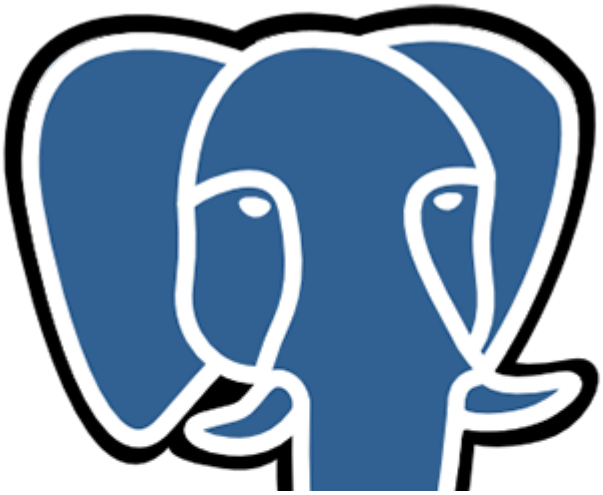
DATABASE TIPS

How To Query a JSONB Array of Objects as a Recordset in PostgreSQL

gravity well (Rob Tomlin) Follow
Aug 4, 2020 · 6 min read ★

JSONB Array of Objects Into Rows Using the jsonb_to_recordset() function



Source. PostgreSQL Wiki

PostgreSQL is an awesome database, with an awesome data type, JSON. It actually has two JSON data types, json and jsonb.

> *JSON data types are for storing JSON (JavaScript Object Notation) data.*
>
> *There are two JSON data types: json and jsonb. They accept almost identical sets of values as input. The major practical difference is one of efficiency. The json data type stores an exact copy of the input text, which processing functions must reparse on each execution; while jsonb data is stored in a decomposed binary format that makes it slightly slower to input due to added conversion overhead, but significantly faster to process, since no reparsing is needed. jsonb also supports indexing, which can be a significant advantage.*

- It is typically recommended to use jsonb.
- PostgreSQL is rich in functions and operators for querying this data type.

Our Goal

In this article we will focus on **one function in particular:**

jsonb_to_recordset(jsonb data).

This function is particularly useful when you have an Array of Objects and want to query against the Values of one or more Keys using standard comparison operators including wildcards.

For example, we may have jsonb data like this, showing products purchased by a customer.

```
[{
  "productid": "3",
  "name": "Virtual Keyboard",
  "price": "150.00"

}, {
  "productid": "1",
  "name": "Dell 123 Laptop Computer",
  "price": "1300.00"

},
 {
  "productid": "8",
  "name": "LG Ultrawide Monitor",
  "price": "190.00"

}
]
```

If we have several customers and their purchases stored in a jsonb column, we may want to know things like,

- Who bought Virtual Keyboards (**Where name = 'Virtual Keyboard'**)
- Who bought any kind of Keyboard (**Where name Like '%Keyboard**)
- Who bought a Laptop and Keyboard (**Where name Like '%Laptop%' or name Like '%Keyboard'**)
- How many of each product (name) has been purchased.

and many others, of course.

Assumptions

I will assume you have PostgreSQL and something along the lines of pgAdmin.

Let's Get Started

Building Our Data

1. Open **pgAdmin** and create a database as desired.

2. Right-Click on the database name and choose **Query Tool**.

3. Run the snippet below to create a simple table that will have an id, purchaser name and a ***jsonb column that stores an array of json objects, which will store items purchased.***

```
CREATE TABLE public.purchases
(
    id   serial PRIMARY KEY,
    purchaser varchar(50),
    items_purchased jsonb
);
```

4. Run the snippet below to insert four records in to the table.

```
INSERT INTO purchases (purchaser,items_purchased) VALUES ('Bob','[{
  "productid": "1",
  "name": "Dell 123 Laptop Computer",
  "price": "1300.00"

},
 {
  "productid": "2",
  "name": "Mechanical Keyboard",
  "price": "120.00"

}
]');

INSERT INTO purchases (purchaser,items_purchased) VALUES ('Carol','[{
  "productid": "3",
  "name": "Virtual Keyboard",
  "price": "150.00"
```

```
}, {
  "productid": "1",
  "name": "Dell 123 Laptop Computer",
  "price": "1300.00"

},
 {
  "productid": "8",
  "name": "LG Ultrawide Monitor",
  "price": "190.00"

}
]');

INSERT INTO purchases (purchaser,items_purchased) VALUES ('Ted','[{
  "productid": "6",
  "name": "Ergonomic Keyboard",
  "price": "90.00"

},
 {
  "productid": "7",
  "name": "Dell 789 Desktop Computer",
  "price": "120.00"

}
]');

INSERT INTO purchases (purchaser,items_purchased) VALUES ('Alice','[{
  "productid": "7",
  "name": "Dell 789 Desktop Computer",
  "price": "120.00"

},
 {
  "productid": "2",
  "name": "Mechanical Keyboard",
  "price": "120.00"

}
]');
```

5. Run the snippet below to see,

```
select * from purchases;
```

a Output Explain Messages Notifications

id [PK] integer	purchaser character varying (50)	items_purchased jsonb
1	Bob	[{"name": "Dell 123 Laptop Computer", "price": "1300.00", "productid": 1}, {"...
2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": 3}, {"name": "Del...
3	Ted	[{"name": "Ergonomic Keyboard", "price": "90.00", "productid": 6}, {"name": "...
4	Alice	[{"name": "Dell 789 Desktop Computer", "price": "120.00", "productid": 7}, {"...

purchases with jsonb column of items purchased

Our Function

Understanding jsonb_to_recordset(jsonb data)

> *Builds an arbitrary set of records from a JSON array of objects (see note below). As with all functions returning record, the caller must explicitly define the structure of the record with an AS clause.*

Note:

> *Extracted key values can then be referenced in other parts of the query, like WHERE clauses.*

What does "*the caller must explicitly define the structure of the record with an AS clause.*" mean?

This means basically that we are defining the structure of our recordset , *the columns and their data type*. Similar to defining a relational table structure.

Examine the following code.

```
select * from purchases,jsonb_to_recordset(purchases.items_purchased)
as items(name text);
```

- In the FROM clause we reference the table and the jsonb column, purchases.items, which is passed to the jsonb_to_recordset() function. It returns a set of Key/Value pairs that we must define the storage for.

- In the AS clause we choose an *arbitrary* name, items, for our recordset structure, the storage of the Key/Values.
- In parentheses would be a comma separated list of the Keys we want (in this case just name) and its data type, text.
- **For each Key, a new column is created when the query is run**.
- Since a new column is created, we can reference it in the Select, Where, Group By and Order by clauses.

Run the code above to get,

Data Output Explain Messages Notifications

	id [PK] inte	purchaser character varying (50)	items_purchased jsonb	name text
1	1	Bob	[{"name": "Dell 123 Laptop Computer", "price": "1300.00", "productid": "1"}, {"name": "Mech...	Dell 123 Laptop Computer
2	1	Bob	[{"name": "Dell 123 Laptop Computer", "price": "1300.00", "productid": "1"}, {"name": "Mech...	Mechanical Keyboard
3	2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": "3"}, {"name": "Dell 123 Laptop ...	Virtual Keyboard
4	2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": "3"}, {"name": "Dell 123 Laptop ...	Dell 123 Laptop Computer
5	2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": "3"}, {"name": "Dell 123 Laptop ...	LG Ultrawide Monitor
6	3	Ted	[{"name": "Ergonomic Keyboard", "price": "90.00", "productid": "6"}, {"name": "Dell 789 Desk...	Ergonomic Keyboard
7	3	Ted	[{"name": "Ergonomic Keyboard", "price": "90.00", "productid": "6"}, {"name": "Dell 789 Desk...	Dell 789 Desktop Computer
8	4	Alice	[{"name": "Dell 789 Desktop Computer", "price": "120.00", "productid": "7"}, {"name": "Mech...	Dell 789 Desktop Computer
9	4	Alice	[{"name": "Dell 789 Desktop Computer", "price": "120.00", "productid": "7"}, {"name": "Mech...	Mechanical Keyboard

Observe the new column, name. The object Key is the column name and the Value the data.

Try this snippet to create two columns.

```
select * from purchases,jsonb_to_recordset(purchases.items_purchased)
as items(name text, price text);
```

Data Output Explain Messages Notifications

	id [PK] integ	purchaser character varying (50)	items_purchased jsonb	name text	price text
1	1	Bob	[{"name": "Dell 123 Laptop Computer", "price": "1300.00", "productid": "1"}, {"name": "Mechan...	Dell 123 Laptop Computer	1300.00
2	1	Bob	[{"name": "Dell 123 Laptop Computer", "price": "1300.00", "productid": "1"}, {"name": "Mechan...	Mechanical Keyboard	120.00
3	2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": "3"}, {"name": "Dell 123 Laptop Co...	Virtual Keyboard	150.00
4	2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": "3"}, {"name": "Dell 123 Laptop Co...	Dell 123 Laptop Computer	1300.00
5	2	Carol	[{"name": "Virtual Keyboard", "price": "150.00", "productid": "3"}, {"name": "Dell 123 Laptop Co...	LG Ultrawide Monitor	190.00
6	3	Ted	[{"name": "Ergonomic Keyboard", "price": "90.00", "productid": "6"}, {"name": "Dell 789 Deskto...	Ergonomic Keyboard	90.00
7	3	Ted	[{"name": "Ergonomic Keyboard", "price": "90.00", "productid": "6"}, {"name": "Dell 789 Deskto...	Dell 789 Desktop Computer	120.00
8	4	Alice	[{"name": "Dell 789 Desktop Computer", "price": "120.00", "productid": "7"}, {"name": "Mechan...	Dell 789 Desktop Computer	120.00
9	4	Alice	[{"name": "Dell 789 Desktop Computer", "price": "120.00", "productid": "7"}, {"name": "Mechan...	Mechanical Keyboard	120.00

Observe two new columns and data

Using Our New Columns

Now that we have the ability to create columns from our jsonb Keys, we can reference them in Select, Where, Group By and Order By clauses.

Try each of the following

1. Using the column in Select and Where

```
select purchaser,items.name from
purchases,jsonb_to_recordset(purchases.items_purchased) as items(name
text)
where items.name like '%Monitor';
```

Data Output Explain Messages Notifications

	purchaser character varying (50)	name text
1	Carol	LG Ultrawide Monitor

2. Using the column in Select and Where

```
select purchaser,items.name, items.price from
purchases,jsonb_to_recordset(purchases.items_purchased) as items(name
text, price text)
where TO_NUMBER(items.price,'9999')>=1000
```

Note: Because our prices are text, we needed to convert to a number. More on TO_NUMBER() can be found here.

Data Output Explain Messages Notifications

	purchaser character varying (50)	name text	price text
1	Bob	Dell 123 Laptop Computer	1300.00
2	Carol	Dell 123 Laptop Computer	1300.00

3. Using a column in a Group By

```
select items.name, count(*) as num from
purchases,jsonb_to_recordset(purchases.items_purchased) as items(name
text)
group by items.name
order by num Desc
```

Data Output | Explain | Messages | Notifications

	name text	num bigint
1	Dell 123 Laptop Computer	2
2	Dell 789 Desktop Computer	2
3	Mechanical Keyboard	2
4	Ergonomic Keyboard	1
5	LG Ultrawide Monitor	1
6	Virtual Keyboard	1

Conclusion

As mentioned before, this is not the only JSON function. There are many functions and operators for JSON data.

I am exploring them everyday as I built a RESTful API for a PERN stack.

If you found this useful, start exploring json, jsonb and the other operators and functions for JSON as well as the regular SQL operations available in PostgreSQL.

Thank you for reading and coding along!

You may also enjoy,

Working with the Array Data Type in PostgreSQL

Create, get, modify, add and remove data from an Array

levelup.gitconnected.com

Creating a Data Pagination Function in PostgreSQL

Using the LIMIT, OFFSET and FETCH NEXT operators.

levelup.gitconnected.com

Working with a JSONB Array of Objects in PostgreSQL

Get, Add and Remove JSON Objects from an Array

levelup.gitconnected.com

Querying SQL Server in Node.js Using Async/Await

A Cleaner Way To Query A Database

medium.com

Downloading Files in AWS S3 to a SQL Server RDS

And Bulk Load the Contents Into a Table

medium.com

Postgresql Postgres NoSQL Json Database