Understanding JSON data in Postgres

INTRODUCTION TO NOSQL

Jake Roach
Data Engineer

datacamp

JSON and JSONB in Postgres

```
{
    'guardian': 'mother',
    'status': 'A',
    'educations': [4, 4],
    'jobs': {
        'P1': 'teacher',
        'P2': 'at_home'
    }
}
```

```
CREATE TABLE students
    <column-name> JSON,
    <column-bame> JSONB
;
```

JSON

- Store data in JSON format
- Key-values pairs, arrays, nested objects

JSONB

- Data is stored in binary format
- Similar to BSON in MongoDB
- More efficient storage and retrieval
- Allows for additional indexing

Why semi-structured data in Postgres?

school	age	address	parent_meta
GP	18	U	{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
GP	17	U	{ 'guardian': 'father', 'status': 'T', 'educations': [1, 1], 'jobs': { 'P1': 'at_home', 'P2': 'other' } }

Querying JSON data with Postgres

```
SELECT
    address,
    famsize,
    ...
FROM students
[WHERE | GROUP BY | ORDER BY];
```

We'll also be able:

- Insert JSON-formatted records to a Postgres table
- Tabular to JSON, JSON to tabular
- Extract individual records from JSON objects

- `row_to_json`, `json_to_record`
- `->`, `->>`, `#>`, `#>>` operators
- `json_extract_path`, `json_extract_path_text`

Executing queries with sqlalchemy and pandas

```
import sqlalchemy
import pandas as pd
```

```
# Create a connection
db_engine = sqlalchemy.create_engine(
    "postgresql+psycopg2://<user>:<password>@<host>:5432/<database>"
)
```

```
# Write a query
query = "SELECT * FROM table_name;"
```

```
# Execute the query, show results
results = pd.read_sql(query, db_engine)
```

Let's practice!

INTRODUCTION TO NOSQL

Storing JSON data in Postgres

INTRODUCTION TO NOSQL

Jake Roach
Data Engineer

datacamp

INSERT INTO and COPY JSON records to Postgres

```
INSERT INTO students (school, age, address, parent_meta) VALUES (
    'GP',
    18,
    'U',
    '{\"guardian\": \"mother\", ...  \"P2\": \"at_home\"}}'
);
```

Populate a table with contents of a file using `COPY ... FROM`

```
COPY students FROM 'students.csv' DELIMITER ',' CSV, HEADER;
```

Turning tabular data into JSON format

school	age	address
GP	18	U
GP	17	U
MS	19	R

row_to_json
{"f1": "GP", "f2": 18, "f3": "U"}
{"f1": "GP", "f2": 18, "f3": "U"}
{"f1": "MS", "f2": 19, "f3": "R"}

`row_to_json` function

- Converts a row to `JSON`
- Use with the `row()` function, and pass column names

```
SELECT
    row_to_json(row(
        school,
        age,
        address
    ))
FROM students;
```

Extracting keys from JSON

parent_meta
{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
...

json_object_keys
guardian
status
educations
jobs
...

`json_object_keys` function

- Extracts keys from a column of type `JSON`

```
SELECT
    json_object_keys(parent_meta)
FROM students;
```

- Pair with `DISTINCT` to find all unique keys

```
SELECT
    DISTINCT json_object_keys(parent_meta)
FROM students;
```

Review

```
SELECT
    row_to_json(row(
        <column-1>,
        <column-2>,
        ...
    ))
FROM <table-name>;
```

```
SELECT
    DISTINCT json_object_keys(parent_meta)
FROM <table-name>;
```

Let's practice!

INTRODUCTION TO NOSQL

Querying JSON data using Postgres

INTRODUCTION TO NOSQL

Jake Roach
Data Engineer

datacamp

Querying JSON data with Postgres

parent_meta
{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
...

guardian	status
'mother'	A
...	...

`->` operator

- Takes a key, returns field as JSON

`->>` operator

- Takes a key, returns field as text

```
SELECT
    parent_meta -> 'guardian' AS guardian
    parent_meta ->> 'status' AS status
FROM student;
```

Querying nested JSON objects

parent_meta
{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
...

jobs_P1	jobs_P2
teacher	at_home
...	...

To query nested JSON objects:

- Use `->` and `->>` in tandem
- First, return nested object
- Then, extract the field

```
SELECT
    parent_meta -> 'jobs' ->> 'P1' AS jobs_P1,
    parent_meta -> 'jobs' ->> 'P2' AS jobs_P2
FROM student;
```

Querying JSON arrays

parent_meta
{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
...

educations_0	educations_1
4	4
...	...

Accessing JSON array elements:

- Pass an `INT` to `->`, returns field as JSON
- Pass an `INT` to `->>`, returns field as text

```
SELECT
    parent_meta -> 'educations' ->> 0
    parent_meta -> 'educations' ->> 1
FROM student;
```

Finding the type of data store in JSON objects

parent_meta

<pre>{
  'guardian': 'mother',
  'status': 'A',
  'educations': [4, 4],
  'jobs': {
  'P1': 'teacher',
  'P2': 'at_home'
  }
}</pre>

...

json_typeof
object
...

`json_typeof` function

- Returns the type of the outermost object
- Use with `->`
- Useful for debugging, building queries
- Typically not used with the `->>` operator

```
SELECT
    json_typeof(parent_meta -> 'jobs')
FROM students;
```

Review

```
SELECT
    -- Top-level fields
    <column-name> -> '<field-name>' AS <alias>,
    <column-name> ->> '<field-name>' AS <alias>,

    -- Nested fields
    <column-name> -> '<parent-field-name>' ->> '<nested-field-name>' AS <alias>,

    -- Arrays
    <column-name> -> '<parent-field-name>' -> 0 AS <alias>,
    <column-name> -> '<parent-field-name>' ->> 1 AS <alias>,

    -- Type of
    json_typeof(<column-name> -> <field-name>) AS <alias>
FROM <table-name>;
```

Let's practice!

INTRODUCTION TO NOSQL

Advanced Postgres JSON query techniques

INTRODUCTION TO NOSQL

Jake Roach
Data Engineer

datacamp

Querying nested JSON data

```
SELECT
    parent_meta -> 'jobs' ->> 'P1' AS jobs_P1,
    parent_meta -> 'jobs' ->> 'P2' AS jobs_P2
FROM student;
```

To make querying nested data easier:

- `#>` and `#>>` operators
- `json_extract_path` and `json_extract_path_text` functions

#> and #>> operators

parent_meta
{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
...

jobs	jobs_P1	income	jobs_P2
{'P1': ...}	'teacher'	null	at_home
...	...	...	...

`#>` operator

- Called on column, takes a string array
- Returns `null` if path does not exist
- `#>>` returns field as text

```
SELECT
    parent_meta #> '{jobs}' AS jobs,
    parent_meta #> '{jobs, P1}' AS jobs_P1,
    parent_meta #> '{jobs, income}' AS income,
    parent_meta #>> '{jobs, P2}' AS jobs_P2
FROM student;
```

json_extract_path and json_extract_path_text

`json_extract_path`

- Provide column and arbitrary list of fields
- Returns `null` if path does not exist
- `json_extract_path_text`

```
SELECT
  json_extract_path(parent_meta, 'jobs') AS jobs,
  json_extract_path(parent_meta, 'jobs', 'P1') AS jobs_P1,
  json_extract_path(parent_meta, 'jobs', 'income') AS income,
  json_extract_path_text(parent_meta, 'jobs', 'P2') AS jobs_P2,
FROM student;
```

parent_meta
{ 'guardian': 'mother', 'status': 'A', 'educations': [4, 4], 'jobs': { 'P1': 'teacher', 'P2': 'at_home' } }
...

jobs	jobs_P1	income	jobs_P2
{'P1': ...}	'teacher'	null	at_home
...	...	...	...

Review

```
SELECT
    <column-name> #> '{<field-name>}' AS <alias>,
    <column-name> #> '{<field-name>, <field-name>}' AS <alias>,
    <column-name> #>> '{<field-name>, <field-name>}' AS <alias>
FROM <table-name>;
```

```
SELECT
  json_extract_path(<column-name>, '<field-name>') AS <alias>,
  json_extract_path(<column-name>, '<field-name>', '<field-name>') AS <alias>,
  json_extract_path_text(<column-name>, '<field-name>', '<field-name>') AS <alias>
FROM <table-name>;
```

Let's practice!

INTRODUCTION TO NOSQL