Implementing Temporal Features in PostgreSQL: SQL Standard and Beyond

DRW

Hettie Dombrovskaya
Database Architect

PG Day Chicago 2024

Who am I

Hettie Dombrovskaya

Database Architect at DRW
Local Organizer of Chicago PostgreSQL User Group

PG Day Chicago Operations Committee

What is this talk about?

- 10+ years of bitemporal model development
- (Almost) no traction in Postgres community
- First elements of temporality coming in PG 17
- Standard – Postgres – Bitemporal model

I want it to be done right!

What will be covered

- Temporal DB Basics
- Temporal features in the SQL standard
- Postgres support for temporal data
- Bitemporal model
- Features comparison
- What is missing
- How to do it right

The Four Things I Would Most Like to See in Postgres - Michael Stonebraker

POSTGRES VISION 2022

Outline

- Time travel
- Cloud
- Warehouses
- User-defined functions (UDFs)

Temporal DB Basics

Why Time Travel?

- ✓ Point-in-time (snapshot) queries: How my report looked on the last day of previous month?
- ✓ Change Log: When and how the state of my request was changed?
- ✓ Fully Temporal:
  - ❑ When these objects co-exist?
  - ❑ Before/after/meets etc.
  - ❑ Temporal joins, aggregations etc.

Potential solutions

All are equivalent

but query performance may differ

Snapshots

- Any theoretical paper starts with it

Event Logs

- Often produced by applications with a hope that they will be used for analysis in the future

Periods

- Used by everyone who actually implements temporal features in a database

Time Dimensions

This is something beyond common sense in the real life, however, we need them

Transactional (system)

Valid = Effective

Asserted- transactional

Much more are defined but not widely known

Temporal Features in the SQL Standard

The Standard:

- ✓ Does not introduce a period type, instead, a pair of timestamp columns can represent the period
- ✓ Defines period as closed-open
- ✓ Supports period predicates: OVERLAPS, MEETS, etc. (Similar but not equivalent to Allen operators)
- ✓ Supports System time (Transaction Time) and/or Application Time (Valid Time)

Types and Predicates

System Time

- System-versioned tables, the name SYSTEM_TIME is fixed.
- Before and now, never in the future
- Can never be changed
- Does NOT require separate table for historical data, although some implementations do that
- Default – CURRENT record

Application (Valid) Time

- Maintained by the user
- Column names can be arbitrary
- No semantics are specified
- Currently at most one additional time dimension can be specified

Primary Keys

The Standard does not provide clean resolution for the PK

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-09-10	3
22217	2010-02-03	2011-11-12	4

Triple (ENo, EStart, EEnd) does not work

Instead, it suggests “no overlap”

```
ALTER TABLE Emp
   ADD PRIMARY KEY (ENo,
    EPeriod WITHOUT OVERLAPS)
```

Referential integrity constraints should be time-aware

The example below won't work:

ENo	EStart	EEnd	EDept
22218	2010-01-01	2011-02-03	3
22218	2011-02-03	2011-11-12	4

DNo	DStart	DEnd	DName
3	2009-01-01	2011-12-31	Test
4	2011-06-01	2011-12-31	QA

```
ALTER TABLE Emp
   ADD FOREIGN KEY
      (Edept, PERIOD EPeriod)
REFERENCES Dept
      (DNo, PERIOD DPeriod)
```

Foreign Keys

Syntax extensions for INSERT, UPDATE, DELETE to specify period(s)

```
UPDATE Emp
 FOR PORTION OF EPeriod
   FROM DATE '2011-02-03'
   TO DATE '2011-09-10'
SET EDept = 4
WHERE ENo = 22217
```

Syntax extensions for SELECT (add FOR to SELECT)

```
SELECT Ename, Edept
FROM Emp
WHERE ENo = 22217 AND
  EPeriod CONTAINS DATE '2011-01-02'
```

Queries and DML

PostgreSQL Support for Temporal Data

What is Available

- ✓ Period data types are provided although are not required by the Standard
- ✓ Rich set of operators and functions for timestamps, intervals, and periods
- ✓ Predicates are implemented as required by the Standard (including closed-open semantics)

Additionally:

- ✓ GIST indexes
- ✓ GIST with exclusion constraints (solve temporal PK problem)
- ✓ PG 17: temporal PK/UQ

DRW

Language extensions:

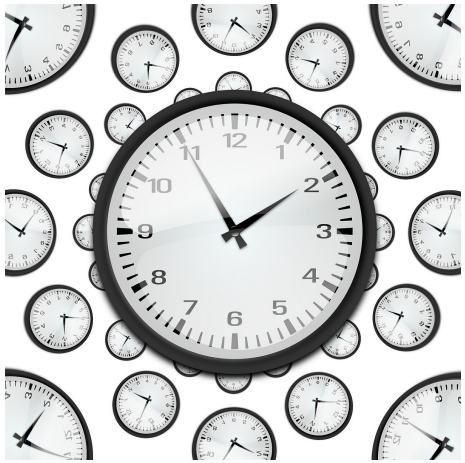
- ✓ CREATE temporal table
- ✓ SELECT within time period (except for adding explicit condition)
- ✓ Modified INSERT/UPDATE/DELETE syntax

What is Missing

DRW

Bitemporal Implementation

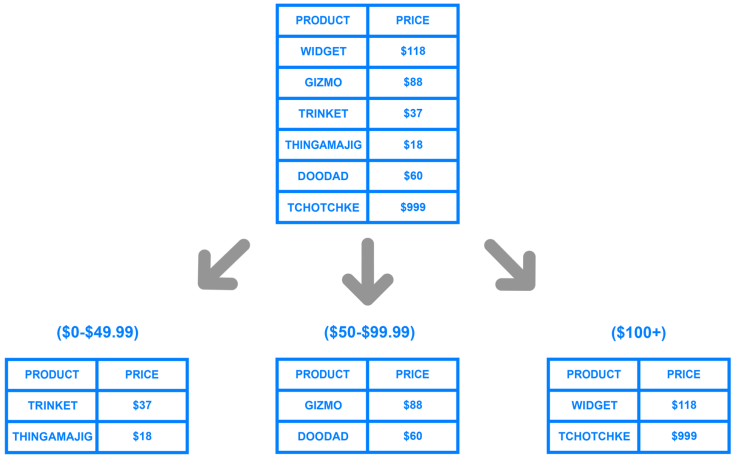
- ✓ Supports asserted and effective time dimensions
- ✓ Transactional dimension can be derived from row_created_at timestamp, but is not explicitly supported
- ✓ Heavily relies on PostgreSQL features
- ✓ Does not provide any syntax extensions
- ✓ Data manipulation is implemented with user-defined functions
- ✓ Provides detailed refinement of data manipulation semantics
- ✓ Supports temporal integrity constraints



Overview of Bitemporal Model

Storage and Indexes

- ✓ Bitemporal primary keys (business keys) are defined as keys with NO OVERLAP utilizing GIST with exclusion constraints.
- ✓ GIST indexes make bitemporal search efficient



Queries

- ✓ Time-related conditions must be explicitly specified (acceptable in the Standard)
- ✓ Built-in predicates (INCLUDES, OVERLAPS, etc.) are helpful
- ✓ Fully temporal queries are still tricky (we don’t know how to write them!)

The refinements of manipulation semantics

- ✓ INSERT
- ✓ UPDATE
- ✓ CORRECTION
- ✓ INACTIVATE
- ✓ DELETE

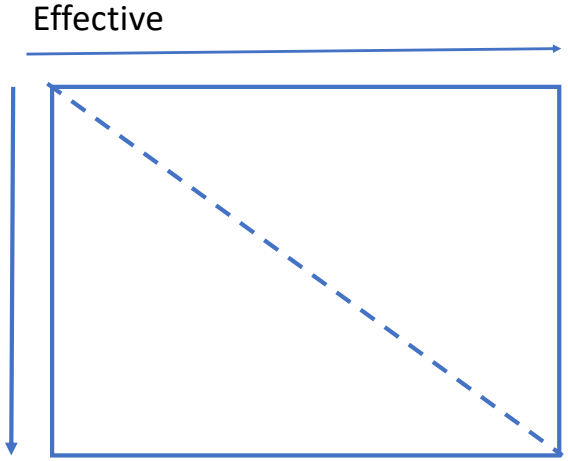
Data Manipulation

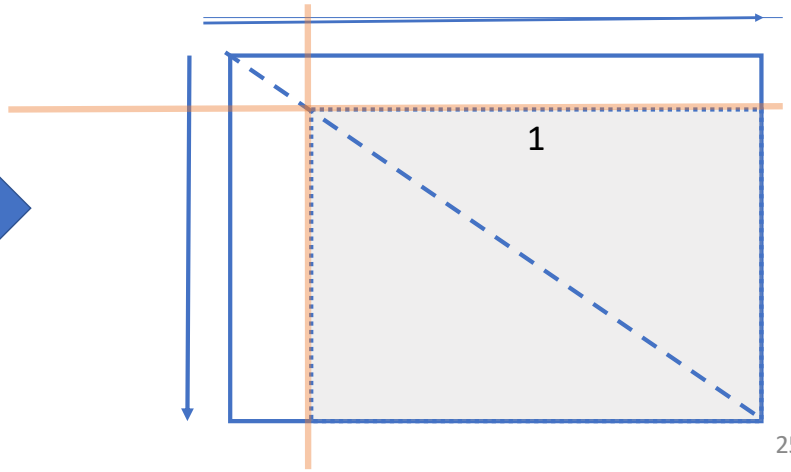
Bitemporal Insert

now = 2022-05-01

```
select ll_bitemporal_insert(
        'customers',
        $$'customer_no', name', 'type' $$,
        $$'C100','John Doe', 'Silver' $$,
        timeperiod('2022-06-01','infinity'),
        timeperiod('2022-05-01','infinity'))
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2022-06-01, oo)	[2022-05-01 , oo)	C100	John Doe	Silver



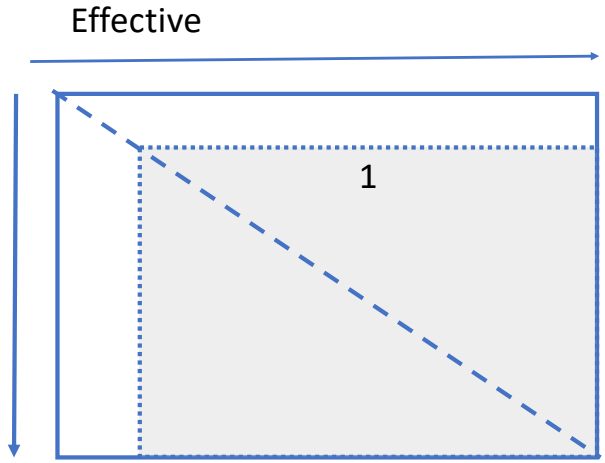


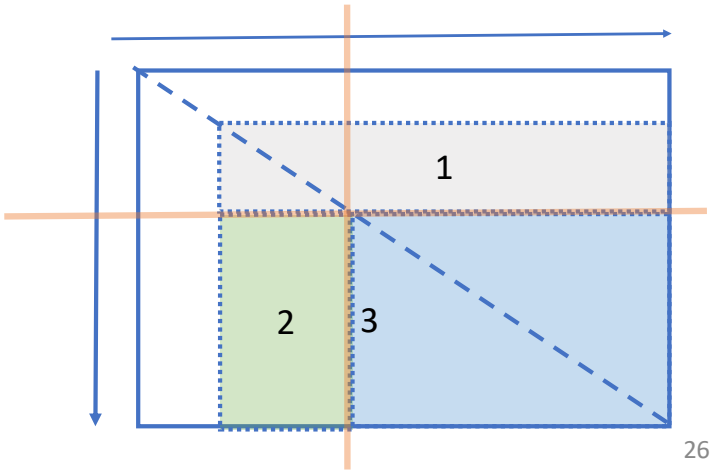
Bitemporal Update

now = 2022-09-15

```
select ll_bitemporal_update($$customers$$,
$$customer_no$$, $$100$$,
$$type$$, $$Gold$$,
timeperiod('2022-09-15', 'infinity'),
timeperiod('2022-09-15', 'infinity'))
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2022-06-01, oo)	[2022-05-01,2022-09-15)	C100	John Doe	Silver
2	[2022-06-01,2022-09-15)	[2022-09-15, oo)	C100	John Doe	Silver
3	[2022-09-15, oo)	[2022-09-15, oo)	C100	John Doe	Gold



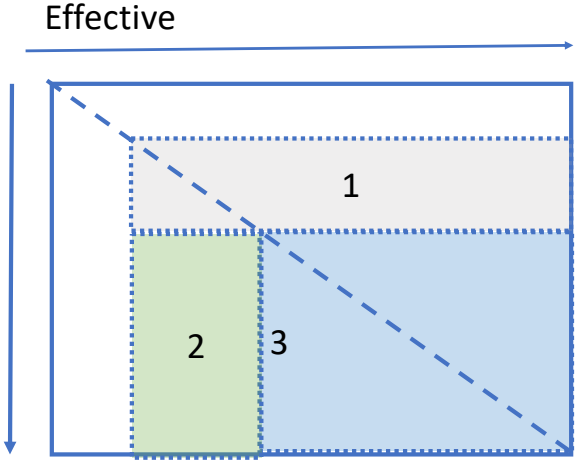


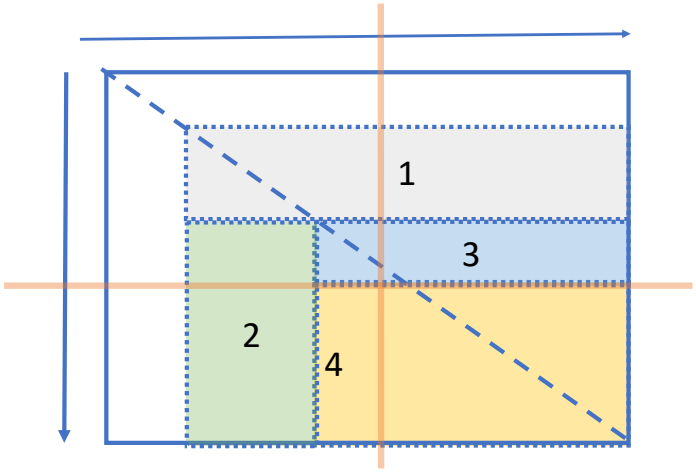
Bitemporal Correction

now = 2022-09-22

```
select ll_bitemporal_correction($$customers$$,
        $$type $$,
        $$ Platinum$$,
        $$ customer_no $$,
        $$ C100$$,
        timeperiod('2022-09-15','infinity'),
        now())
```

Effective Interval	Assertive Interval	Customer No.	Name	Type
[2022-06-01, oo)	[2022-05-01,2022-09-15)	C100	John Doe	Silver
[2022-06-01,2022-09-15)	[2022-09-15, oo)	C100	John Doe	Silver
[2022-09-15, oo)	[2022-09-15, 2022-09-22)	C100	John Doe	Gold
[2022-09-15, oo)	[2022-09-22, oo)	C100	John Doe	Platinum



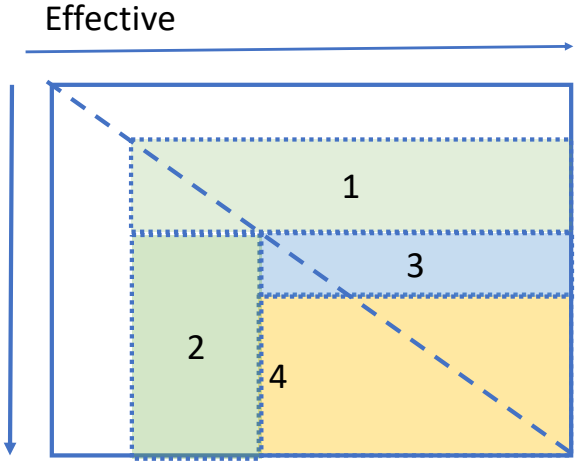


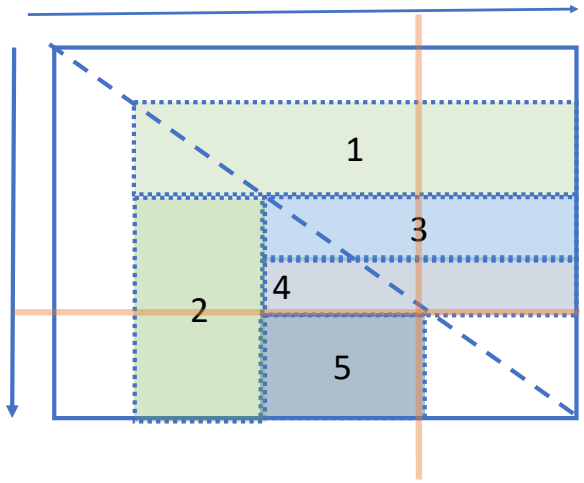
Bitemporal Inactivate

now = 2022-11-05

```
select ll_bitemporal_inactivate(
      $$customers$$,
      $$customer_no$$,
      $$C100$$,
      timeperiod('2022-12-31','infinity'),
      timeperiod('2022-11-05','infinity'),
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2022-06-01, oo)	[2022-05-01,2022-09-15)	C100	John Doe	Silver
2	[2022-06-01,2022-09-15)	[2022-09-15, oo)	C100	John Doe	Silver
3	[2022-09-15, oo)	[2022-09-15, 2022-09-22)	C100	John Doe	Gold
4	[2022-09-15, oo)	[2022-09-22, 2022-11-05)	C100	John Doe	Platinum
5	[2022-09-15,2022-12-31)	[2022-11-05, oo)	C100	John Doe	Platinum



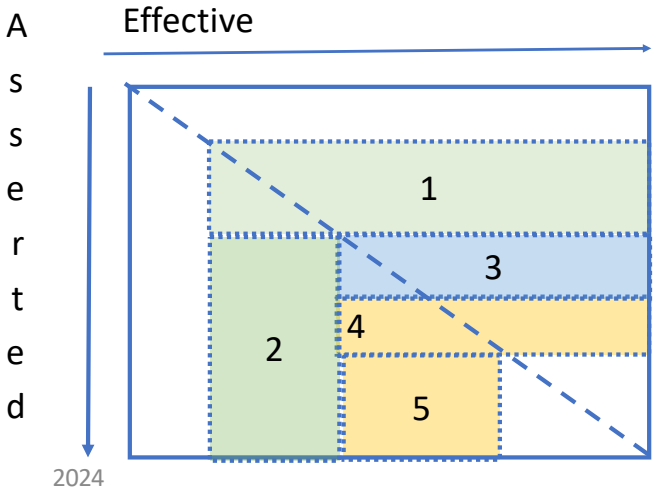


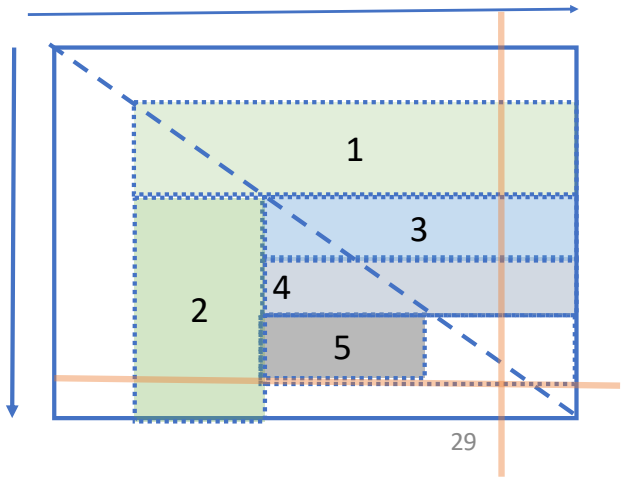
Bitemporal Delete

now = 2022-11-17

```
select ll_bitemporal_delete(
        'customers',
        $$ customer_no $$,
        $$ 'C100' $$,
        timeperiod('2022-11-17','infinity'))
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2022-06-01, oo)	[2022-05-01,2022-09-15)	C100	John Doe	Silver
2	[2022-06-01,2022-09-15)	[2022-09-15, oo)	C100	John Doe	Silver
3	[2022-09-15, oo)	[2022-09-15, 2022-09-22)	C100	John Doe	Gold
4	[2022-09-15, oo)	[2022-09-22, 2022-11-05)	C100	John Doe	Platinum
5	[2022-09-15,2022-12-31)	[2022-11-05, 2022-11-17)	C100	John Doe	Platinum





Bitemoral vs. Standard Comparison

Feature	Standard	Bitemporal
Period type	Not required	✓
Open/close semantics and predicates	✓	✓
SYSTEM_TIME	✓	Implicit
APPLICATION_TIME	✓	✓
ASSERTED_TIME	Semantics not specified	✓
Modified SQL syntax	✓	No
(bi) temporal PK	✓	✓
Referential integrity constraints	✓	✓

What is Sill Missing?

What is Missing in the Standard

- ✓ Support for more than one application time dimension, which means no support for:
  - Future assertion
  - UPDATE vs. CORRECTION semantics
- ✓ Comprehensive temporal JOIN support
  - Calculating result dimensions properly
  - OUTER temporal joins
- ✓ Period AGGREGATE support

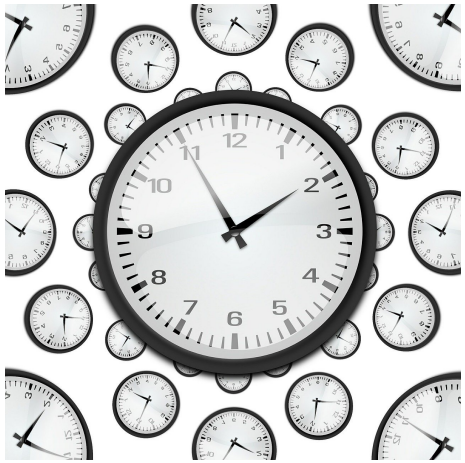
What is Missing in PostgreSQL

Of course, bitemporal!

Actually, there is no SQL-supported temporality

PG 17 makes only first steps towards that goal

What is Missing in Bitemporal



- ✓ Syntax extensions
- ✓ Design methodologies (also missing in the Standard)
- ✓ Explicit support for transactional time dimension, although transactional dimension is not really needed (the Standard identifies SYSTEM_END as the time when a record stops being CURRENT)

Conclusion

Past, Present, and Future of Temporal Databases

- The SQL Standard provides very reasonable conservative support of Temporal tables
- PostgreSQL contains everything that is needed for efficient implementation but not an implementation
- Bitemporal is not too far from the Standard
- Find more: https://github.com/hettie-d/pg_bitemporal

Q&A

Hettie Dombrovskaya
Database Architect DRW

hdombrovska@drwholdings.com

www.drw.com