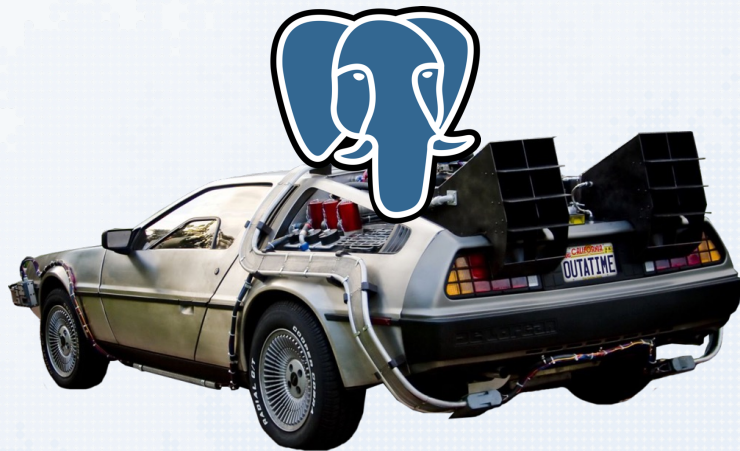


Time Travel Queries with Postgres

Qian Li
Co-Founder @DBOS, Inc

Twitter: @qianl_cs
LinkedIn: qianli-dev
Email: qian.li@dbos.dev



Once Upon a Time...

Looking Back at Postgres

Joseph M. Hellerstein
hellerstein@berkeley.edu

ABSTRACT

This is a recollection of the UC Berkeley Postgres project, which was led by Mike Stonebraker from the mid-1980's to the mid-1990's. The article was solicited for Stonebraker's Turing Award book [Bro19], as one of many personal/historical recollections. As a result it focuses on Stonebraker's design ideas and leadership. But Stonebraker was never a coder, and he stayed out of the way of his development team. The Postgres codebase was the work of a team of brilliant students and the occasional university "staff programmers" who had little more experience (and only slightly more compensation) than the students. I was lucky to join that team as a student during the latter years of the project. I got helpful input on this writeup from some of the more senior students on the project, but any errors or omissions are mine. If you spot any such, please contact me and I will try to fix them.

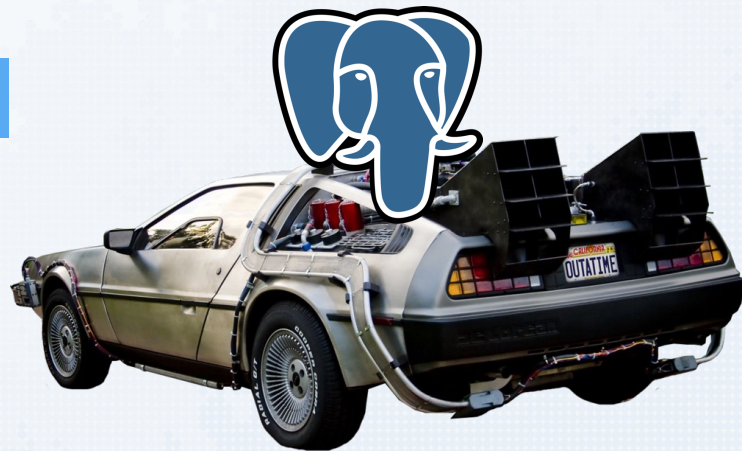
etc." "efficient spatial searching" "complex integrity constraints" and "design hierarchies and multiple representations" of the same physical constructions [SRG83]. Based on motivations such as these, the group started work on indexing (including Guttman's influential R-trees for spatial indexing [Gut84], and on adding Abstract Data Types (ADTs) to a relational database system. ADTs were a popular new programming language construct at the time, pioneered by subsequent Turing Award winner Barbara Liskov and explored in database application programming by Stonebraker's new collaborator, Larry Rowe. In a paper in SIGMOD Record in 1983 [OFS83], Stonebraker and students James Ong and Dennis Fogg describe an exploration of this idea as an extension to Ingres called ADT-Ingres, which included many of the representational ideas that were explored more deeply—and with more system support—in Postgres.

...They Didn't Live Happily Ever After

Time travel is deprecated: The remaining text in this section is retained only until it can be rewritten in the context of new techniques to accomplish the same purpose.

Volunteers? - thomas 1998-01-12

DBOS Time Travel Demo!



```
widget_store=> select product, inventory from products;
```

product	inventory
Premium Quality Widget	92

```
(1 row)
```

```
widget_store=> DBOS TS "2024-09-30T16:30:00-04:00";  
widget_store=> select product, inventory from products;
```

product	inventory
Premium Quality Widget	93




```
(1 row)
```

```
widget_store=> DBOS TS "2024-09-30T16:20:00-04:00";  
widget_store=> select product, inventory from products;
```

product	inventory
Premium Quality Widget	98

```
(1 row)
```

```
widget_store=> █
```

 DBOS Debug... node Task ✓ Time Travel Audit

Time Travel Is Useful

- Periodic reporting
- Auditing
- Debugging
- Regulatory Compliance (GDPR, CCPA, ...)
- Recovery

Many Existing Implementations

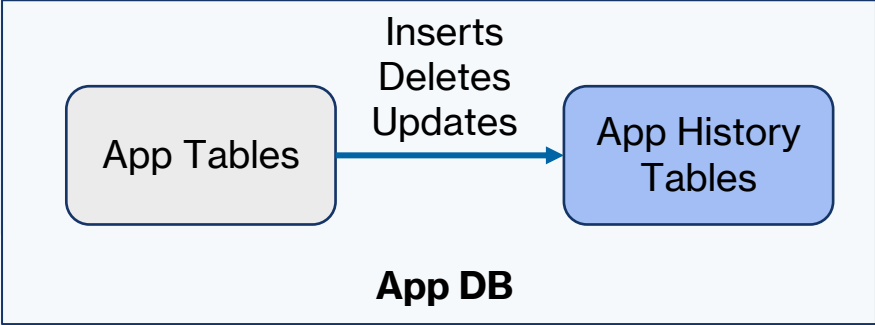
Example 41.4. A PL/pgSQL Trigger Function for Auditing

This example trigger ensures that any insert, update or delete of a row in the `emp` table is recorded (i.e., audited) in the `emp_audit` table. The current time and user name are stamped into the row, together with the type of operation performed on it.

```
CREATE TABLE emp (  
    empname      text NOT NULL,  
    salary       integer  
);  
  
CREATE TABLE emp_audit(  
    operation     char(1) NOT NULL,  
    stamp        timestamp NOT NULL,  
    userid       text NOT NULL,  
    empname      text NOT NULL,  
    salary       integer  
);  
  
CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS $emp_audit$  
BEGIN
```

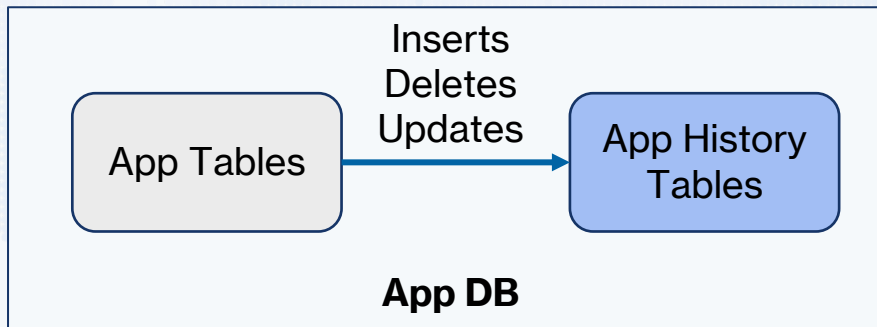
we're forced to make it ourselves.

Why New Implementation?



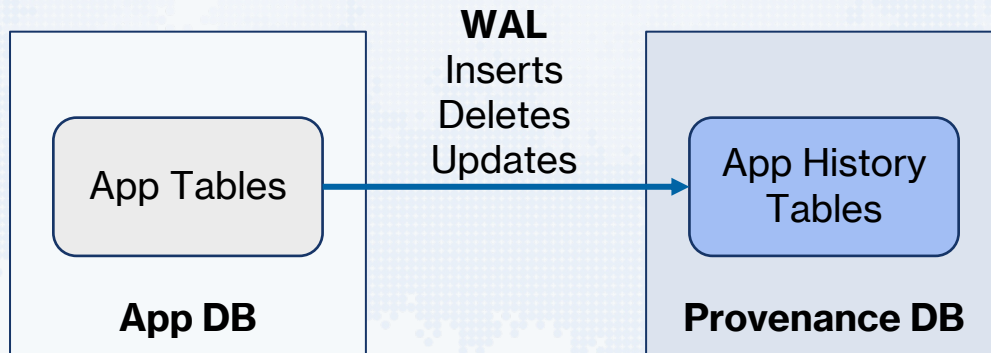
Why New Implementation?

- Store all past versions in the **same app database**
 - Performance impact
 - Hard to maintain



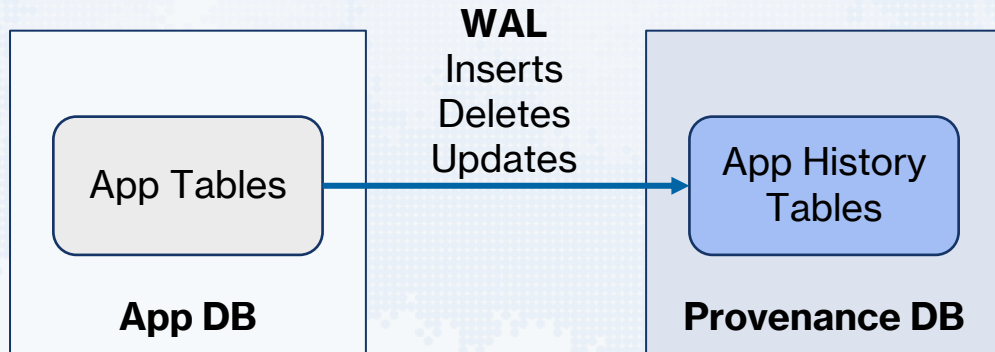
DBOS Time Travel

- Export history data to a **separate provenance DB**
- Track changes per **transaction**
- Main idea: Logical replication + multi-versioning



Benefits

- No impact on the app DB
- Safe schema migration
- Work with off-the-shelf/managed Postgres servers
- Bonus: Enable transaction debugging



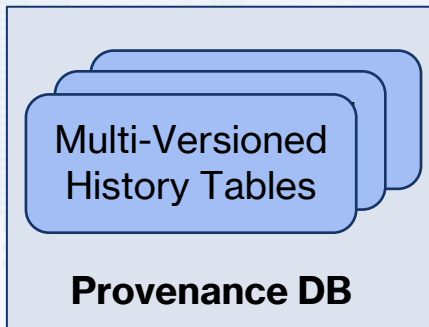
Main Components

- Multi-versioned WAL exporter
- Time travel proxy
- Time travel debugging

Multi-Versioned WAL Exporter

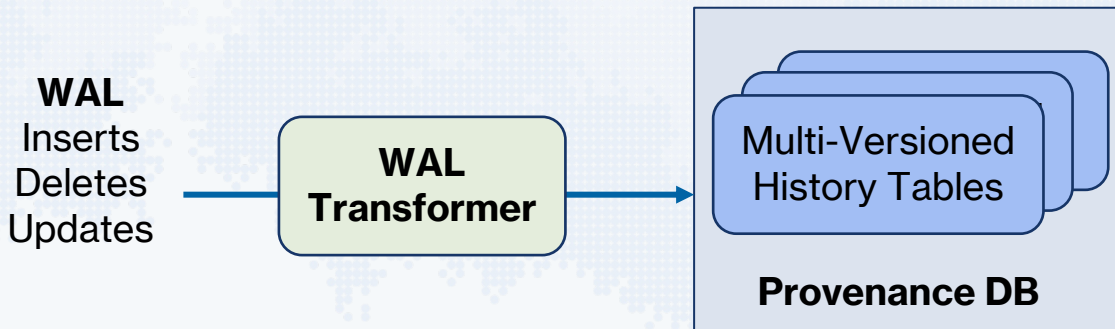
Main Idea

- Multi-versioned tables
 - Capture all versions of each data record
 - Capture the **begin** and **end** timestamp of each version



Main Idea

- Multi-versioned tables
 - Capture all versions of each data record
 - Capture the **begin** and **end** timestamp of each version
- Write-Ahead Log (WAL) transformer



Multi-Versioned History Tables

- Use transaction ID as the logical timestamp
- Extend each table with two columns
 - `begin_xid`: **added** the record
 - `end_xid`: **deleted** or **updated** the record with a new version
- Each version is *visible* between `begin_xid` and `end_xid`

Example

App Table: Current data

product	inventory
Premium Quality Widget	98

Provenance Table: History data

product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Widget Store



Premium Quality Widget — Only **98** left!

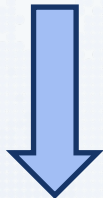
Enhance your productivity with our top-rated widgets!

[Buy Now for \\$99.99](#)

Example

App Table: Current data

product	inventory
Premium Quality Widget	98



How?

Provenance Table: History data

product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Widget Store



Premium Quality Widget — Only **98** left!

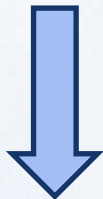
Enhance your productivity with our top-rated widgets!

[Buy Now for \\$99.99](#)

Example

App Table: Current data

product	inventory
Premium Quality Widget	98

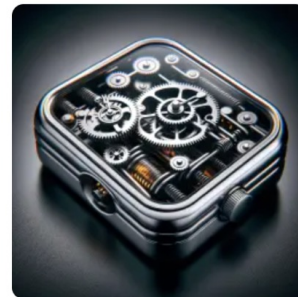


WAL to the rescue!

Provenance Table: History data

product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Widget Store



Premium Quality Widget — Only **98** left!

Enhance your productivity with our top-rated widgets!

Buy Now for \$99.99

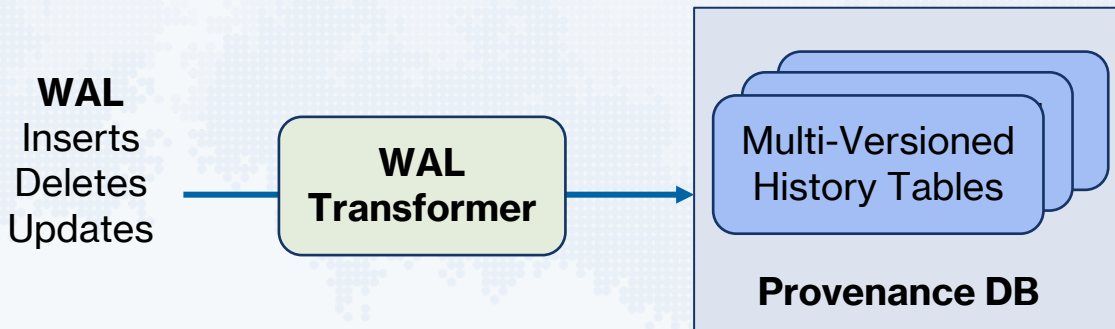
Write-Ahead Log (WAL)

- WAL describes data changes
- Logical decoding (e.g., wal2json) converts WAL to a readable format
- Example:

```
{  
  "xid": "24818",  
  "kind": "insert",  
  "schema": "public",  
  "table": "products",  
  "columnnames": ["product", "inventory"],  
  "columntypes": ["text", "integer"],  
  "columnvalues": ["Premium Quality Widget", 100]  
}
```

WAL Transformer

- Think of an ETL pipeline
- Enhance WAL with version info and update provenance tables
 - Records are append-only
 - Only metadata can be modified



WAL Transformer: Insert

- For an insert, append the new record to the table
 - begin_xid set to the transaction ID
 - end_xid to infinity (latest version)
- Example: add a new product

product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	∞

WAL Transformer: Delete

- For a delete, find the latest record ($\text{end_xid}=\infty$)
 - Update end_xid to the transaction ID
- Example: delete a product

product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	∞




product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824

WAL Transformer: Update

- For an update, first perform a delete and then insert a new version
- Example: update a product's inventory

product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	∞



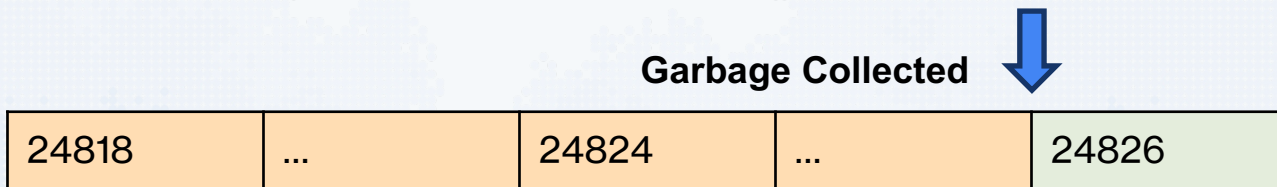
product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	∞

Garbage Collection

- Bound the size of the provenance DB
- Retention policy
- Periodically remove old versions based on **end_xid**

Garbage Collection

- Periodically remove old versions based on **end_xid**
- Example:



product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Time Travel Proxy

Main Idea

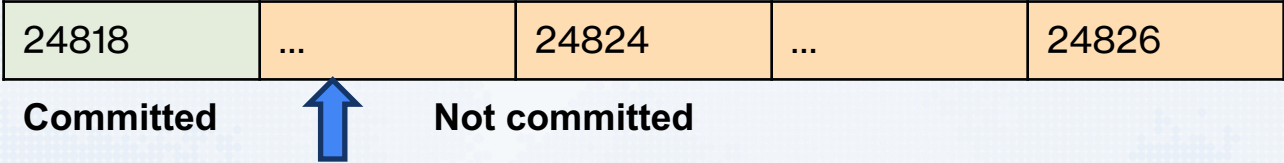
- Transform normal PostgreSQL queries to time traveled queries
- Read the *visible* version at any given point in time
 - Only see the committed versions

Visibility Rule

- A version is visible at a given timestamp T if:
 - The `begin_xid` is a transaction **committed** before T
 - And the `end_xid` is **not committed** before T

Visibility Rule

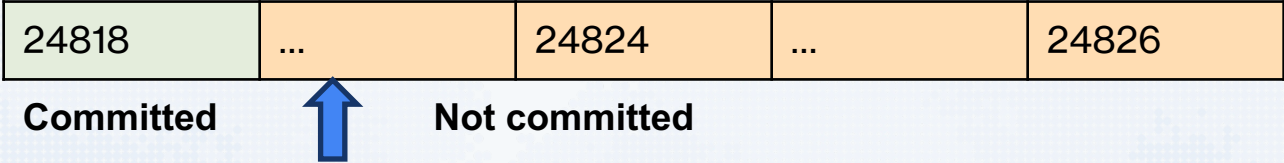
```
SELECT product, inventory FROM products;
```



product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Visibility Rule

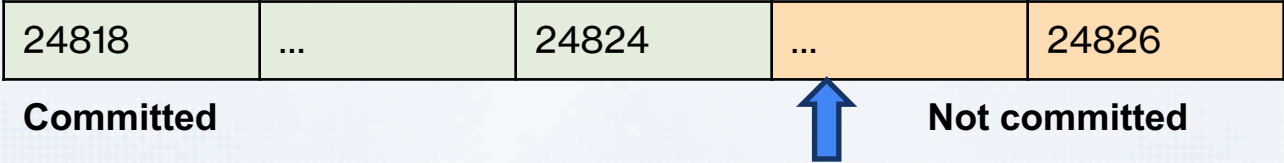
```
SELECT product, inventory FROM products;
```



product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Visibility Rule

```
SELECT product, inventory FROM products;
```



product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Visibility Rule

```
SELECT product, inventory FROM products;
```

24818	...	24824	...	24826
-------	-----	-------	-----	-------

Committed



product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Visibility Rule

```
SELECT product, inventory FROM products;
```

24818	...	24824	...	24826
-------	-----	-------	-----	-------

Committed



product	inventory	begin_xid	end_xid
Premium Quality Widget	100	24818	24824
Premium Quality Widget	99	24824	24826
Premium Quality Widget	98	24826	∞

Which Transactions Have Committed?

- Use the PostgreSQL's snapshot info: `pg_current_snapshot()`
 - `xmin`: transaction IDs $< xmin$ are committed
 - `xmax`: transaction IDs $\geq xmax$ are not finished
 - `xip_list`: between `xmin` and `xmax` but not committed

Query Transformation


- Append two predicates to a query:

```
(begin_xid < xmax AND NOT begin_xid = ANY(xip_list))  
AND (end_xid >= xmax OR end_xid = ANY(xip_list))
```

- First, select versions *added* by committed transactions
- Second, select versions *not deleted* by committed transactions

Example

```
SELECT product, inventory FROM products;
```

24818	...	24824	...	24826
Committed				Not committed

```
SELECT product, inventory FROM products  
WHERE (begin_xid < 24825) AND (end_xid >= 24825)
```


Implementation

- Implement a Postgres proxy, using `libpg_query` to parse and transform queries
 - Postgres wire-compatible
 - Kudos to my amazing teammate Harry Pierson (@DevHawk)
- DBOS keeps track of the timestamp to snapshot mapping

Time Travel Debugging

Replay Statements As Of a Past Transaction

BEGIN

```
UPDATE products SET inventory = 50 WHERE product LIKE 'Premium%'  
RETURNING product, inventory;
```

```
UPDATE products SET inventory = 100 WHERE product = 'Premium Widget'  
RETURNING product, inventory;
```

COMMIT

Challenges

- Can't modify history data
- Require read-your-own-writes within a transaction
- WAL doesn't track SQL statements
 - Each statement may change **multiple** records
 - Each record may be changed **multiple** times

Main Idea

- Proxy transforms insert/delete/update to **select** queries
- Keep track of **which statement** within the transaction made **what** changes
 - `begin_seq`: The statement ID that added the record
 - `end_seq`: The statement ID that deleted the record
- Use PG triggers to record how many records are changed **per statement**
 - Emit WAL messages

Local Debug

VARIABLES

- Local: subtractInventory
 - ctxt = Transaction...
 - numAffected = unde...
 - this = null
- Block

WATCH

CALL STACK

- Time ... RUNNING
- PAUSED ON BR...
- subtractInventory
- subtractInventory
- wrappedTransaction

```

TS operations.ts
TS utilities.ts x
JS debug_workflows

src > TS utilities.ts > ShopUtilities > subtractInventory > numAffected

40 const reportSes = (process.env['REPORT_EMAIL_TO_ADDRESS'] &&
41   : undefined;
42
43
44 export class ShopUtilities {
45   @Transaction()
46   static async subtractInventory(ctxt: KnexTransactionContext):
47     Promise<void> {
48     const numAffected = await ctxt.client<Product>('products').
49       where('product_id', PRODUCT_ID).andWhere('inventory', '>=',
50       1)
51     .update({
52       inventory: ctxt.client.raw('inventory - ?', 1)
53     });
54   if (numAffected <= 0) {

```

PROBLEMS 18 OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

TERMINAL

```

2024-09-30 20:33:14 [info]: Workflow executor initialized

```

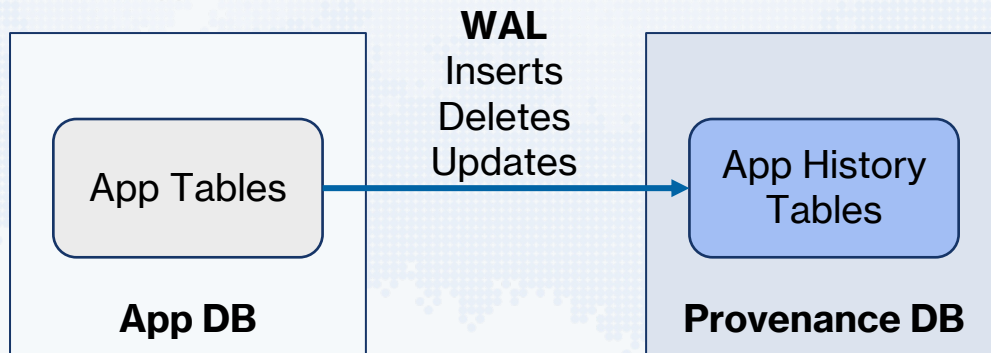
Try DBOS Time Travel

- Tutorial: <https://docs.dbos.dev/cloud-tutorials/interactive-timetravel>
- Case Study: <https://www.dbos.dev/blog/database-time-travel>



Summary

- Export history data to a **separate provenance DB**
- Leverage logical replication + multi-versioning
 - No impact on the app DB
 - Work with off-the-shelf/managed Postgres servers
 - Bonus: Enable transaction debugging



Chat with Us!

- Export history data to a **separate provenance DB**
- Leverage logical replication + multi-versioning
 - No impact on the app DB
 - Work with off-the-shelf/managed Postgres servers
 - Bonus: Enable transaction debugging



Qian Li
Co-founder



Peter Kraft
Co-founder

