TimeTraveling in PostgreSQL

Varlena, LL

A. Elein Mustain
www.varlena.com
elein@varlena.com

Time Flies Like an Arrow…

Fruit Flies Like a Banana.

Did you ever want to know…

- How many widgets were in your inventory yesterday at 4:05pm?
- How long it took to sell 500 thingies?
- When is the best time to order more thingies?

No Cheap Fares for TimeTravel

- Row space over head of two timestamp columns
- Query qualification on time for every select
- Small update overhead

Old Postgres TimeTravel

- Postgres name for no-overwrite storage with no vacuum.
- Min and max timestamps stored per row.
- Enabled selection from at any point in time until next vacuum.

New-Fangled PostgreSQL TimeTravel

- Does not rely on no-overwrite storage.
- Start and End timestamps stored per row.
- Enabled selection from at any point in time.
- Only INSERTS, no DELETES or UPDATES.
  - Deletes and Updates close time period for row

What do we want to see?

- Current selection

```
SELECT item_name, in_stock
FROM current_inventory
WHERE item_id = 17;
```

- TimeTravel selection

```
SELECT item_name, in_stock
FROM inventory_at_time('03/18/06 1:00pm')
WHERE item_id = 17;
```

TimeTravel Parts

- Each Table Requires
  - Start and End Time columns, Indexes
  - View on Current Data
  - At Time Functions
  - Delete Rule
  - Update Trigger
  - Optional Insert Trigger
- Code for all items is the same except for column names.

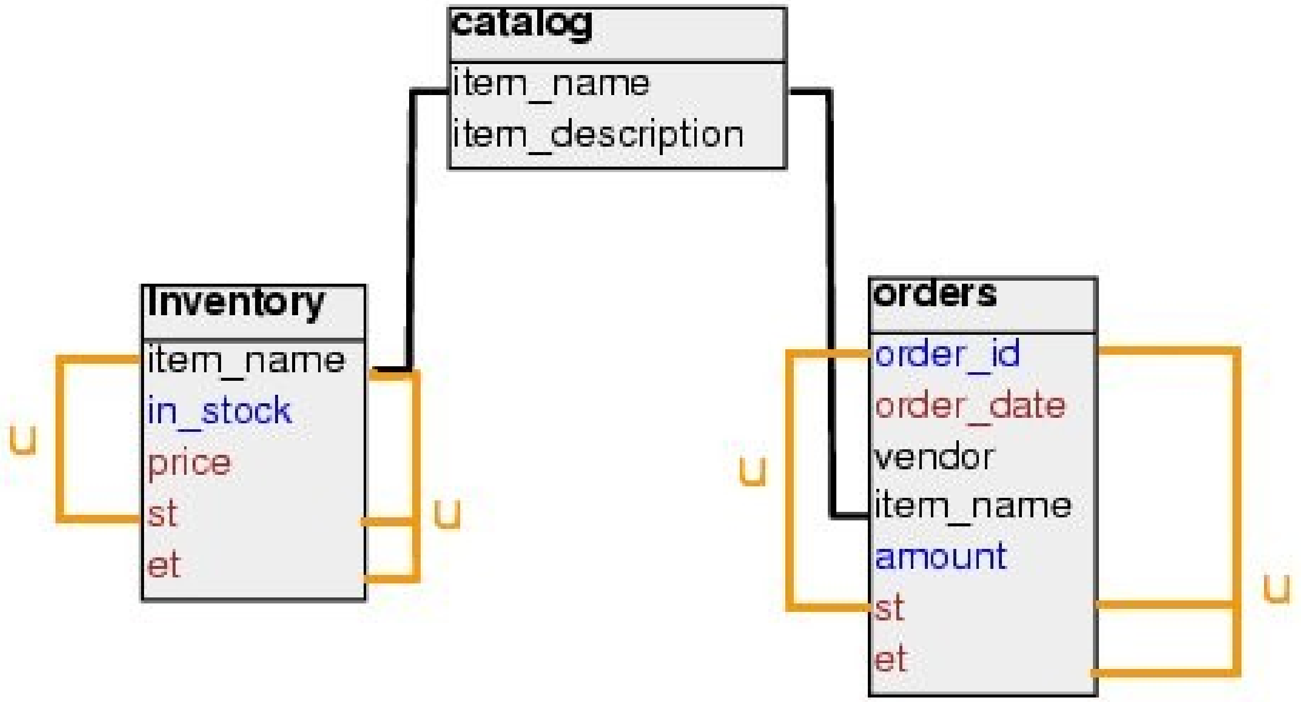
TimeTravel Tables

- Define Table
  - having unique key
  - with start and end time columns
  - For Insert:
    - Default start to current_timestamp

TimeTravel Tables

- Define Indexes
  - unique indexes across key and start and end times
  - unique index on key where end time is null
- Define new table or use ALTER TABLE

Example Tables



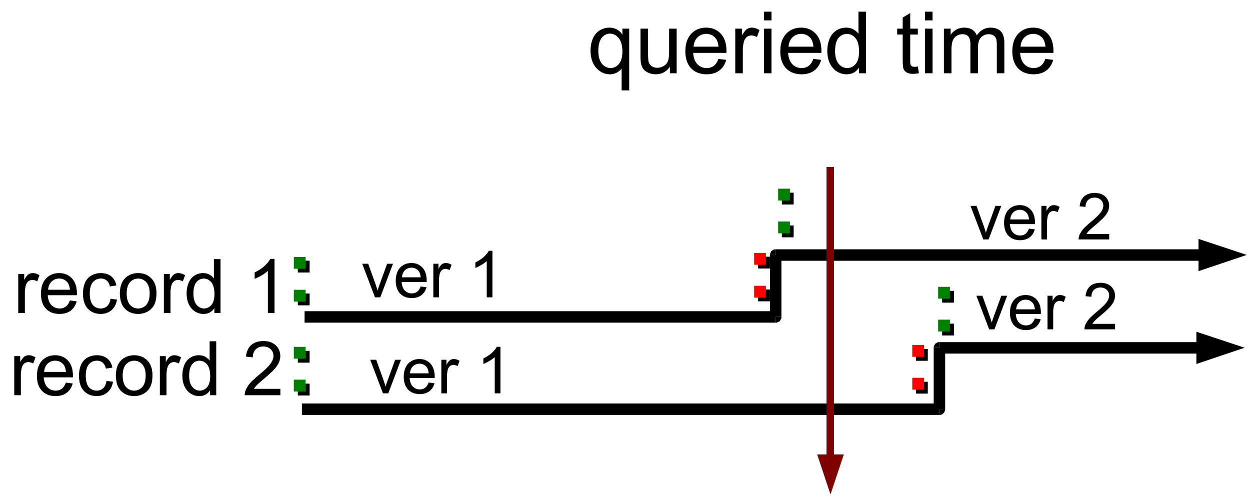
Current View

- Select Rows where end time is null.

```
CREATE VIEW current_inventory AS
SELECT item_name, in_stock, price
FROM inventory
WHERE et IS NULL;
```

At Time Function

- Select current rows at a time and
- Select rows active between start and end time.



At Time Function

```
CREATE OR REPLACE FUNCTION
inventory_at_time(timestamptz)
RETURNS SETOF current_inventory AS
$$
    SELECT item_name, in_stock, price
    FROM inventory
    WHERE (SELECT CASE WHEN et IS NULL
                THEN (st <= $1)
                ELSE (st <= $1 AND et > $1)
                END) ;
$$ LANGUAGE 'SQL';
```

Delete Rule

- Rows are never deleted. Set end time instead.
- Delete Rule
  - Sets end time and passes row to UPDATE TRIGGER

Delete Rule

```
CREATE RULE inv_del
    AS ON DELETE TO inventory
    DO INSTEAD
       UPDATE inventory
          SET et=current_timestamp
       WHERE item_name = OLD.item_name
          AND et IS NULL;
```

Update Trigger

- Update Trigger
  - Disallow updates of old rows (end time is not null)
    - Quietly for DELETE on id to work
  - If NEW end time is present, Perform UPDATE only
  - Otherwise, INSERT into table OLD row into table with end time and allow UPDATE

Update Trigger Function

```
CREATE OR REPLACE FUNCTION upd_inventory()
RETURNS TRIGGER AS
$$
   BEGIN
      ...
   END;
$$ LANGUAGE 'plpgsql';
```

Update Trigger Function

```
IF OLD.et IS NOT NULL THEN
   RETURN NULL; -- quietly disallow
END IF;
IF NEW.et IS NULL THEN
  INSERT INTO inventory VALUES
    (OLD.item_name, OLD.in_stock,
     OLD.price, OLD.st, current_timestamp);
  NEW.st = current_timestamp;
END IF;
RETURN NEW;
```

Update Trigger

```
CREATE TRIGGER upd_inventory
BEFORE UPDATE ON inventory
FOR EACH ROW
EXECUTE PROCEDURE upd_inventory();
```

Insert Trigger

- Insert defaults start time.
- Leaving Insert open allows the inserts to set the start time and end time.
  - Helpful for loading old data
  - Good for trusted applications.
- Aggressive Insert Trigger
  - Set start time to current_timestamp
  - Set end time to NULL

Insert Trigger Function

```
CREATE OR REPLACE FUNCTION
ins_inventory
RETURNS TRIGGER AS
$$
   NEW.st := now();
   NEW.et := NULL;
   RETURN NEW;
$$ LANGUAGE 'SQL';

CREATE TRIGGER ins_inventory
BEFORE INSERT ON inventory
FOR EACH ROW
EXECUTE PROCEDURE ins_inventory();
```

TimeTravel Parts

- Each Table Requires
  - Start and End Time columns
  - View on Current Data
  - At Time Functions
  - Delete Rule
  - Update Trigger
  - Optional Insert Trigger
- Code for all items is the same except for column names.

Application Functions

- Sales Function
  - Parameterized Query
  - Decrements Inventory
- Receive Order Function
  - Closes Order
  - Updates Inventory

Sales

```
CREATE FUNCTION sale(text, integer)
RETURNS VOID AS
$$
  UPDATE inventory
  SET in_stock = in_stock - $2
  WHERE item_name = $1;
$$ LANGUAGE 'SQL';
```

Receiving Orders

- Validate Order
- Close order
- Upsert Inventory

Receiving Orders

```
CREATE OR REPLACE FUNCTION
   receive_order(r_order_id integer)
RETURNS integer AS
$$
DECLARE
rowcount integer;
orec RECORD;
BEGIN
   ...
END;
$$ LANGUAGE 'plpgsql';
```

Receiving Orders

```
SELECT INTO orec
   order_id, item_name, amount
FROM orders_current o
WHERE o.order_id = r_order_id;
IF NOT FOUND THEN
   RAISE EXCEPTION
    'Cannot Receive Order % ',r_order_id;
ELSE
   DELETE FROM orders
   WHERE o.order_id = r_order_id;
END IF;
```

Receiving Orders

```
LOOP
  UPDATE inventory ...
  IF FOUND THEN
     RETURN
  ELSE
     BEGIN
        INSERT INTO inventory ...
        RETURN;
     EXCEPTION WHEN unique_violation THEN
        -- do nothing: loop around
     END;
  END IF;
END LOOP;
```

Receiving Orders

```
LOOP
  UPDATE inventory
     SET in_stock = in_stock + orec.amount
  WHERE inventory.item_name = orec.item_name;
  IF FOUND THEN
     RETURN r_order_id;
  ELSE
     BEGIN
        INSERT INTO inventory VALUES
          (orec.item_name, orec.amount, NULL);
        RETURN;
     EXCEPTION WHEN unique_violation THEN
        -- do nothing: loop around
     END;
  END IF;
END LOOP;
```

Let's Try it Out

- Showing TimeTravel live...

Catalog

```
    item_name
------------------
 widgets
 thingies
 whatchamacallits
 thatstuff
 thisstuff
(5 rows)
```

Inventory

```
    item_name       |   |st                   | et
--------------------+---+---------------------+----
 widgets            |30 | 17:50:50.602
 thingies           |25 | 17:50:51.143
 whatchamacallits   |50 | 17:50:51.193
 thatstuff          |40 | 17:50:51.253
 thisstuff          |60 | 17:50:51.333
(5 rows)
```

Orders

```
order_id|item_name          |amount|st
--------+-------------------+------+--------------
1       |widgets            |100   |05:50:51.403
2       |thingies           |200   |05:50:51.643
3       |whatchamacallits   | 25   |05:50:51.754
4       |thatstuff          | 50   |05:50:51.814
5       |thisstuff          | 75   |05:50:51.874
(5 rows)
```

Update Orders

```
update orders
set amount = amount + 20
where order_id = 3;
```

Update Orders Results

orders:

```
id|item_name         |amt|st           |et
--+------------------+---+-------------+-------------
 3|whatchamacallits|25 |05:50:51.754|06:22:56.421
 3|whatchamacallits|45 |06:22:56.421|
```

current_orders:

```
id|item_name         |amt
--+------------------+---
3 |whatchamacallits|45
```

Receive Order

```
select receive_order(1);

-- close order #1
-- update inventory per order #1
```

Receive Orders Results

```
orders where order_id = 1:

id|item_name          |amt|st          |et
--+-------------------+---+------------+------------
 1|widgets            |100|05:50:51.403|06:40:20.563


current_orders where order_id = 1:

id|item_name          |amt
--+-------------------+---
```

Receive Inventory Results

inventory where item_name = widgets:

```
item_name|in_stock|st              |et
--+------------------+--------------+-------------
widgets  |30       |17:50:50.602|18:40:20.563
widgets  |130      |18:40:20.563|
```

current_inventory where item_name = widgets:

```
item_name|in_stock
---------+----------
widgets  |130
```

Time Travel

```
--After initialization
--Before receiving order #1

tt=# select * from inventory_at_time
  ( '2006-06-29 18:00' );
    item_name        | in_stock | price
--------------------+-----------+-------
 thingies            |        25 |
 whatchamacallits    |        50 |
 thatstuff           |        40 |
 thisstuff           |        60 |
 widgets             |        30 |
(5 rows)
```

Time Travel JOIN

```
--After initialization
--After receiving order #1

SELECT i.item_name, i.in_stock,
  sum(o.amount) AS on_order, '19:00' AS time
FROM inventory_at_time('2006-06-29 19:00') i
JOIN
   orders_at_time('2006-06-29 19:00') o
ON (i.item_name = o.item_name)
GROUP BY i.item_name, i.in_stock, time
ORDER BY i.item_name;
```

Time Travel JOIN

```
   item_name        | in_stock | on_order | time
--------------------+----------+----------+-------
 thatstuff          |       40 |       50 | 19:00
 thingies           |       25 |      200 | 19:00
 thisstuff          |       60 |       75 | 19:00
 whatchamacallits   |       50 |       45 | 19:00
 widgets            |      130 |          | 19:00
```

Time Flies Like an Arrow...

Fruit Flies Like a Banana.