PostgreSQL as a Document Store

CS383 -- April 2

Setup

Make a table

- jsonb or json
  - 'b' is for binary -- or better
    - 'b' is similar to Mongo BSON
- Use
  - json when lots of inserts and simple queries
    - only json operators are -> and ->>
  - jsonb when need complex queries
- Can have any number of normal SQL table items in same table

```
drop table if exists jsonact;

create table jsonact (
    id int generated always as identity,
    jjdata jsonb,
    primary key(id)
);
```

Add Data

- Using same code as shown last week except put into PSQL rather than Mongo
- " and '
- json.dumps makes a string from python dictionary & array

```
{'actor': 200,
 'first_name': 'JULIA',
 'last_name': 'FAWCETT',
 'films': [{'categ': 2,
            'catname': 'Animation',
            'filmid': 121,
            'filmname': 'CAROL TEXAS'},
           {'categ': 3,
            'catname': 'Children',
            'filmid': 993,
            'filmname': 'WRONG BEHAVIOR'},
           ...]}
```

```
for rr in rrr:
    print(rr)
    cursor.execute("insert into jsonact(jjdata) values('{}')"
                   .format(json.dumps(rr)))
```

Fields list?

jsonb_object_keys(COLUMN_NAME)

- What fields are in your documents,
  - how often?
- So in this table, all of the documents have exactly the same keys
  - 4*199=796

```
SELECT jsonb_object_keys(jjdata) from jsonact
....
796 rows

with aaa(keys) as (SELECT jsonb_object_keys(jjdata) from jsonact)
     select count(keys), keys
            from aaa
            group by keys;
 count |    keys
-------+------------
   199 | actor
   199 | films
   199 | first_name
   199 | last_name
(4 rows)
```

Basics

-> and ->>

- -> gets a JSON object
- ->> gets a stringified objects
- Most JSON operators have > and >> versions
  - Note quotation marks
    - in query
    - in results

```
sakila=# select jjdata->'first_name'
     from jsonact
     order by jjdata->'first_name'
     limit 2;
 ?column?
----------
 "ADAM"
 "ADAM"

sakila=# select jjdata->>'first_name' as
first_name
          from jsonact
          order by jjdata->>'first_name'
          limit 2;
 first_name
------------
 ADAM
 ADAM
```

Select .. where

always use ->>

- You might be able to use ->, I have never gotten it to work
- This works for any field at top level in document
- All usual SQL comparators work
  - like, in, <, <=, ...
- BUT ->> returns text so if you want a numeric comparison must cast
    - (jjdata->>'actor')::int=19
    - cast(jjdata->>'actor' as int)=19

```
sakila=# select *
            from jsonact
            where jjdata->>'first_name'='BOB';

 id | jjdata
----+--------
 19 | {"actor": 19, "films": [{"categ": 1, "filmid":
212, "catname": "Action", "filmname": "DARN
FORRESTER"}, {"categ": 2, "filmid": 208,
"catname": "Animation", "filmname": "DARES
PLUTO"}, {"categ": 14, "filmid": 711,
"catname": "Sci-Fi", "filmname": "RAGING
AIRPLANE"}, ....], "last_name": "FAWCETT",
"first_name": "BOB"}
(1 row)
```

Alternate query operator

more like Mongo

- @> (and @>>)
  - specify column on left and JSON to match on right
- I usually find this less useful
  - and rather annoying
    - '' and ""

```
select jjdata->>'first_name'
                as first_name,
        jjdata->>'last_name'
                as last_name
from jsonact
where jjdata @> '{"first_name":"BOB"}';

 first_name | last_name
------------+-----------
 BOB        | FAWCETT
(1 row)
```

One-to-One Embedded Documents

just stack ->

```
create table sample_table (json_data jsonb);
insert into sample_table
values
    ('{ "year": "2011", "make":"Toyota", "model":"Camry", "misc": {"color": "Gray", "doors": "4"}}'),
    ('{ "year": "2017", "make":"Honda", "model":"Civic", "misc": {"color": "White", "doors": "4"}}'),
    ('{ "year": "2017", "make":"Toyota", "model":"Camry", "misc": {"color": "Red", "doors": "2"}}'),
    ('{ "year": "2023", "make":"Honda", "model":"Accord"}'),
    ('{ "year": "1908", "make":"Ford", "model":"T", "misc": {"doors": "2"}}')
;
select * from sample_table where json_data->'misc'->>'color'='Red';
                                          json_data
-------------------------------------------------------------------------------------
 {"make": "Toyota", "misc": {"color": "Red", "doors": "2"}, "year": "2017", "model": "Camry"}


select json_data->'misc'->>'color' as color from sample_table
where (json_data->'misc'->>'doors')::int>3;
 color
-------
 Gray
 White
 Gray
 White
(4 rows)
```

Casting to do numeric compaisons

One-to-Many Embedded Documents

like films in the actor table

- note that {"categ":16} is contained in []
- finds all actors who were in a category 16 movie at least once

```
select jjdata->'first_name' as first_name,
       jjdata->'last_name' as last_name
  from jsonact
  where jjdata @> '{"films":[{"categ":16}]}'
  limit 2;

 first_name | last_name
------------+-----------
 "NICK"     | "WAHLBERG"
 "ED"       | "CHASE"
```

JSONB_ARRAY_ELEMENTS

getting one from the many (not easy in Mongo)

- JSONB_ARRAY_ELEMENTS breaks up the array
  - as expected
- But, still get every film by any actor who was in a category 16 film, not just the category 16 films.

```
select jjdata->'first_name' as first_name,
       jjdata->'last_name' as last_name,
       JSONB_ARRAY_ELEMENTS(jjdata->'films') as films
from jsonact
where jjdata @> '{"films":[{"categ":16}]}';

 first_name | last_name  |
films
------------+------------
+-------------------------------------------------------------
----------------------
 "NICK"     | "WAHLBERG" | {"categ": 1, "filmid": 105,
"catname": "Action", "filmname": "BULL SHAWSHANK"}
 "NICK"     | "WAHLBERG" | {"categ": 2, "filmid": 314,
"catname": "Animation", "filmname": "FIGHT JAWBREAKER"}
 "NICK"     | "WAHLBERG" | {"categ": 3, "filmid": 485,
"catname": "Children", "filmname": "JERSEY SASSY"}
```

"Lateral" Joins

A postgreSQL thing

- query at right looks like a cross join
  - so would give count(jsonact)*count(jsonb...) rows
    - 199*(199*20ish)
- But it is a "lateral" join so only gives 199*20ish
  - 5447
- select count(*) from film_actor;
  - 5462 (I do not know what happened to the missing 15)

```
select count(*)
    from jsonact,
        JSONB_ARRAY_ELEMENTS(jjdata->'films') AS films;
```

Lateral again

- Clearly see here that actors are only getting "their" films

```
select jjdata->>'actor', count(*)
from jsonact,
     JSONB_ARRAY_ELEMENTS(jjdata->'films') AS films
where jjdata->>'first_name' in ('BOB', 'LUCILLE')
group by jjdata->>'actor';
```

```
 ?column? | count
----------+-------
 138      |    24
 19       |    25
 20       |    30
```

Lateral for just categ 16

- First one gives all films for any actor who was in a cat 16 film
  - this is Mongo equivalent
- Part of second query is essentially identical to first query
  - why '= 16' twice

```
select count(*)
from jsonact,
     JSONB_ARRAY_ELEMENTS(jjdata->'films') AS films
where jjdata @> '{"films":[{"categ":16}]}';
 --- 4536
```

```
with aaa(id, jjdata, filmdata) as (select *
from jsonact,
     JSONB_ARRAY_ELEMENTS(jjdata->'films') AS films
where jjdata @> '{"films":[{"categ":16}]}')
select jjdata->>'first_name', filmdata->>'filmname'
from aaa
where (filmdata->>'categ')::int=16;
 --- 318
```

```
select jjdata->>'first_name' as first_name,
       films->>'filmname' as filmname
from jsonact,
     JSONB_ARRAY_ELEMENTS(jjdata->'films') AS films
where (films->>'categ')::int=16;
 --- 318
```

Queries from last week

but in Postgres

Query	Postgres
find one actor whose actorid is less than 5	select * from jsonact where (jjdata->>'actor')::int<5 limit 1;
find all actors whose actorid is less than or equal to 5	select * from jsonact where (jjdata->>'actor')::int<=5;
find all actors whose actor id is greater than 198	select * from jsonact where (jjdata->>'actor')::int > 198;
find all actors in films with an id greater than or equal to 990	select * from jsonact where (jjdata->>'actor')::int >= 990;
find all actors whose name is not BOB	select * from jsonact where jjdata->>'first_name' != 'BOB';
find all actors whose name is BOB or LUCILLE (use in)	select * from jsonact where jjdata->>'first_name' in ('BOB', 'LUCILLE');
find all actors whose name is not BOB or LUCILLE	select * from jsonact where jjdata->>'first_name' not in ('BOB', 'LUCILLE');

Queries from Lab

but in Postgres

number of actors with first name BOB	select count(*) from jsonact where jjdata->>'first_name' = 'BOB';
number of actors with first name BOB or PENELOPE	select count(*) from jsonact where jjdata->>'first_name' in ('BOB', 'PENELOPE');
actor with first name that starts with S and end with R show only first name	select jjdata->'first_name' from jsonact where jjdata->>'first_name' like 'S%R';
actor with a Z in either first or last name	select jjdata->>'first_name' as firstname, jjdata->>'last_name' as lastname from jsonact where jjdata->>'first_name' like '%Z%' or jjdata->>'last_name' like
first name of all actors in film with id 513	select jjdata->>'first_name' as firstname, from jsonact where jjdata @> '{"films":[{"filmid":513}]}';
same as previous, but only showing the name of the film	select films->'filmname', films->'filmid' as fid from jsonact, jsonb_array_elements(jjdata->'films') as films where (films->'filmid')::int=513;
actor whose first name has an E and has been in a film in category 16	select films->'filmname', jjdata->>'first_name' from jsonact, jsonb_array_elements(jjdata->'films') as films where (films->'categ')::int=16 and jjdata->>'first_name' like '%E%';

References

- https://gist.github.com/kcranston/b309664dc8864e680813f0f2b87c3b5b
- https://hashrocket.com/blog/posts/dealing-with-nested-json-objects-in-postgresql