We All Deserve Arrays! The Hidden Superpower of PostgreSQL

Ryan Booz

Scale20x
March 2023

redgate

redgate | About me

Ryan Booz

PostgreSQL & DevOps Advocate

@ryanbooz

/in/ryanbooz

www.softwareandbooz.com

github.com/ryanbooz/presentations

Agenda

01 Intro: 

02 Arrays in PostgreSQL

03 Working with Arrays

04 Pattern Matching for ELT

05 Inserting Rows with Arrays

06 Conclusion

redgate

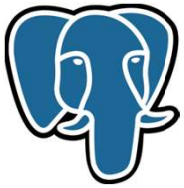
01/06

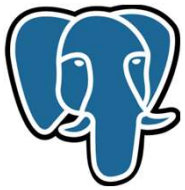
Intro: 

redgate

My Journey

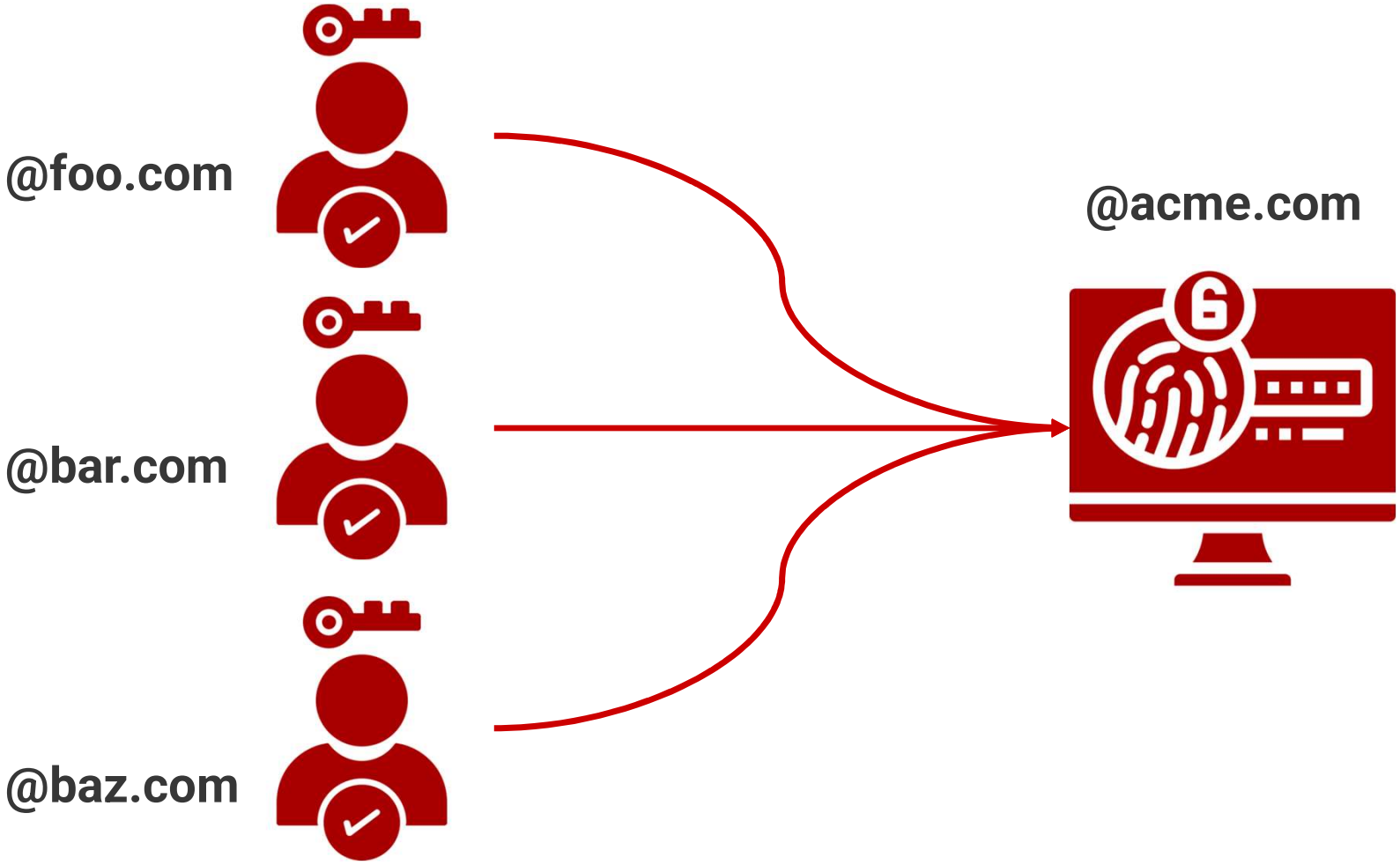
Work



Hobby

1999 2004 2018 2020 2022

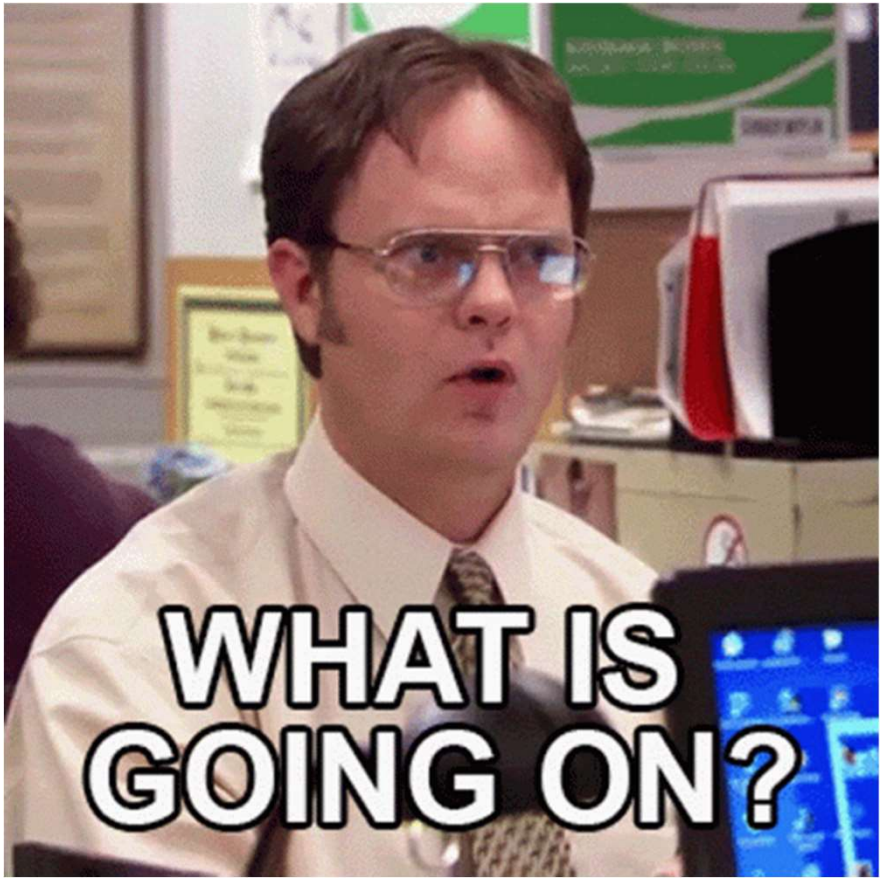


redgate

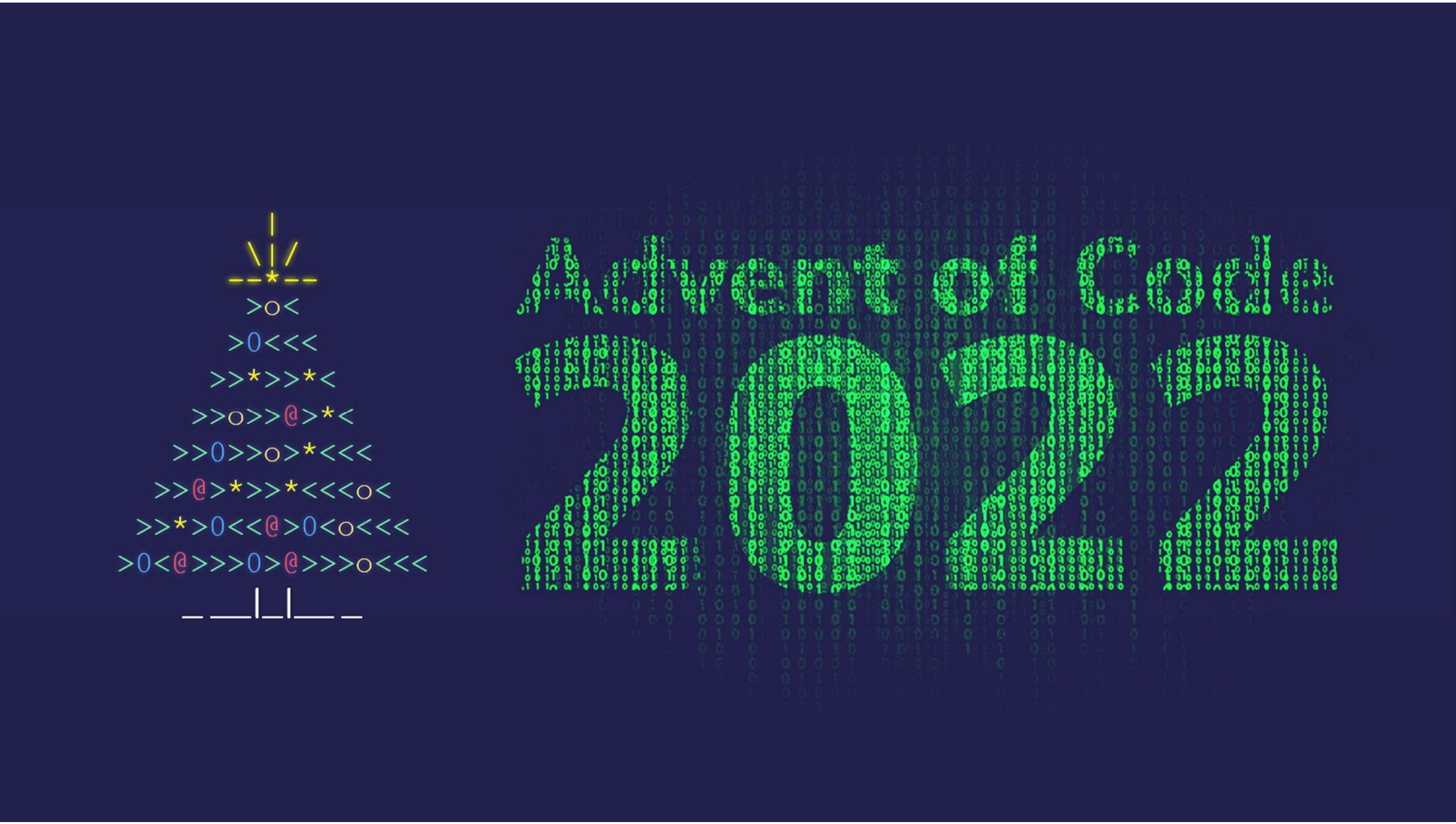
```
public class AuthProvider {
      public int AuthProviderId { get; set; }
      public string ProviderName { get; set; }
      public string RedirectURL { get; set; }
}

public class AuthProviderDomain {
      public int AuthProviderId { get; set; }
      public int ClientId { get; set; }
      public string Domain { get; set; }
}
```

```
public class AuthProvider {
    public int AuthProviderID { get; set; }
    public int ClientId { get; set; }
    public string ProviderName { get; set; }
    public string RedirectURL { get; set; }
    public List<string> Domains { get; set; }
}
```

```
CREATE TABLE auth_provider (
    auth_provider_id int PRIMARY KEY,
    client_id int not null,
    provider_name text not null,
    redirect_url text not null,
    domains text[] not null
);
```


WHAT IS
GOING ON?

A DAILY WORD GAME


Merry Christmas
2022

02/06
Arrays

redgate

Arrays

- Lists of related data, of the same type
- Multi-dimensional arrays are a list of arrays
- Referenced by numerical position
- PostgreSQL uses 1-based positioning

PostgreSQL Arrays

- **Columns** can be defined as types
- **Functions** and **Procedures** support arrays internally for data processing and logic
- Array literal:

  `'{1,2,3,4}'` or `'{"test","one","two","three"}'`
- Constructor:

  `ARRAY[1,2,3,4]` or `ARRAY['test','one','two','three']`

DEMO

redgate

```sql
CREATE TABLE film (
    film_id int PRIMARY KEY,
    title TEXT NOT NULL,
    film_type TEXT[] NULL
);

INSERT INTO film
VALUES (1,
        'Power to Postgres',
       '{documentary,thriller,action}');

INSERT INTO film
VALUES (2,
        'PostgreSQL 2: The SQL',
        ARRAY['documentary','suspense','action']);
```

```
=# SELECT * FROM film;
film_id|title                 |film_type                       |
-------+----------------------+--------------------------------+
      1|Power to Postgres     |{documentary,thriller,action}|
      2|PostgreSQL 2: The SQL|{documentary,suspense,action}|


=# SELECT film_type[1] FROM film;
film_type  |
-----------+
documentary|
documentary|

=# SELECT film_type[0] FROM film;
film_type  |
-----------+
NULL       |
NULL       |
```

WARNING

Arrays are not sets; searching for specific array elements can be a sign of database misdesign.

Consider using a separate table with a row for each item that would be an array element. This will be easier to search, and is likely to scale better for a large number of elements

https://www.postgresql.org/docs/current/arrays.html

03/06
Working With Arrays

redgate

DEMO

redgate

UNNEST to pivot an array to rows

```
SELECT title, unnest(film_type) FROM film
WHERE film_id=1;

title                    |unnest     |
-------------------------+-----------+
Power to Postgres        |documentary|
Power to Postgres        |thriller   |
Power to Postgres        |action     |
PostgreSQL 2: The SQL|documentary|
PostgreSQL 2: The SQL|suspense   |
PostgreSQL 2: The SQL|action     |
```

redgate

Aggregate to pivot rows to array

```
SELECT array_agg(title) FROM film;

array_agg                                          |
---------------------------------------------------+
{"Power to Postgres","PostgreSQL 2: The SQL"}|


SELECT array_agg(title ORDER BY title) FROM film;

array_agg                                          |
---------------------------------------------------+
{"PostgreSQL 2: The SQL","Power to Postgres"}|
```

Slice and dice arrays

```
SELECT title, film_type[1:2] FROM film;
SELECT title, film_type[:2] FROM film;
SELECT title,
       film_type[:array_length(film_type,1)-1]
FROM film;
```

```
title                   |film_type              |
------------------------+-----------------------+
Power to Postgres       |{documentary,thriller}|
PostgreSQL 2: The SQL|{documentary,suspense}|
```

Position in array

```
SELECT title, film_type[1:2] FROM film;
SELECT title, film_type[:2] FROM film;
SELECT title,
       film_type[:array_length(film_type,1)-1]
FROM film;
```

```
title                   |film_type              |
------------------------+-----------------------+
Power to Postgres       |{documentary,thriller}|
PostgreSQL 2: The SQL|{documentary,suspense}|
```

Updating an array

```
UPDATE film SET film_type = '{documentary,thriller,action}'
WHERE film_id=2;

film_type                          |
-----------------------------------+
{documentary,thriller,action}|


UPDATE film SET film_type[2] = 'suspense'
WHERE film_id=2;

film_type                          |
-----------------------------------+
{documentary,suspense,action}|
```

Multi-dimensional

```
SELECT ARRAY[[1,2,3],[4,5,6],[7,8,9]];

array                       |
----------------------------+
{{1,2,3},{4,5,6},{7,8,9}}|


SELECT ARRAY[[1,2,3],[4,5,6],[7,8,9,10]];

ERROR: multidimensional arrays must have array expressions
with matching dimensions
```

redgate

Appending arrays

```
UPDATE film SET film_type = '{documentary,thriller,action}'
WHERE film_id=2;

film_type                          |
-----------------------------------+
{documentary,thriller,action}|


UPDATE film SET film_type[2] = 'suspense'
WHERE film_id=2;

film_type                          |
-----------------------------------+
{documentary,suspense,action}|
```

redgate

Searching an array

```
-- is it found in any element of the array
SELECT * FROM film WHERE 'documentary' = any(film_type);
SELECT * FROM film WHERE film_type[1] = 'documentary';

film_id|title                 |film_type                        |
-------+--------------------+--------------------------------+
      1|Power to Postgres     |{documentary,thriller,action}|
      2|PostgreSQL 2: The SQL|{documentary,suspense,action}|

SELECT * FROM film WHERE 'suspense' = any(film_type);
-- do the arrays overlap in any way
SELECT * FROM film WHERE film_type && '{suspense}';

film_id|title                 |film_type                        |
-------+--------------------+--------------------------------+
      2|PostgreSQL 2: The SQL|{documentary,suspense,action}|
```

redgate

Array containment

```
=# SELECT * FROM film WHERE film_type @> '{suspense}';

film_id|title                 |film_type                        |
-------+----------------------+---------------------------------+
      2|PostgreSQL 2: The SQL|{documentary,suspense,action}|


=# SELECT * FROM film WHERE film_type <@ '{suspense}';

film_id|title                 |film_type                        |
-------+----------------------+---------------------------------+

=# SELECT * FROM film WHERE film_type <@
       '{suspense,documentary,action}';

film_id|title                 |film_type                        |
-------+----------------------+---------------------------------+
      2|PostgreSQL 2: The SQL|{documentary,suspense,action}|
```

redgate

Indexing arrays

```
CREATE INDEX idx_film_type ON film USING GIN (film_type);

-- encourage Postgres to use the index on this small table
SET enable_seqscan=OFF;
EXPLAIN analyze
SELECT * FROM film WHERE film_type @> '{suspense}';
```

```
QUERY PLAN                                                                                                                  |
----------------------------------------------------------------------------------------------------------------------------+
Bitmap Heap Scan on film  (cost=8.00..12.01 rows=1 width=68) (actual time=0.013..0.013 rows=1 loops=1)                      |
  Recheck Cond: (film_type @> '{suspense}'::text[])                                                                         |
  Heap Blocks: exact=1                                                                                                      |
  -> Bitmap Index Scan on idx_film_type  (cost=0.00..8.00 rows=1 width=0) (actual
time=0.008..0.008 rows=1 loops=1)|
        Index Cond: (film_type @> '{suspense}'::text[])                                                                     |
Planning Time: 0.062 ms                                                                                                     |
Execution Time: 0.030 ms                                                                                                    |
```

redgate

Cool!

So What?

ETL vs ELT

Extract, Transform, Load

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

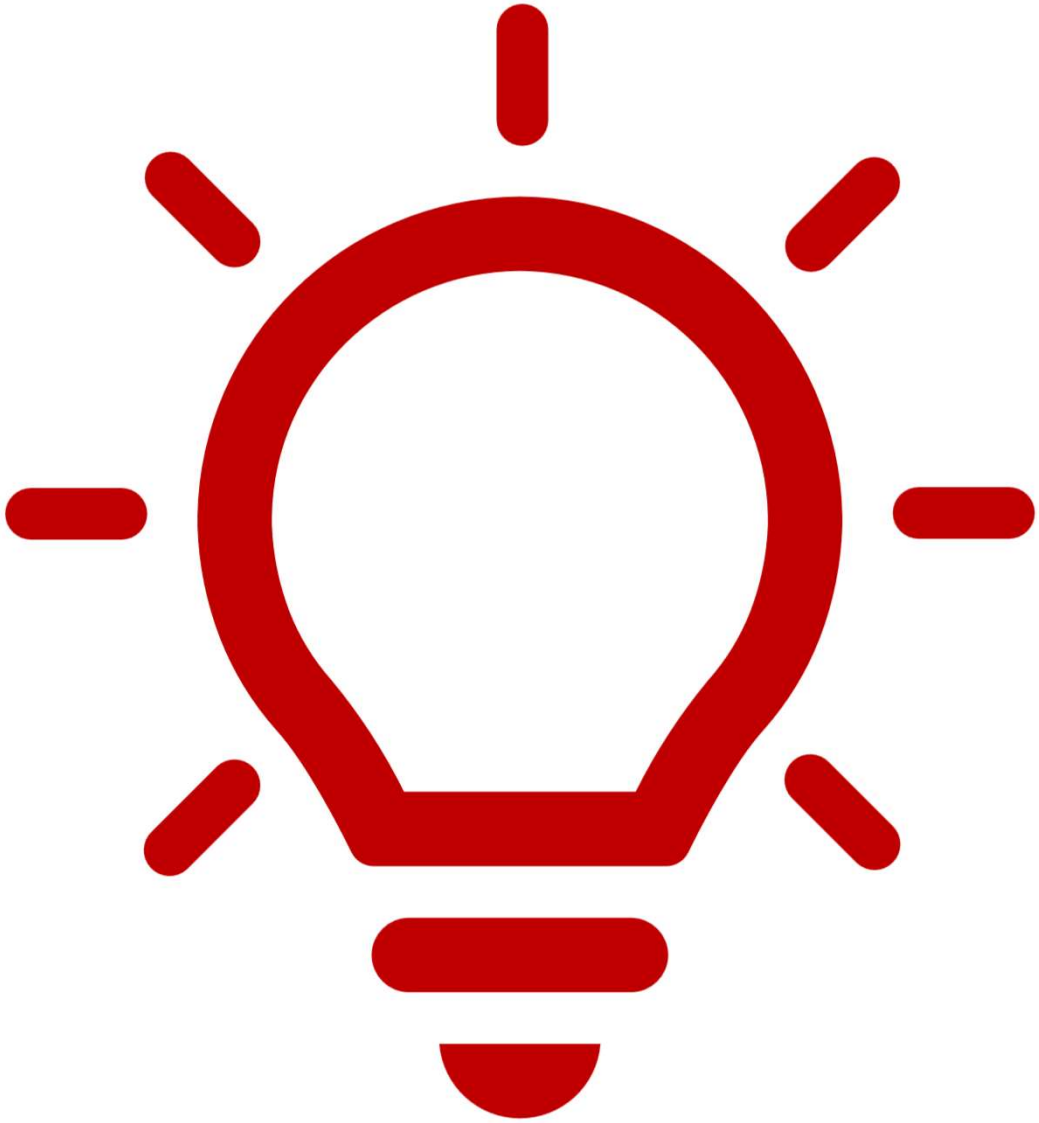
Extract, Load, Transform

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

redgate

ELT in PostgreSQL

- PostgreSQL has numerous functions for processing and transforming data
  - Regex
  - JSON
  - Stings
- Some of them utilize **arrays** as input or output



04/06

Pattern Matching for ELT

redgate

Pattern Matching

- regexp_split_to_array
- regexp_match
- regexp_matches
- string_to_array (PG 14+)

CROSS JOIN LATERAL and ORDINALITY

- CROSS JOIN LATERAL allows you to refer to previous relations for additional processing
- Iterate each row and execute a Set Returning Function (SRF)
- Adding WITH ORDINALITY provides the numerical value of an item in the set

DEMO

redgate

Creating arrays with pattern matching

```
-- essentially a "split to array"
=# SELECT string_to_array(title,' ') FROM film;
string_to_array               |
------------------------------+
{Power,to,Postgres}           |
{PostgreSQL,2:,The,SQL}       |

-- But it returns the string in whole if no match is found
=# SELECT string_to_array(title,':') FROM film;
string_to_array                  |
---------------------------------+
{"Power to Postgres"}            | <-- whole string
{"PostgreSQL 2"," The SQL"}      |
```

redgate

Creating arrays with regexp_*

```
-- more power than 'string_to_array' with regex capabilities
=# SELECT regexp_split_to_array(title,'\s+') FROM film;
regexp_split_to_array          |
-------------------------------+
{Power,to,Postgres}            |
{PostgreSQL,2:,The,SQL}        |

-- more advanced splitting and pattern matching using regex
SELECT regexp_match(title,'(.*):') FROM film;
regexp_match     |
-----------------+
NULL             |
{"PostgreSQL 2"}|
```

Creating arrays with regexp_*

```
-- only return the element from the title where
-- it started with text and contained a colon
=# SELECT p FROM film,
       regexp_match(title,'(.*):') AS p
   WHERE p IS NOT null;

regexp_match    |
----------------+
{"PostgreSQL 2"}|
```

Adding array order without row_number()

```
-- and finally, we can use WITH ORDINALITY with the
-- output of any SRF
=# SELECT title, ft.* FROM film,
       unnest(film_type)
       WITH ORDINALITY AS ft(film_type,array_order);
title                |film_type  |array_order|
---------------------+-----------+-----------+
Power to Postgres    |documentary|          1|
Power to Postgres    |thriller   |          2|
Power to Postgres    |action     |          3|
...                  |...        |        ...|
```

redgate

Adding array order without row_number()

```
-- first split the word and then unnest that
-- array for word order in this example
=# SELECT ft.* FROM film,
       unnest(regexp_split_to_array(title,' '))
       WITH ORDINALITY ft(word,o);

word       |o|
-----------+-+
Power      |1|
to         |2|
Postgres   |3|
PostgreSQL|1|
2:         |2|
The        |3|
SQL        |4|
```

05/06

Inserting Rows With Arrays

redgate

ARRAY values INSERT

- Can be faster than multi-valued INSERT
  - …in some cases
- Avoids the 65,535 parameter limit 😀
- YMMV with language support for PostgreSQL arrays
- Caution: May not handle custom types correctly

Inserting rows with arrays

```
INSERT INTO film (film_id, title)
    SELECT * FROM UNNEST(
        $1::int[],
        $2::TEXT[]
    );
```

redgate

DEMO

redgate

Inserting rows with arrays

```
/*
* Inserting with arrays
*/
=# INSERT INTO film (film_id, title)
       SELECT * FROM UNNEST('{3,4}'::int[],
       '{Postgres 95: The beginning,Postgres 6: A new
       beginning}'::TEXT[]);

=# SELECT film_id, title FROM film;
film_id|title                          |
-------+-------------------------------+
      1|Power to Postgres              |
      2|PostgreSQL 2: The SQL          |
      3|Postgres 95: The beginning     |
      4|Postgres 6: A new beginning    |
```

redgate

06/06
Conclusion

redgate

Conclusion

- Arrays are one of the unique and powerful datatypes provided by PostgreSQL
- Learning how to use and manipulate arrays can open new avenues for processing data in the database
- Arrays in PostgreSQL are 

redgate

What Questions do you have?

redgate

 THANK YOU! 

github.com/ryanbooz/presentations

redgate