



# 10 Things I Learned Solving Advent of Code Puzzles with PostgreSQL

Ryan Booz

Postgres Conference Silicon Valley 2023

 redgate | About me

# Ryan Booz

PostgreSQL & DevOps  
Advocate



[@ryanbooz](https://twitter.com/@ryanbooz)



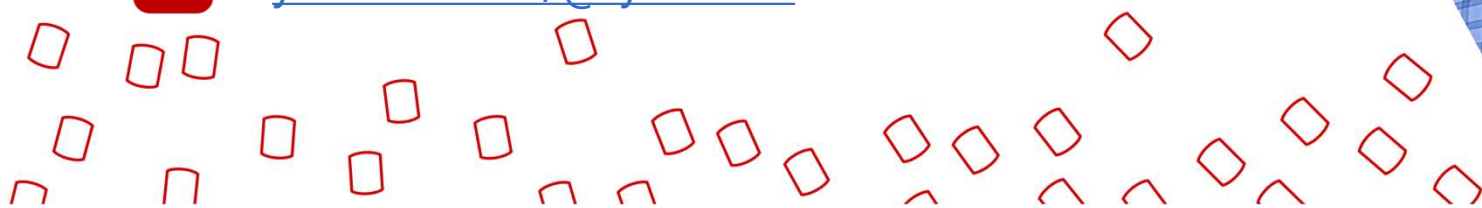
[/in/ryanbooz](https://in.linkedin.com/in/ryanbooz)



[www.softwareandbooz.com](http://www.softwareandbooz.com)



[youtube.com/@ryanbooz](https://youtube.com/@ryanbooz)



[github.com/ryanbooz/presentations](https://github.com/ryanbooz/presentations)



# Agenda

**01** ETL vs ELT

**02** Loading Data

**03-10** 7 SQL/PostgreSQL Features

**Bonus** Community



# WORDLE

**A DAILY WORD GAME**

A few functions are  
only included with  
PostgreSQL  $\geq 14$



01/10

# ETL vs ELT

# ETL vs ELT

## Extract, Transform, Load

- External processing of non-relational data to create relational data
- Not SQL focused

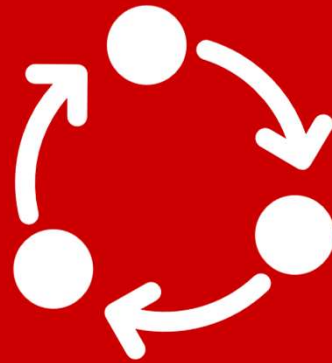
## Extract, Load, Transform

- Internal processing of non-relational data to create relational data
- SQL focused

Convert non-relational data  
into relational, tabular data.

# Why Has ETL Been So Popular?

- External tools could more quickly bring specialized functionality to data processing
- Databases didn't speak web languages well
  - ie. XML or JSON
- Specialized tools = specialized jobs



Iteration is slow

**Keep processing close to  
the data for faster iteration**

# ELT in PostgreSQL

- Retain transactional consistency and control
- PostgreSQL has a plethora of functions for processing and transforming data
  - Regex
  - JSON
  - String
- Array and JSON output are particularly useful for processing

02/10

# Inserting Data



# Inserting Data

- Quickly dump data to tables and keep the schema simple
- Post-process JSON, XML, strings, arrays, etc.
- Use **COPY**:
  - most supported method of getting data in quickly
  - CSV or custom delimiters
- Use code:
  - work in batches of rows to reduce transaction overhead

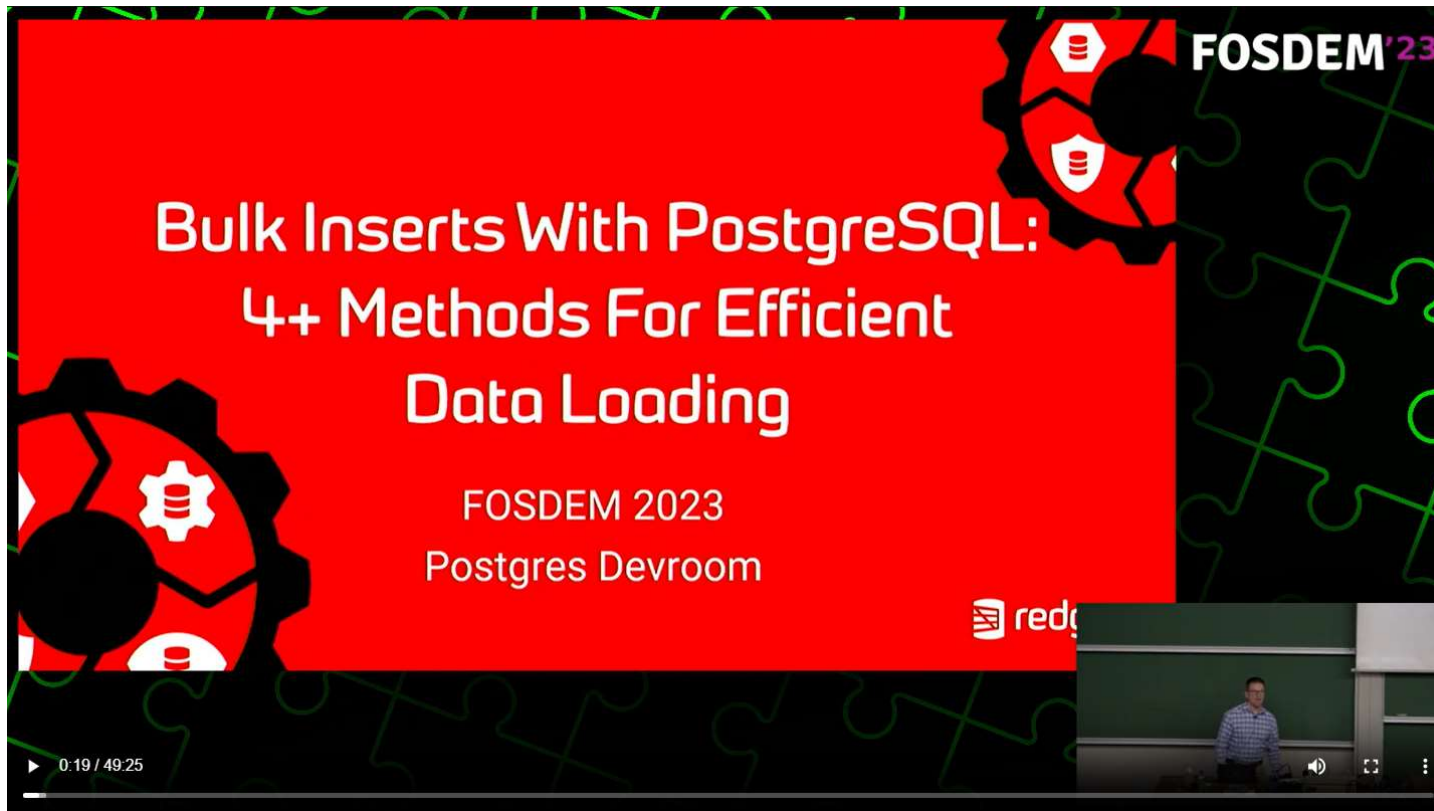
# COPY vs \copy

- COPY is a PostgreSQL command, not SQL standard
- COPY requires files local to the server
- My examples primarily use psql \copy command
- This streams data from local files to PostgreSQL

STDIN COPY

# COPY Caution

- Requires correct column order, matching data types, and clean data (no conversion)
- Options like [pgloader](#) overcome some limitations
  - pre-checks on certain columns of data



<https://bit.ly/postgres-bulk-load-fosdem23>



# Data Import Rules



- 1 Create a generated ID for ordering later if needed
- 2 Add a timestamp column if it's time-series data
- 3 Pre-processes what makes sense, but don't go overboard

# K.I.S.S. – Advent of Code

```
create table dec05 (  
    id integer generated by default as identity,  
    puzzle_input text  
);  
  
-- COPY the text into the appropriate columns  
\COPY dec05 (puzzle_input) FROM input_05.txt NULL '';
```



# K.I.S.S. – Wordle

```
CREATE TABLE tweets_raw(  
    ts timestampz NOT NULL,  
    tweet_id bigint NOT NULL,  
    tweet_raw JSONB NOT null,  
);
```



# K.I.S.S. – Wordle

```
CREATE TABLE wordle_tweet (  
  ts timestampz NOT NULL,  
  created_at timestampz NOT NULL,  
  author_id bigint NOT NULL,  
  author_handle TEXT NOT NULL,  
  author_verified bool,  
  author_location TEXT,  
  tweet_id bigint NOT NULL,  
  tweet TEXT NOT null,  
  game int NULL,  
  guess_total int null  
);
```

03/10

# Common Table Expression

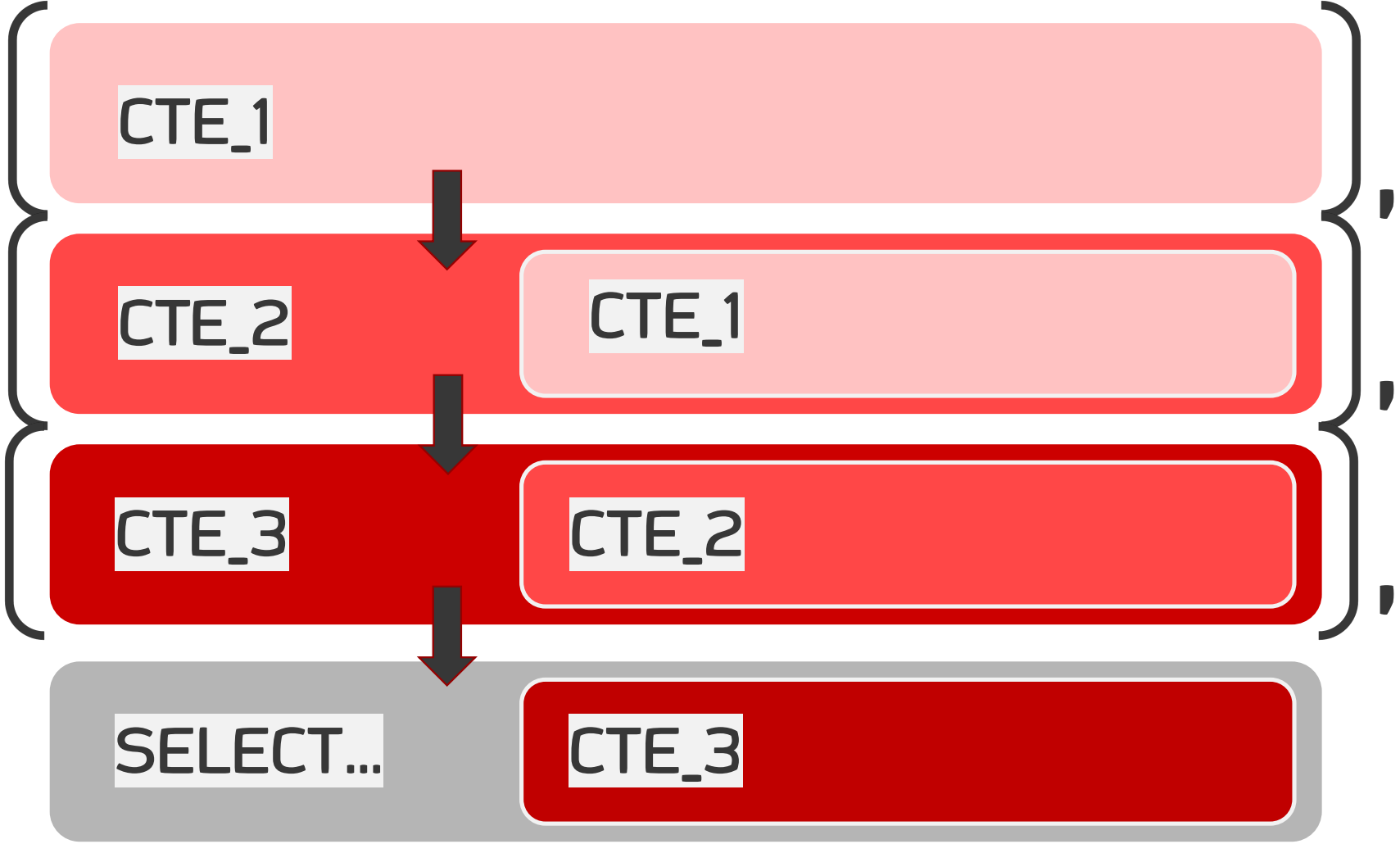
# Common Table Expression (CTE)

- Also called WITH queries
- Reference the output of the query by a unique name
- Prior to Postgres 12 the CTE was materialized first
  - Now in-lined unless you add MATERIALIZED

# Common Table Expression (CTE)

- Multiple CTEs can be chained together, referring to each other as you go
- Particularly helpful when you'll reuse a query more than once (readability)
- Name output columns with parenthesis

WITH



# CTEs

```
SELECT
  nullif(calories, '')::bigint AS calories,
  count(*) FILTER (WHERE calories is null) OVER (ORDER BY id) AS elf,
  id
FROM
  dec01
```

# CTEs

```
WITH inventory AS (  
  SELECT  
    nullif(calories, '')::bigint AS calories,  
    count(*) FILTER (WHERE calories is null) OVER (ORDER BY id) AS elf,  
    id  
  FROM  
    dec01  
)
```

# CTEs

```
WITH inventory AS (  
    SELECT  
        nullif(calories, '')::bigint AS calories,  
        count(*) FILTER (WHERE calories is null) OVER (ORDER BY id) AS elf,  
        id  
    FROM  
        dec01  
)  
SELECT sum(calories) as c  
FROM inventory  
GROUP BY elf  
ORDER BY 1 desc;
```



# CTEs

```
WITH inventory (calories, elf) AS (  
    SELECT  
        nullif(calories, '')::bigint,  
        count(*) FILTER (WHERE calories is null) OVER (ORDER BY id),  
        id  
    FROM  
        dec01  
)  
SELECT sum(calories) as c  
FROM inventory  
GROUP BY elf  
ORDER BY 1 desc;
```

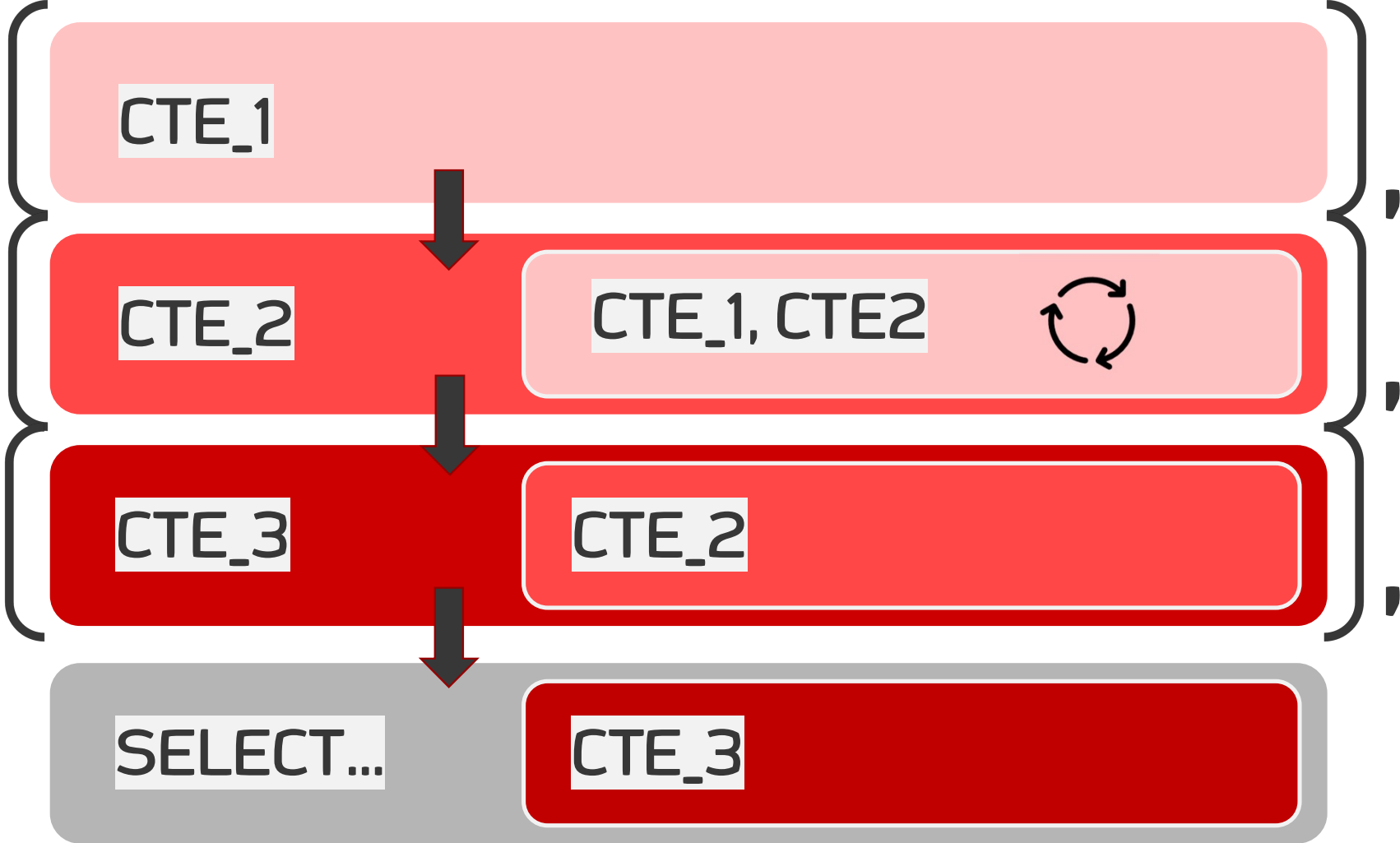
04/10

# Recursive CTEs

# Recursive CTEs

- The SQL language is declarative and batch-based by implementation
- Recursive CTEs provide iterative processing using SQL that wouldn't otherwise be possible
- Recursive CTEs allow SQL to be a Turing complete language

# WITH RECURSIVE



name	parent_folder	size
Folder_A		
Folder_A_1	Folder_A	
Folder_B	Folder_A	
Folder_A_2	Folder_A	
Folder_B_1	Folder_B	
File_A1.txt	Folder_A	1234
File_A2.txt	Folder_A	9876
File_B1.txt	Folder_B	4567

# Recursive CTEs

```
WITH recursive files AS (  
    -- start with a non-recursive, initial query  
    SELECT name, parent_folder, SIZE FROM files_on_disk  
    WHERE parent_folder IS NULL  
)  
SELECT * FROM files;
```

name	parent_folder	size
Folder_A		

# Recursive CTEs

```
WITH recursive files AS (  
    -- start with a non-recursive, initial query  
    SELECT name, parent_folder, SIZE  
    FROM files_on_disk  
    WHERE parent_folder IS NULL  
UNION ALL  
    SELECT fid.name, fid.parent_folder, fid.SIZE  
    FROM files_on_disk fid  
    INNER JOIN files f ON fid.parent_folder = f.name  
)  
SELECT * FROM files;
```

# Recursive CTEs

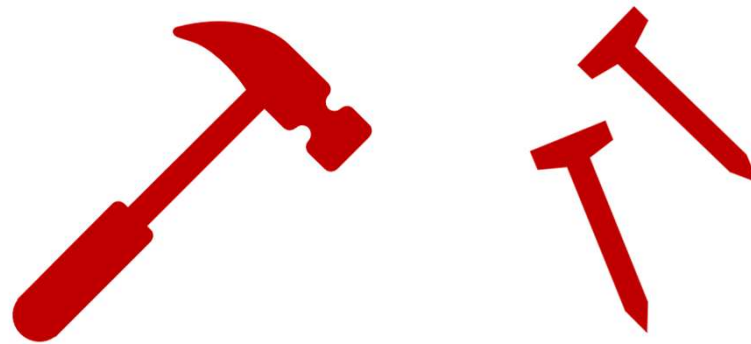
name	parent_folder	size	
Folder A			<-- Initial query
Folder_A_1	Folder_A		<--
Folder_B	Folder_A		
Folder_A_2	Folder_A		- Result of first join
File_A1.txt	Folder_A	1234	
File_A2.txt	Folder_A	6789	<--

```
WITH recursive files AS (  
    ...  
    UNION ALL  
        SELECT fid.name, fid.parent_folder, fid.SIZE  
        FROM files_on_disk fid  
        INNER JOIN files f ON fid.parent_folder = f.name  
    )  
SELECT * FROM files;
```



# Recursive CTEs - Caution!

- Recursion continues until working table is empty
- Make sure there is an ending point (or add one!)



**05/10**  
**FILTER Clause**

# **FILTER Clause**

- Available for many aggregate and window functions
- An internal predicate for the aggregate as rows pass through
- Useful in place of pivot-type queries
- Also helpful to spot sequences of data (rows preceding)

06/10  
Text to...

# Convert text to...

- Arrays, JSON, Tables
- Arrays and JSON are helpful as intermediate stores, particularly in recursive queries
- Both are fully supported datatypes in PostgreSQL, including indexing
- Many functions can output either datatype

# Arrays, Tables, & JSON

- Some functions for converting text
  - `string_to_array`
  - `regexp_matches`
  - `regexp_split_to_table`
  - `string_to_table`
  - `json_each` & `json_object_agg`

☰ We All Deserve Arrays: The Undervalued PostgreSQL Superpower

**We All Deserve Arrays!  
The Hidden Superpower of  
PostgreSQL**

Ryan Booz

Scale20x  
March 2023

Scroll for details

redgate



<https://bit.ly/ryan-booz-2023-talks>



07/10

# CROSS JOIN LATERAL



# CROSS JOIN LATERAL

- For every row on the left, execute query on the right
- Output is the product of both sets
- Allows chained queries to "reach back" to previous result sets for data
- Very useful with Set Returning Functions (SRF)

```
SELECT T.a, CJ.b, CJ2.c FROM
```

```
SELECT string_to_table(T.a, null)
```

```
SELECT string_to_array(CJ.a, null)
```

,

CJ,

CJ2

# CROSS JOIN LATERAL

- Also useful for simplifying SQL at a higher level by hiding calculations lower
- Reorganize data by returning VALUES

# CROSS JOIN LATERAL

```
...
select hm.step,
        hm.x, hm.y,
        h.x, h.y,
        t.x, t.y
from tmove tm
      join hmove hm on tm.step+1 = hm.step
cross join lateral
      (VALUES (tm.hx+hm.x, tm.hy+hm.y)) as h(x,y)
cross join lateral
      (VALUES (
        case when abs(h.y-tm.ty) = 2 then h.x
          when abs(tm.tx-h.x) <= 1 then tm.tx
          else tm.tx + hm.x end,
        case when abs(h.x-tm.tx) = 2 then h.y
          when abs(tm.ty-h.y) <= 1 then tm.ty
          else tm.ty + hm.y end
      )) t(x,y)
...

```

# CROSS JOIN LATERAL

```
...
select hm.step,
       hm.x, hm.y,
       h.x, h.y,
       t.x, t.y
from tmove tm
     join hmove hm on tm.step+1 = hm.step
cross join lateral
    (VALUES (tm.hx+hm.x, tm.hy+hm.y)) as h(x,y)
cross join lateral
    (VALUES (
        case when abs(h.y-tm.ty) = 2 then h.x
              when abs(tm.tx-h.x) <= 1 then tm.tx
              else tm.tx + hm.x end,
        case when abs(h.x-tm.tx) = 2 then h.y
              when abs(tm.ty-h.y) <= 1 then tm.ty
              else tm.ty + hm.y end
    )) t(x,y)
...
```

08/10  
WITH ORDINALITY

# WITH ORDINALITY

- Any Set Returning Function can also return the ordinal value of each row
- Faster than ROW\_NUMBER()
- Retains order without an ORDER BY

09/10  
WINDOW Functions



# WINDOW Functions

- Aggregates on steroids that work in context of the current query row
- Look backwards and forwards
- Powerful data analysis tool that can be challenging to master...
- ...but worth the investment!

10/10  
Range Type

# Range Type

- Ranges of dates and numbers
- Multi-range values supported in PostgreSQL 14+
- Can be inclusive or exclusive of each bound
- Many built-in range operators for easy comparison
- Indexable!

# Bonus Community

# PostgreSQL Community

- Twitter
- Slack
- Discord
- LinkedIn
- Postgres Weekly Newsletter
- PostgreSQL.life Interviews
- #PGSQLPhriday monthly blog event

# PostgreSQL Community

- Vik Fearing
- Feike Steenbergen
- David Kohn
- Sven Klemm
- John Pruitt
- Tobias Petry
- Bruce Momjain
- Andreas Scherbaum
- Ryan Lambert
- More, more, more...

 THANK YOU! 

[github.com/ryanbooz/presentations](https://github.com/ryanbooz/presentations)