Implementing Temporal Features in PostgreSQL: SQL Standard and Beyond

Hettie Dombrovskaya DRW
Boris Novikov

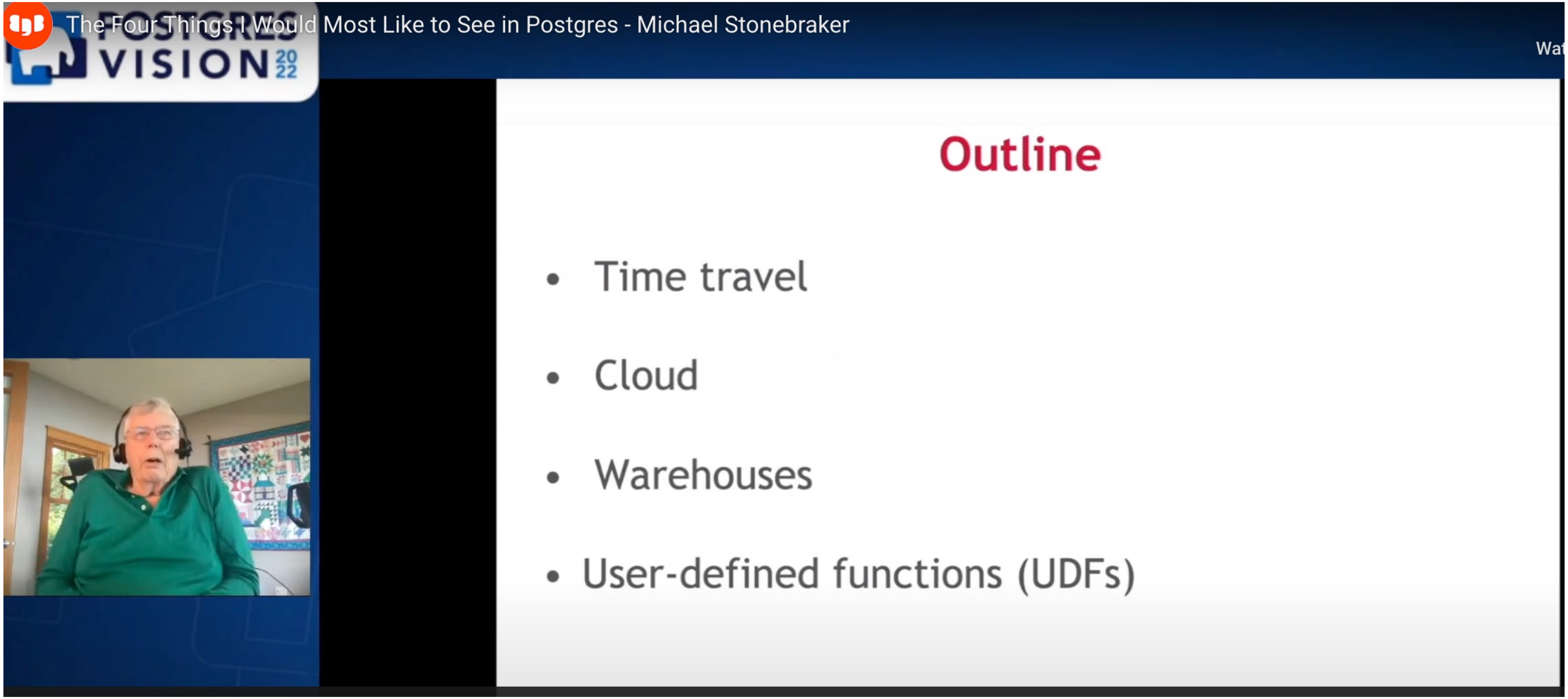
Who am I

Hettie Dombrovskaya

Database Architect at DRW Holdings Chicago IL

Local Organizer of Chicago PostgreSQL User Group

Temporal DB Basics


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)

2023

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

- ✓ 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

All are equivalent but query performance may differ

Time Dimensions

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

- Transactional (system)
- Valid = Effective
- Asserted ~ transactional

Much more were defined but are not widely known

Temporal Features in the SQL Standard

Types and Predicates

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)

System (Transaction) 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)
```

Foreign Keys

Referential integrity constraints should be time-aware, the example below won't work:

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-09-10	3
22217	2010-02-03	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)
```

Queries and DML

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

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

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)

What is Missing

Language extensions

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

Bitemporal Implementation

Overview of Bitemporal Model

- 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

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

Data Manipulation

The refinements of manipulation semantics

- ❑ INSERT
- ❑ UPDATE
- ❑ CORRECTION
- ❑ INACTIVATE
- ❑ DELETE

Bitemporal vs. SQL 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 Still 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
- ✓ Period JOIN support
- ✓ Period AGGREGATE support

What is missing in PostgreSQL

Of course, bitemporal!

Actually, there is no SQL-supported temporality

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