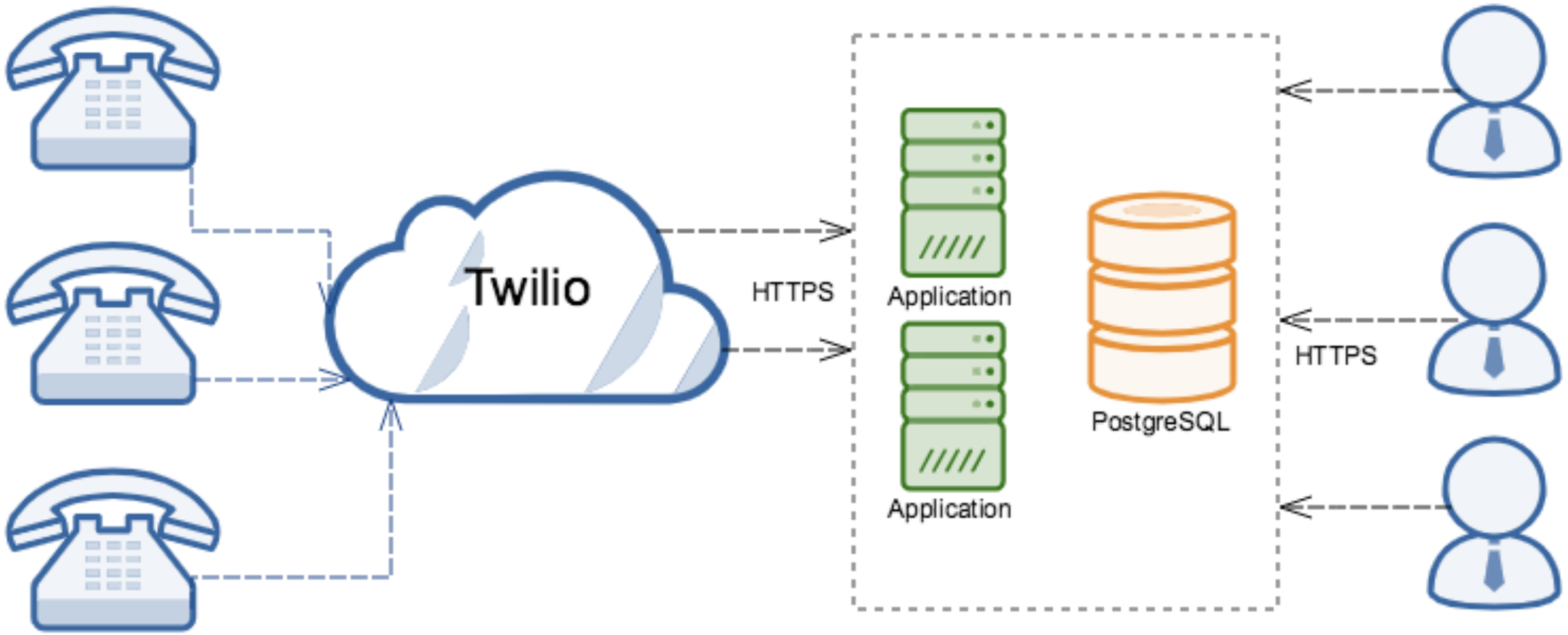
Temporal data & time travel in PostgreSQL

Alexei Mikhailov, Glia

Outline

- **How we usually investigate issues in production**
- **Temporal tables**
- **PostgreSQL extension temporal_tables**
- **Problems**
- **Alternative solutions**

Call center



Clients complaints

- Why was this call rejected?
- Why was this call picked up by operator X instead of Y?
- Why operator X had significantly less calls than others on day Y?

Logs

Logs

13:48:41.219	info	Operator updated	a925e5a5-52bd-43a4-8009-55fdf65b6a24	temporarily-unavailable
13:48:51.542	info	Operator removed	a925e5a5-52bd-43a4-8009-55fdf65b6a24	temporarily-unavailable
13:48:59.280	info	Operator added	b29beeaf-2863-4d27-984b-b6b8b3ac5465	available
13:48:59.298	info	Queue availability statuses are updated	-	-
13:49:12.172	info	Operator updated	b29beeaf-2863-4d27-984b-b6b8b3ac5465	temporarily-unavailable

Logs

- **Incomplete**
  `> 2019-03-10 11:11 Operator is updated`
  What operator? What was updated?

- **Personally identifiable information (PII)**

- **Reactive**
  Didn't have logs back then? GL HF

- **Hard to use**
  `> 2018-03-`**`10`**` 23:00 Operator is ready, id=007`
  `> 2018-03-`**`12`**` 01:00 Operator is offline, id=007`

  *Search logs for date 2018-11-**11**: no results for agent 007*

- **Not durable** *

Temporal tables

- **Debugging**
- **Audit and security**
- **Undo functionality**
- **BI**

Temporal tables

- Big new feature of SQL 2011
  https://cs.ulb.ac.be/public/_media/teaching/infoh415/tempfeaturessql2011.pdf

- Introduces new syntax
  `... FROM table FOR SYSTEM_TIME AS OF '<timestamp>'`

$$$	**Oracle**	Yes	Since 10g+ (2005)
$$$	**IBM DB2**	Yes	Since version 10 (2010)
$$$	**MS SQL Server**	Yes	Since SQL Server 2016
	PostgreSQL	No	3rd party extension temporal_tables
	MySQL	No	

Temporal tables

- Tables have additional time-period (interval) information
- Time period is closed-open (inclusive from the left, exclusive from the right)
    - PostgreSQL: **`tstzrange('[2018-01-01, 2018-12-01)')`**
- **System time-period**: what was the ACID state of my data on <time>?
    - Maintained by system
    - **UPDATE** / **DELETE** / **INSERT** automatically maintain this info
    - Only supports history, must never be future
    - Used by **`...`**`FROM table FOR SYSTEM_TIME AS OF`
    - **Usually separate table**

Temporal tables

- **Business time-period**
    - Maintained by application / user
    - Represent business reality, e.g. "This contract will have price X for next 3 months"

```
contract_id | price |       valid_period
------------+-------+--------------------------
     1      |  50   | [2018-01-01, 2018-02-01)
     1      |  100  | [2018-02-01, 2019-02-01)
```

    - Can be in the future
    - No special syntax for SELECT
- **Bi-temporal tables** = tables with system period and business period

Historical table

```
CREATE TABLE agents (
  id integer NOT NULL,
  status character varying NOT NULL,
  system_period tstzrange NOT NULL
);

CREATE TABLE agents_history (LIKE agents);
```

Historical table

```
CREATE FUNCTION save_previous_version() RETURNS TRIGGER AS $body$
BEGIN
  IF (TG_OP = 'INSERT') THEN
    NEW.system_period := tstzrange(current_timestamp, NULL, '[)');
    RETURN NEW;
  ELSIF (TG_OP = 'UPDATE') THEN
    NEW.system_period := tstzrange(current_timestamp, NULL, '[)');
    INSERT INTO agents_history (id, status, system_period) VALUES
(OLD.id, OLD.status, tstzrange(lower(OLD.system_period),
current_timestamp, '[)'));
    RETURN NEW;
  ELSE
    INSERT INTO agents_history (id, status, system_period) VALUES
(OLD.id, OLD.status, tstzrange(lower(OLD.system_period),
current_timestamp, '[)'));
    RETURN OLD;
  END IF;
END;
$body$
LANGUAGE plpgsql;

CREATE TRIGGER history_trigger
BEFORE INSERT OR UPDATE OR DELETE ON agents
FOR EACH ROW EXECUTE PROCEDURE save_previous_version();
```

Historical table

```
INSERT INTO agents (id, status) VALUES (1, 'ready);
```

"agents"

```
 id | status | system_period
----+--------+-----------------
  1 | ready  | ["18:59:17",)
(1 row)
```

"agents_history"

```
 id | status | system_period
----+--------+-----------------
(0 rows)
```

History table

```
UPDATE agents SET status = 'away' WHERE id = 1;
```

"agents"

```
 id | status   | system_period
----+----------+-----------------
  1 |  away    | ["19:10:47",)
(1 row)
```

"agents_history"

```
 id | status |      system_period
----+--------+--------------------------
  1 | ready  | ["18:59:17","19:10:47")
(1 row)
```

History table

```
UPDATE agents SET status = 'offline' WHERE id = 1;
```

"agents"

id	status	system_period
1	offline	["19:15:06",)

(1 row)

"agents_history"

id	status	system_period
1	ready	["18:59:17","19:10:47")
1	away	["19:10:47","19:15:06")

(2 rows)

History table

```
DELETE FROM agents WHERE id = 1;
```

"agents"

id	status	priority	valid_from	valid_to

(0 rows)

"agents_history"

id	status	system_period
1	ready	["18:59:17","19:10:47")
1	away	["19:10:47","19:15:06")
1	offline	["19:15:06","19:17:47")

(3 rows)

Queries

```
CREATE VIEW agents_with_history AS
SELECT * FROM agents
UNION ALL
SELECT * FROM agents_history
```

Queries

How did the data look like at point in time T?

```
SELECT * FROM agents_with_history
WHERE id = 1
  AND '19:12'::timestamptz <@ system_period

 id | status  |     system_period
----+---------+-----------------------
  1 |  away   | ["19:10:47","19:15:06")
(1 row)

SELECT * FROM agents_with_history
WHERE id = 1
  AND '19:09'::timestamptz <@ system_period

 id | status |     system_period
----+--------+-----------------------
  1 | ready  | ["18:59:17","19:10:47")
(1 row)
```

Queries

How did the data change in interval (T1, T2)

```
SELECT * FROM agents_with_history
WHERE id = 1
  AND tstzrange('[18:55,19:12]') <@ system_period

 id | status |     system_period
----+--------+-----------------------
  1 | ready  | ["18:59:17","19:10:47")
  1 | away   | ["19:10:47","19:15:06")
(2 rows)

SELECT * FROM agents_with_history
WHERE id = 1
  AND tstzrange('[19:05,)') <@ system_period

 id | status  |     system_period
----+---------+-----------------------
  1 | away    | ["19:10:47","19:15:06")
  1 | offline | ["19:15:06","19:15:47")
(2 rows)
```

Extension

- https://github.com/arkhipov/temporal_tables
- **pgxnclient install temporal_tables**
- **CREATE EXTENSION temporal_tables**

  Requires superuser

- ```
  CREATE TRIGGER versioning_trigger
  BEFORE INSERT OR UPDATE OR DELETE ON agents
  FOR EACH ROW EXECUTE PROCEDURE
  versioning ('system_period', 'agents_history', true);
  ```

- **Doesn't require same set of columns**
  ```

Advantages

- **ACID**
- **Backups**
- **Easy to integrate with existing database**
- **Low performance overhead**

Data duplication

- **Normalization**
  Move heavy columns out of the main table

  ***Before**: users (id, name, picture)*
  ***After**: users(id, name, picture_id), pictures(id, data)*

- **Remove heavy columns from historical table**
  Data will be lost

- **Prune historical tables**

- **Separate tablespace on cheap HD**

Caveats

- **Structure is not in sync with main table**
    - Can use inheritance, but with caution
    - Event triggers (i.e. *do something when schema changes*)
- **No primary key, no foreign keys**
- **You MUST use tstzrange for system_period column**
  If you're using temporal_tables extension. Only "gist" index is supported.
- **"Custom" extension**

Alternative solutions

- **RDBMS with support for temporal tables**
  Oracle, IBM DB2, MS SQL

- **Event sourcing**

- **Immutable data**

```
| agents |    | statuses   |
+--------+    +------------+
| id     |<---| agent_id   |
| name   |    | status     |
                | timestamp  |
```

- **Application-managed audit logs**

Demo

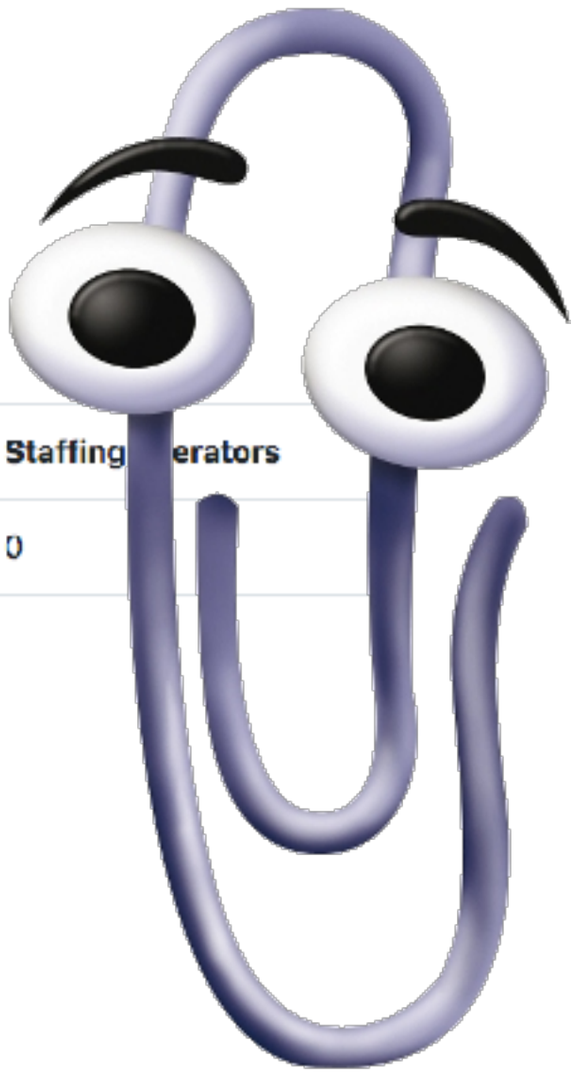
Site ID(s) cc0a1ccc-e00a-4b28-9ea5-277548995fcc ×

Queue ID(s)

Point in time dd/mm/yyyy, --:--:-- (GMT-06:00) Central Time (US

Show

Queue Name	ID	Status	Enqueued visitors	Staffing erators
General	8788b207-edf8-4a66-a27f-ccd73b565e60	unstaffed	0	0



Demo

Site ID(s) cc0a1ccc-e00a-4b28-9ca5-277548995fec ×

Queue ID(s)

Point in time dd/mm/yyyy, --:--:-- (GMT-06:00) Central Time (US

Show

Queue Name	ID	Status	Enqueued visitors	Staffing operators
General	8788b207-edf8-4a66-a27f-ccd73b565e60	unstaffed	0	0

Recap

- Logs are not great
- Temporal tables are possible in any DB
- Use with caution
- Know alternatives

Questions?

Slides:
https://github.com/take-five/temporal-tables-presentation