DWDC

JSON Processing in the Database

Getting, processing, and reshaping JSON data using PostgreSQL 9.4

Ryan B. Harvey

January 7, 2015

DATAWRANGLERSDC

JSON Data

- JavaScript Object Notation (JSON) data interchange format: RFC 7159
- Data formatted in JavaScript syntax
- Basic elements:
  - Object {“key”: “value”, “key2”: “value”}
  - List [“string”, 5, 6.24, true, “other string”]
  - Atomics “string”, 5, 2.64, true, false, null

IETF RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format

JSON Data: Example

DWDC


```
[
  {
    "id": "usajobs:377106500",
    "url": "https://www.usajobs.gov/GetJob/ViewDetails/377106500",
    "maximum": 254848,
    "minimum": 98000,
    "end_date": "2015-07-29",
    "locations": ["San Francisco, CA", "Washington, DC"],
    "start_date": "2014-07-31",
...
  },
  {
    "id": "usajobs:382608000",
    "url": "https://www.usajobs.gov/GetJob/ViewDetails/382608000",
...
  },
...
]
```


DigitalGov Jobs API: http://api.usa.gov/jobs/search.json?lat_lon=37.783333,-122.416667&query=full-time+positions&size=100&from=0

But... Analysis Data

DWDC

- As a data analyst/scientist/hacker/..., we often want tabular analysis data
- JSON format is convenient for API responses, but not usually for analysis
- Somehow, we need to munge that data into a clean table (often lots of work)
- PostgreSQL adds JSON processing to the already available relational model (unlike other relational DBMS with JSON)

Now, to the DataBase!

DWDC

- For this, we'll be looking at PostgreSQL version 9.4 (Released Dec 18, 2014)
- PostgreSQL adds extensions to SQL for JSON manipulation
- JSON is supported via column types
  - json (text, for fast inserts; 9.2, improved in 9.3)
  - jsonb (binary, indexable, for fast reads; 9.4)
- Other databases do this differently!

Review: Relational Data

- Relational data is organized in tables consisting of columns and rows
- Fields (columns) consist of a column name and data type constraint
- Records (rows) in a table have a common field (column) structure and order
- Records (rows) are linked across tables by key fields

Relational Data Model: Codd, Edgar F. “A Relational Model of Data for Large Shared Data Banks” (1970)

SQL: Working with Objects

DWDC

- Data Definition Language (DB Objects)
  - CREATE (table, index, view, function, ...)
  - ALTER (table, index, view, function, ...)
  - DROP (table, index, view, function, ...)

SQL: Working with Rows

DWDC

- Query Language (Records)
  - SELECT ... FROM ...
  - INSERT INTO ...
  - UPDATE ... SET ...
  - DELETE FROM ...

JSON in PostgreSQL

DWDC

```
CREATE TABLE my_raw_api_data (
  response    JSON, /* or JSONB */
  captured_at TIMESTAMP WITH TIME ZONE
              NOT NULL
              DEFAULT CURRENT_TIMESTAMP,
  captured_by CHARACTER VARYING
              NOT NULL
              DEFAULT CURRENT_USER
);
```

JSON in PostgreSQL

DWDC

```
/* Note specified column list: default
   values used for everything else.     */

COPY my_raw_api_data (response)
FROM PROGRAM
  'curl "http://api.usa.gov/jobs/..."';

SELECT response, captured_at, captured_by
FROM my_raw_api_data;
```

JSON Operators

```
/* Field 'a' of object j */
SELECT j->'a' FROM my_table; /* json return */
SELECT j->>'a' FROM my_table; /* text return */

/* Second element of list j (zero indexed) */
SELECT j->1 FROM my_table; /* json return */
SELECT j->>1 FROM my_table; /* text return */

/* Walk object j along specified path */
SELECT j#>'{a,1,b}' FROM my_table; /* json */
SELECT j#>>'{a,1,b}' FROM my_table; /* text */
```

Parsing JSON

```
/* If response is a list. [ ... ] */

SELECT json_array_length(response)
FROM my_raw_api_data;

SELECT json_array_elements(response)
FROM my_raw_api_data;
```

Parsing JSON

DWDC

```
/* If response is an object. { ... } */
SELECT json_object_keys(response)
FROM my_raw_api_data;

SELECT response->'a', response#>'{a,b}'
FROM my_raw_api_data;

SELECT response->>'id' AS object_id,
       (jsonb_each(response)).key AS field_name,
       (jsonb_each(response)).value AS field_value
FROM my_raw_api_data;
```

Populating Tables

```
/* If response is an object. { ... } */
CREATE TABLE my_table (...field list...);

SELECT
  json_populate_record(NULL::my_table, response)
FROM my_raw_api_data;

INSERT INTO my_table (...columns...)
SELECT ...manipulate results of above...
FROM (...above query...);
```

JSONB Key Existence

DWDC

```
/* Does key 'a' exist in response? */
SELECT response ? 'a'
FROM my_raw_api_data;

/* Do any of the keys listed exist in response? */
SELECT response ?| array['a','b','c']
FROM my_raw_api_data;

/* Do all of the keys listed exist in response? */
SELECT response ?& array['a','b','c']
FROM my_raw_api_data;
```

JSONB Containment

DWDC

```
/* Is the specified JSON contained in response? */

SELECT response @> '{"a":1, "b":2}'::jsonb
FROM my_raw_api_data;

SELECT '{"a":1, "b":2}'::jsonb <@ response
FROM my_raw_api_data;
```

JSONB Indexing

DWDC

- For the jsonb data type, containment and existence tests can be sped up considerably via the GIN index types.
  - jsonb_ops (default): indexes every key and value; supports @>, ?, ?&, ?| ops
  - jsonb_path_ops: indexes hash of value and keys leading to it; only supports @>

Caveats: Use UTF8

- If you're processing JSON, it's best to use the UTF8 character set.
- PostgreSQL (and RFC 7159) allow Unicode escape sequences of the form \uXXXX (where X is a hex digit).
- The jsonb type restricts Unicode escapes for non-ASCII characters (above \u007F) unless the database is UTF8.

Caveats: Nulls

- The JSON null value does not have its own type in PostgreSQL (it's a TEXT value), and is not the SQL NULL type.
- To see the difference:
  - ```
    /* JSON null value */
    SELECT json_typeof('null'::json);
    ```
  - ```
    /* SQL NULL value */
    SELECT json_typeof(NULL::json);
    ```

Relevant Docs

DWDC

- IETF RFC 7159
  http://rfc7159.net/rfc7159
- PostgreSQL 9.4 Documentation
  - JSON Types
    http://www.postgresql.org/docs/9.4/interactive/datatype-json.html
  - JSON Functions and Operators
    http://www.postgresql.org/docs/9.4/interactive/functions-json.html

Example Code & Data

DWDC

- All code for examples is on GitHub at:
  https://github.com/nihonjinrxs/dwdc-january2015

- Examples use data from two public APIs:
  - Open Weather Map
    http://openweathermap.org/api
  - DigitalGov Jobs API
    http://search.digitalgov.gov/developer/jobs.html

PPDC

Ryan B. Harvey

http://datascientist.guru
ryan.b.harvey@gmail.com
@nihonjinrxs
+ryan.b.harvey

Employment & Affiliations*

IT Project Manager
Office of Management and Budget
Executive Office of the President

Data Scientist & Software Architect
Kitchology Inc.

Research Affiliate
Norbert Wiener Center for Harmonic Analysis & Applications
College of Computer, Mathematical & Natural Sciences
University of Maryland at College Park

Thank you!

Questions?

* My remarks, presentation and prepared materials are my own, and do not represent the views of my employers.