I implemented the first “Vacuum Cleaner Daemon” in Postgres (the most hated thing about Postgres)

Curt Kolovson

Postgres Conference Silicon Valley 2024

Original Design of Postgres Storage System (1987)

- Keep all past states of tuples – immutable records
- Enables queries on “historical” (temporal) data, including “time travel”
- Periodic “vacuuming” of historical data to archive storage (WORM drives)
- Indexing technique(s) for searching historical data

NOTE:

- Original Vacuum Cleaner had nothing to do with MVCC (added to PostgreSQL v6.5.0, 1999)

But…

- There were performance issues with original historical data implementation in Postgres.
- It was thought that keeping past versions of data was impractical, even with periodic vacuuming.
- Was eventually deprecated.

What was my role in all this

- Implemented the early version of the “Vacuum Cleaner Daemon”

- Developed indexing techniques for temporal and spatial data:
  - Curtis P. Kolovson, Michael Stonebraker: **Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data.** SIGMOD Conference 1991: 138-147
  - Curtis P. Kolovson, Michael Stonebraker: **Indexing Techniques for Historical Databases.** ICDE 1989: 127-137

- My PhD thesis: “Indexing Techniques for Multidimensional Spatial Data and Historical Data in Database Management Systems” (1990)

That was then…

… Let’s talk about the future!!

Let’s talk about… Bitemporal Data Model

- Two time dimensions – Effective Time and Asserted Time (*)
- Effective Time – when a tuple is effective in the real world
  - INACTIVATE means no longer effective
- Asserted Time – when a tuple exists in the database
  - Asserted Time can be now or a future time
  - Generally, Asserted Time is system time (transaction commit time) to support Time Travel and point-in-time recovery
  - CORRECTION means update but keep past state
- Primary key is “Business” key(s) + Effective Time + Asserted Time
  - Has implications for uniqueness RI and FK constraints
- Time periods are [closed, open). +/- ∞ are possible values

(*) Ref: Tom Johnston, Randall Weis, *Managing Time in Relational Databases*, 2010.

Why is the Bitemporal Data Model important?

- Useful construct for many classes of applications
  - Accounting / Auditing
  - Financial services
  - Scientific & Medical applications
  - Security / Auditing (ABAC, RBAC, PBAC)
  - Commercial enterprises
- Three main use cases
  - Time Travel (past and future)
  - Bitemporal queries
  - Changelogs

Why revisit now?

- Need is as great as ever
- Compute and storage resources are vastly better than 1980s-90s
- High value of data
- Data retention / Compliance regulations
- High Availability / Disaster Recovery
- Build it and they will come
- Add to the maturation and widespread adoption of PostgreSQL

- Stonebraker's closing keynote at EDB PostgresVision 2022…
  → "The Four Things I Would Most Like to See in Postgres"

Example query

Find current (effective and asserted) event records for a given customer

```
select *
from postgres_air_bitemp.frequent_flyer_transaction
where
   frequent_flyer_id=39189
   and now()<@asserted
   and now()<@effective
```

```
--------

 frequent_flyer_transaction_key | frequent_flyer_id | level | booking_leg_id | award_points | status_points |
effective              |              asserted               |        row_created_at
-------------------------------+-------------------+-------+----------------+--------------+---------------+-----------
-------------------------+-----------------------------------+-------------------------------
                        2011298 |             39189 | 2     |        6204494 |        67064 |         13564 |
["2022-07-19 23:40:00-07",infinity) | ["2022-07-19 23:40:00-07",infinity) | 2022-10-02 21:12:50.071493-07
(1 row)
```

Example query – individual history

Find frequent flier event history for a given customer

```
select frequent_flyer_id,
  lower(effective) as start_time,
  booking_leg_id,
  departure_airport,
  arrival_airport,
  award_points,
  status_points,
  level
from postgres_air_bitemp.frequent_flyer_transaction t
  join postgres_air.booking_leg bl using (booking_leg_id)
  join postgres_air.flight f using (flight_id)
where frequent_flyer_id=39189
  and now()<@asserted
  order by lower(effective)
```

Results

frequent_flyer_id	start_time	booking_leg_id	departure_airport	arrival_airport	award_points	status_points	level
39189	2022-03-12 19:43:06.483092-08	17463738	CDG	BOS	47064	63864	1
39189	2022-04-03 03:23:31.982721-07	2318694	NRT	KUL	30564	63864	1
39189	2022-04-03 03:23:34.982721-07	2318697	CPH	FRA	28464	63864	1
39189	2022-04-30 17:08:16.48465-07	6204495	KBP	GME	27564	63864	1
39189	2022-05-31 04:14:53.04-07	17463737	TNR	CDG	36364	72664	1
39189	2022-06-08 10:12:58.08-07	17463739	BOS	CDG	41964	78264	1
39189	2022-06-09 09:23:43.08-07	17463740	CDG	TNR	50764	87064	1
39189	2022-06-23 06:57:37.68-07	2318695	KUL	FRA	60764	97064	1
39189	2022-06-24 06:18:00.72-07	2318696	FRA	CPH	61464	97764	1
39189	2022-07-01 02:09:42.981964-07	10874409	EWR	MSY	55764	97764	1
39189	2022-07-04 09:40:00-07	2318698	FRA	KUL	65764	7764	2
39189	2022-07-05 05:25:00-07	2318699	KUL	NRT	71264	13264	2
39189	2022-07-13 07:07:08.233812-07	12743928	ORD	DEN	66764	13264	2
39189	2022-07-19 23:40:00-07	6204494	GME	KBP	67064	13564	2

(14 rows)

Why should bitemporal data be in PostgreSQL core?

- Performance at scale
- A more elegant standards compliant syntax
- Keep pushing the boundaries of PostgreSQL functionality
- Security / Data Integrity
- Most major commercial RDBMSs have it

To support the Bitemporal Model, we need…

- Query language enhancements
- Create bitemporal table (all CRUD ops)
- Immutable tuples (non-temporal attributes)
- Bitemporal predicates
  - EFFECTIVE AS OF <time> (or <period>)
  - ASSERTED AS OF <time> (or <period>)
- Bitemporal operations – insert, update, correction, inactivate, delete
- Period aggregates – over a time period
- Period join
  - Equal, after, overlap, contain, during, intersection, start, finish, meet, before, left-overlap, right-overlap
- Bitemporal Integrity Constraints
  - Recent contributions by Peter Eisentraut and Paul Jungwirth
- Bitemporal indexes (btree_gist and possibly others)
- Point-in-time recovery
- Periodic migration of old tuples to archival storage

- Come to my talk on Adding Bitemporal Data to PostgreSQL Core at the Meetup.
- Watch Hettie's talk: Implementing Temporal Features in PostgreSQL (CitusCon 2023).

Don’t blame me for the current VACUUM in PostgreSQL

- Original Vacuum Cleaner migrated old versions to archival storage
- Current AUTOVACUUM does garbage collection for MVCC
- MVCC didn’t exist in original “University Postgres”
- See: “The part of PostgreSQL we hate the most”, by Bohan Zhang and Prof. Andy Pavlo for what’s wrong with the current VACUUM
    - https://ottertune.com/blog/the-part-of-postgresql-we-hate-the-most

Related Work on Temporal Data

- Ideas go back decades – at least since 1980s, and probably earlier
- Prof Richard Snodgrass et al – work on TSQL, temporal operators
- SQL Standard added bitemporal data concepts in 2011
- Most major commercial RDBMSs have temporal data support (Wikipedia):
  - MariaDB version 10.3.4 added support for SQL:2011 standard as "System-Versioned Tables".[11]
  - Oracle Database – Oracle Workspace Manager is a feature of Oracle Database which enables application developers and DBAs to manage current, proposed and historical versions of data in the same database.
  - Teradata provides two products. Teradata version 13.10 and Teradata version 14 have temporal features based on TSQL2[14] built into the database.
  - IBM Db2 version 10 added a feature called "time travel query"[2] which is based on the temporal capabilities of the SQL:2011 standard.[1]
  - Microsoft SQL Server introduced Temporal Tables as a feature for SQL Server 2016. The feature is described in a video on Microsoft's "Channel 9" web site.[15]

Call to action

- Let’s coordinate and collaborate on temporal related work in PostgreSQL

- Current work on PK constraints looks promising…
  - Correctness of implementation depends on assumptions and precise definitions.
  - Only deals with one temporal dimension. Bitemporal model requires two.
  - Should not be possible to update or insert into the past.
  - System or Valid time? Test cases seem to confuse them.

Acknowledgements

- **Henrietta (“Hettie”) Dombrovskaya** – for her comprehensive work on bitemporal data for PostgreSQL

And of course

- **Prof Michael Stonebraker** (my PhD advisor), whose design of Postgres (and so many other things) have changed the world, and continue to do so (e.g., DBOS)

Closing thoughts

You can’t change the past. You can only change the present and future.

You can always make more money, but you can’t make more time.

So don’t waste your time.