Arrays and JSON in Postgres

Advanced SQL and Cloud

NASHVILLE SOFTWARE SCHOOL

Objectives

1. Understand how Postgres stores arrays and JSON/JSONB data
2. Access, manipulate, and query array elements and JSON values
3. Use array and JSON functions to transform and extract structured information
4. Prepare data for row-wise expansion using unnest() or JSON functions in later queries

Introduction

You should be familiar with lists and dictionaries from your study of Python. Postgres has similar structures, here called arrays and JSON.

These structures are useful for many real-world datasets which may have:

- Multi-value columns (tags, categories, codes)
- Nested or hierarchical data

Introduction

Example: Addresses

```
{
  "street": "123 Main St",
  "city": "Nashville",
  "state": "TN",
  "zip": "37211"
}
```

NASHVILLE SOFTWARE SCHOOL

Introduction

Example: Orders

```
{
  "order_id": 1001,
  "items": [
    {"product": "Laptop", "qty": 1},
    {"product": "Mouse", "qty": 2}
  ]
}
```

Introduction

Example: Electronic Health Record (EHR) Data

```
{
  "patient_id": 12345,
  "conditions": ["Diabetes", "Hypertension"],
  "medications": [
    {"drug": "Metformin", "dose": "500mg"},
    {"drug": "Lisinopril", "dose": "10mg"}
  ]
}
```

Introduction

JSON and Arrays are useful for many reasons:

- Schema flexibility: data structure can change without altering the table schema
- Capturing rich context: easily store nested information
- Quick ingestion: data can be loaded before fully normalizing it

However, they can make filtering, grouping, or joining more challenging and make consistency and data validation difficult.

NASHVILLE SOFTWARE SCHOOL

Arrays

What?

- Columns that can hold multiple values of the same type.
- Think: Python lists or numpy arrays
- Data type will be text[] or int[], for example.

Querying Arrays

Selecting from an array

- Think: Python lists.
- Can select by index (where the index starts at 1)
  - arr[2]
- Can also select slices (where both lower and upper bound are inclusive)
  - arr[2:4]

Querying Arrays

Filtering with arrays

- @> checks whether an array contains all elements of the array on the right

  `ARRAY[1,4,3] @> ARRAY[3,1,3]` → `True`

- && check whether there is overlap between two arrays

  `ARRAY[1,4,3] && ARRAY[2,1]` → `True`

Useful Functions Involving Arrays

- UNNEST(arr)
  - Expands input array(s) into multiple rows.
- ARRAY_AGG(exp)
  - Combines multiple values into an array.
- ARRAY_TO_STRING
  - Concatenates an array into a string.

Example:

```
SELECT product_id, UNNEST(tags) AS tag
FROM products;
```

JSON/JSONB

What?

- Columns that store semi-structured data in JSON format
- JSONB = binary; faster for queries
    - Note: keys must be unique when using JSONB
- JSON preserves formatting and is stored as raw text; slower for queries
    - Note: can have multiple instances of the same key in a record

Querying JSON

You can select by key or by index from a json column.

-> extract JSON object field as JSON

```
'{"a": {"b":"foo"}}'::json -> 'a' → {"b":"foo"}
```

->> extract JSON object field as text

```
'{"a":1,"b":2}'::json ->> 'b' → 2
```

NASHVILLE SOFTWARE SCHOOL

Querying JSON

There are a number of different operators that can be used on JSON/JSONB to query them.

See https://www.postgresql.org/docs/current/functions-json.html

Querying JSON

You can also specify a path if navigating nested JSON.

#> extract JSON sub-object

```
'{"a": {"b": ["foo","bar"]}}'::json #> '{a,b,1}' → "bar"
```

->> extract JSON sub-object as text

```
'{"a": {"b": ["foo","bar"]}}'::json #>> '{a,b,1}' → bar
```

NASHVILLE SOFTWARE SCHOOL

Querying JSON

You can also specify a path if navigating nested JSON.

@> and <@ check where one JSON value contains (is contained in) another

? checks whether a string is a top-level key

?| checks whether any of an array of strings is a top-level key

?& checks whether all of an array of strings is a top-level key

JSON Functions

There are a number of useful json functions:

- **JSON_EACH** expands a json object across multiple rows into key/value pairs
- **JSON_ARRAY_ELEMENTS** expands a json array into a set of JSON values
- **JSON_OBJECT_KEYS** returns the set of keys

Tips for Working with JSON and Arrays

- Peek inside before querying using JSON_EACH or JSON_OBJECT_KEYS to see the structure.
- Extract what you need early using CTEs or views that pull out key fields from JSON for reuse.
- Watch for inconsistent keys or missing data; JSON can have typos, missing fields, or inconsistent nesting.
- Be mindful that arrays don't enforce uniqueness or ordering.