min(rec_version)
```

```
, max(rec_version)
```

```
FROM timetravel_v
```

```
ORDER BY 1 DESC
```

```
;
```

	AZ data_source	123 recordcount	123 unique_records	123 active_records	123 deleted_records	123 min_version_number	123 max_version_number
1	view statistics	299,998	299,998	299,998	0	1	5
2	table statistics	2,472,341	300,002	299,998	4	1	5



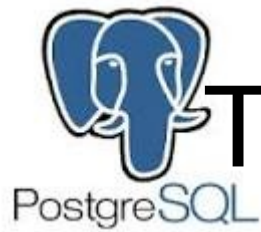
Link List



Slide and sources are available on Codeberg:

<https://codeberg.org/sjstoelting/talks/>





Time Traveling PostgreSQL



This document by [Stefanie Janine Stölting](#) is covered by the [Creative Commons Attribution 4.0 International](#)