PostgreSQL JSON

Summary: in this tutorial, we will show you how to work with PostgreSQL JSON data type. In addition, we will introduce you to some of the most common PostgreSQL JSON operators and functions for handling JSON data.

JSON stands for JavaScript Object Notation. JSON is an open standard format that consists of key-value pairs. The main usage of JSON is to transport data between a server and web application. Unlike other formats, JSON is human-readable text.

PostgreSQL supports native JSON data type since version 9.2. It provides many functions and operators for manipulating JSON data.

Let's get started by creating a new table for practicing with JSON data type.

```
CREATE TABLE orders (
 ID serial NOT NULL PRIMARY KEY,
 info json NOT NULL
);
```

The `orders` table consists of two columns:

1. The `id` column is the primary key column that identifies the order.
2. The `info` column stores the data in the form of JSON.

Insert JSON data

To insert data into a JSON column, you have to ensure that data is in a valid JSON format. The following `INSERT` statement inserts a new row into the `orders` table.

```
INSERT INTO orders (info)
```

```
VALUES
 (
 '{ "customer": "John Doe", "items": {"product": "Bee
r","qty": 6}}'
 );
```

It means `John Doe` bought `6` bottle of `beers`.

Let's insert multiple rows at the same time.

```
INSERT INTO orders (info)
VALUES
 (
 '{ "customer": "Lily Bush", "items": {"product": "Diap
er","qty": 24}}'
 ),
 (
 '{ "customer": "Josh William", "items": {"product": "T
oy Car","qty": 1}}'
 ),
 (
 '{ "customer": "Mary Clark", "items": {"product": "Toy
 Train","qty": 2}}'
 );
```

Querying JSON data

To query JSON data, you use the `SELECT` statement, which is similar to querying other native data types:

```
SELECT
 info
FROM
 orders;
```

info
{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}
{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}
{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}
{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}

PostgreSQL returns a result set in the form of JSON.

PostgreSQL provides two native operators `->` and `->>` to help you query JSON data.

The operator `->` returns JSON object field by key.
The operator `->>` returns JSON object field by text.

The following query uses the operator `->` to get all customers in form of JSON:

```
SELECT
 info -> 'customer' AS customer
FROM
 orders;
```

customer
"John Doe"
"Lily Bush"
"Josh William"
"Mary Clark"

And the following query uses operator `->>` to get all customers in form of text:

```
SELECT
 info ->> 'customer' AS customer
FROM
 orders;
```

customer
John Doe
Lily Bush
Josh William
Mary Clark

Because `->` operator returns a JSON object, you can chain it with the operator `->>` to retrieve a specific node. For example, the following statement returns all products sold:

```
SELECT
 info -> 'items' ->> 'product' as product
FROM
 orders
ORDER BY
 product;
```

product
Beer
Diaper
Toy Car
Toy Train

First `info -> 'items'` returns items as JSON objects. And then `info->'items'->>'product'` returns all products as text.

Use JSON operator in WHERE clause

We can use the JSON operators in `WHERE` clause to filter the returning rows. For example, to find out who bought `Diaper`, we use the following query:

```
SELECT
 info ->> 'customer' AS customer
FROM
 orders
WHERE
 info -> 'items' ->> 'product' = 'Diaper'
```

customer
Lily Bush

To find out who bought two products at a time, we use the following query:

```
SELECT
 info ->> 'customer' AS customer,
 info -> 'items' ->> 'product' AS product
FROM
 orders
WHERE
 CAST (
 info -> 'items' ->> 'qty' AS INTEGER
 ) = 2
```

customer	product
Mary Clark	Toy Train

Notice that we used the type cast to convert the `qty` field into `INTEGER` type and compare it with two.

Apply aggregate functions to JSON data

We can apply aggregate functions such as MIN, MAX, AVERAGE, SUM, etc., to JSON data. For example, the following statement returns minimum quantity, maximum quantity, average quantity and the total quantity of products sold.

```
SELECT
 MIN (
 CAST (
 info -> 'items' ->> 'qty' AS INTEGER
 )
 ),
 MAX (
 CAST (
 info -> 'items' ->> 'qty' AS INTEGER
 )
 ),
 SUM (
 CAST (
 info -> 'items' ->> 'qty' AS INTEGER
 )
 ),
```

```
 AVG (
 CAST (
 info -> 'items' ->> 'qty' AS INTEGER
 )
 )

FROM
 orders
```

min	max	sum	avg
1	24	33	8.25

PostgreSQL JSON functions

PostgreSQL provides us with some functions to help you process JSON data.

json_each function

The `json_each()` function allows us to expand the outermost JSON object into a set of key-value pairs. See the following statement:

```
SELECT
 json_each (info)
FROM
 orders;
```

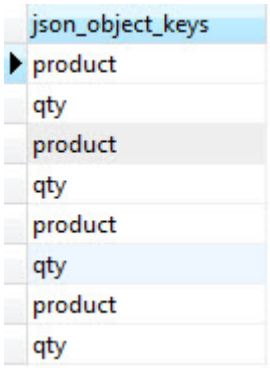
json_each
(customer,"""John Doe""")
(items,"{""product"": ""Beer"",""qty"": 6}")
(customer,"""Lily Bush""")
(items,"{""product"": ""Diaper"",""qty"": 24}")
(customer,"""Josh William""")
(items,"{""product"": ""Toy Car"",""qty"": 1}")
(customer,"""Mary Clark""")
(items,"{""product"": ""Toy Train"",""qty"": 2}")

If you want to get a set of key-value pairs as text, you use the `json_each_text()` function instead.

json_object_keys function

To get a set of keys in the outermost JSON object, you use the `json_object_keys()` function. The following query returns all keys of the nested `items` object in the `info` column

```
SELECT
 json_object_keys (info->'items')
FROM
 orders;
```



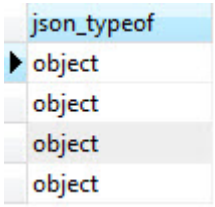
json_object_keys
product
qty
product
qty
product
qty
product
qty

json_typeof function

The `json_typeof()` function returns type of the outermost JSON value as a string. It can be `number`, `boolean`, `null`, `object`, `array`, and `string`.

The following query return the data type of the items:

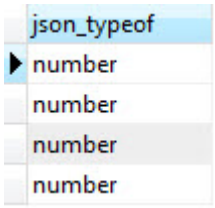
```
1  SELECT
2   json_typeof (info->'items')
3  FROM
4   orders;
```



json_typeof
object
object
object
object

The following query returns the data type of the qty field of the nested items JSON object.

```
1  SELECT
2   json_typeof (info->'items'->'qty')
3  FROM
4   orders;
```



json_typeof
number
number
number
number

There are more PostgreSQL JSON functions if you want to dig deeper.

In this tutorial, we have shown you how to work with PostgreSQL JSON data type. We showed you some of the most important JSON operators and functions that help you process JSON data more effectively.