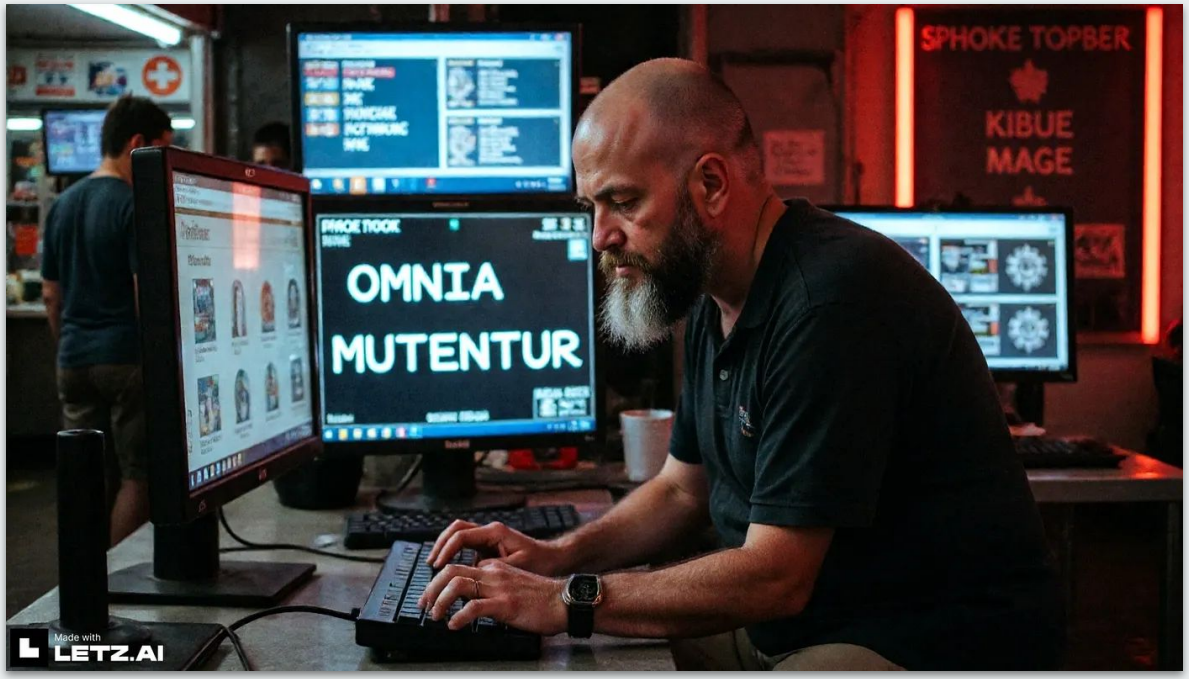
# Slowly Changing Dimensions in Postgres




Marc Linster
PGConf.EU 2025

dbtune

Digital Learning Hub

# What are SCDs and why do we care about them?

- Reference data changes over time
  - Price lists
  - Bill of Material
  - Employee hierarchy
  - Project assignments
  - …
- Multiple ways to track reference data changes over time
  - When was the change effective?
  - From when to when how long was it valid?
  - What were the prior values?
  - What was the history, …?
- SCDs introduced by Ralph Kimball
- Pros and cons - depending on desired outcome
- Powerful tools in Postgres to keep it simple

# Six Types of SCDs

Type 1: Corrections in the data - no tracking of when the change happened (except in the log) or how long it was valid

Type 2: A new row for every change, with start date and end date.

Type 3: Track date of last change and prior value

Type 4: A new row for every change, with validity date

Type 5: (1+4): Current table + history table

Type 6: 1+2+3 combined into a single table with start and end dates and a flag to indicate which record represents the current value.

# Type 2

- Each change is assigned a new row in the data table, and the primary key is expanded with start date and end date columns.
- NULL can be used to indicate that no end date has been set; however, in that case, the column cannot be part of the primary key definition, as SQL does not allow the NULL value in a primary key column.

| product_id | price | start_date | end_date |
|---|---|---|---|
| 12345 | 19.99 | 2025-01-01 | 2025-01-31 |
| 12345 | 20.99 | 2025-02-01 | NULL |

# Type 3

- Extra columns to record the previous value and the date of the last change. This only enables tracking of a single prior version.

| product_id | price | last_update | prior_price |
|---|---|---|---|
| 12345 | 20.99 | 2025-01-01 | 19.99 |

# Type 4

- The current price and the historical prices are kept in the same table. The current price is the price with the most current effective date.

| product_id | price | effective_date |
|---|---|---|
| 12345 | 19.99 | 2025-01-01 |
| 12345 | 20.99 | 2025-02-01 |

# Type 5

- Combines Type 1 and Type 4
- Includes both the current data and a history table.

| product_id | price |
|---|---|
| 12345 | 20.99 |

| product_id | price | effective_date |
|---|---|---|
| 12345 | 19.99 | 2025-01-01 |
| 12345 | 20.99 | 2025-02-01 |

# Type 6

- Merges Types 1, 2, and 3 into a single table
- Includes start and end dates for changes and a flag to indicate which record shows the current value.

| product_id | price | start_date | end_date | current |
|---|---|---|---|---|
| 12345 | 19.99 | 2025-01-01 | 2025-01-31 | false |
| 12345 | 20.99 | 2025-02-01 | NULL | true |

# Pros and Cons of the Different Forms of SCDs

| SCD Type | Pro | Con |
|---|---|---|
| Type 1 | Simple | No historical data |
| Type 2 | Simple | Lookups can become expensive if the dimension sees a lot of changes.<br>Start date and end date consistency can be challenging to manage. |
| Type 3 | Simple | Not an effective way to track historical data |
| Type 4 | Relatively simple | Lookups can become expensive if the dimension sees a lot of changes.<br>It can be challenging to answer questions like "What was the price on Feb 2, 2025?" |
| Type 5 | Addresses some of the problems of Type 4. Very fast access to current value. | Inserts and updates cause transactions in two tables.<br>Transactions referring to historical data are problematic!!! |
| Type 6 | Fast lookups for current data. | Start date and end date consistency can be challenging (Except in PostgreSQL).<br>The table can become large if data changes frequently. |

# Perceived Problems with Type 6 SCDs

Developers, *especially those who don't know PostgreSQL very well*, are often concerned about:

1. Defining ranges
2. Managing the boundaries between ranges
3. Finding out what the value was on a given day
4. Avoiding overlaps between ranges

# Postgres to the Rescue

## 1. How to define ranges

⇒ Postgres DATERANGE

No need to have start & end columns!

Simple queries using '@>' for inclusion

## 2. Managing boundaries

⇒ [,],(,) define boundaries cleanly

⇒ e.g.: [2025-06-01, 2025-07-01)

No need to have custom inclusion/exclusion logic

## 3. Finding out what the value was on a given day

```
SELECT price FROM product price scd
 WHERE validity @>'2025-06-18'::date
 AND product_id = 12345;
```

Easy peasy!

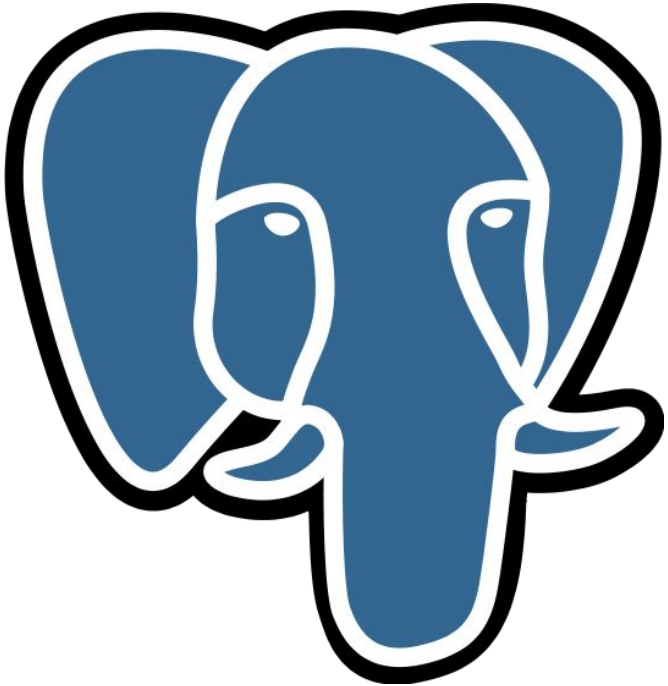
## 4. Avoiding overlapping ranges

```
CREATE TABLE product_price_scd (
 product_id INTEGER NOT NULL,
 price NUMERIC DEFAULT 0 NOT NULL,
 validity DATERANGE,
 current BOOLEAN,
 EXCLUDE USING GIST (product_id WITH =, validity WITH &&)
);
```

# Conclusion

- Postgres simplifies the solution enormously with
  - Innovative data types: DATERANGE
  - Extension: BTREE_GIST
    - Combination of two index types
    - BTREE (for equality)
    - GiST (for date ranges, geometry, etc.)
    - Great for multi-column queries and exclusion constraints

- No custom code to deal with start/end date, overlapping ranges, or to find the value for a give day!
- Accompanying blog: https://marclinster.medium.com/



Use Postgres - Get Stuff Done!