Postgres Advanced Data Types

Fanavaran Anisa
Iran Linux House

Linux & Open Source Training Center

www.anisa.co.ir

Table of Contents

JSON Data Type

- JSON data **types are for storing JSON** (JavaScript Object Notation) data, as specified in RFC 7159.
- Such data can also be stored as text, but the JSON data types have the advantage of **enforcing that each stored value is valid** according to the JSON rules. There are also assorted **JSON-specific functions and operators** available for data stored in these data types

PostgreSQL offers two types for storing JSON data:

- JSON

- JSONB

PostgreSQL also provides the **jsonpath** data type

JSON vs JSONB

- The json and jsonb data types accept almost identical sets of values as input.
- The major practical difference is **efficiency**.
    - The **json** data **type stores an exact copy of the input text**, which processing functions must **reparse** on each execution
    - **jsonb** data is **stored in a decomposed binary format**
        - slower input
        - significantly faster to process

Jsonb also **supports indexing**, which can be a significant advantage

Which One to Use?

Why JSONB is the Always Better ?

- jsonb does not preserve white space
- does not preserve the order of object keys
- does not keep duplicate object keys. If duplicate keys are specified in the input, only the last value is kept.

In general, **most applications should prefer to store JSON data** as jsonb, unless there are quite specialized needs, such as legacy assumptions about ordering of object keys.

Which One to Use?

Why JSONB is the Always Better ?

- jsonb does not preserve white space
- does not preserve the order of object keys
- does not keep duplicate object keys. If duplicate keys are specified in the input, only the last value is kept.

In general, **most applications should prefer to store JSON data** as jsonb, unless there are quite specialized needs, such as legacy assumptions about ordering of object keys.

Which One to Use?

Why JSONB is the Always Better ?

- jsonb does not preserve white space
- does not preserve the order of object keys
- does not keep duplicate object keys. If duplicate keys are specified in the input, only the last value is kept.

In general, **most applications should prefer to store JSON data** as jsonb, unless there are quite specialized needs, such as legacy assumptions about ordering of object keys.

```
CREATE TABLE journal (
  id Int NOT NULL PRIMARY KEY,
  day VARCHAR(10),
  diary_information JSONB
);
```

First Steps

```
INSERT INTO journal (id, day, diary_information)
 VALUES ( 1, “Tuesday”,
'{"title": "My first day at work", "Feeling": "Mixed feeling"}' );
```

```
SELECT * FROM journal
```

	id [PK] character varying	age integer	diary_information jsonb
1	2	27	{"title": "My first day at work", "Feeling": "Mixed feelin...

First Steps – Functions & Operators

->: This operator allows you to extract a specific value from a JSON object, you specify the key as a “child” to the “parent”.

```
SELECT Id, day,
diary_information -> 'Feeling' AS Feeling
FROM journal;
```

First Steps – Functions & Operators

->>: This operator allows you to extract a JSON object field as text without the quotes around it from a JSON object.

```
SELECT id, day, dairy_information ->> 'Feeling' as Feeling
FROM products;
```

First Steps – Functions & Operators

json_agg: This function aggregates JSON values into a JSON array.

For example, SELECT json_agg(my_column) FROM my_table; will return a JSON array containing the values in the "my_column" column of the "my_table" table.

jsonb_set: This function updates a JSON object field with a new value

```
UPDATE journal SET diary_information =
 jsonb_set( diary_information, '{Feeling}', '"Excited"' )
WHERE id = 1;
```

First Steps – Functions & Operators

You can use a JSONB_BUILD_OBJECT function to insert a plain text record and this will convert it to JSON data

```
INSERT INTO journal (id, day, feeling)
VALUES ( 2, 'Wednesday',
JSONB_BUILD_OBJECT(
'Morning',
  'Everybody is annoying today',
'Evening',
'Cannot wait to go home')
);
;
```

@> Containment Operator

the @> operator tests if the left-hand JSON value contains the right-hand JSON value.
The <@ operator tests if the right-hand JSON value contains the left-hand JSON value

```
INSERT INTO employees (id, name, skills) VALUES
( 1, 'John', '[{"name": "Python", "level": "Intermediate"},
{"name": "JavaScript", "level": "Expert"}]' );
```

```
SELECT name FROM employees WHERE skills @ > '[{"name": "Python"}]' :: jsonb
```

@> Containment Operator

the @> operator tests if the left-hand JSON value contains the right-hand JSON value.
The <@ operator tests if the right-hand JSON value contains the left-hand JSON value

```
INSERT INTO employees (id, name, skills) VALUES
( 1, 'John', '[{"name": "Python", "level": "Intermediate"},
{"name": "JavaScript", "level": "Expert"}]' );
```

```
SELECT name FROM employees WHERE skills @ > '[{"name": "Python"}]' :: jsonb
```

JSON Path

```
SELECT jsonb_extract_path('{"brand": "Honda", "sold":
false}'::jsonb, 'brand');

SELECT '{"brand": "Honda", "sold": false}'::jsonb @> '{"brand":
"Honda"}'::jsonb AS result

SELECT
'{"brand": "Honda", "sold": false}' :jsonb @@ '$.brand ==
"Honda"' AS result_with_@@,
 '{"brand": "Honda", "sold": false}'::jsonb @? '$.brand ? (@ ==
"Honda")' AS result_with_@?;
```

HSTORE

The hstore module implements the hstore data type for storing key-value pairs in a single value. The keys and values are text strings only.

CREATE EXTENSION hstore;

```
CREATE TABLE books ( id serial primary key, title
VARCHAR (255), attr hstore );

INSERT INTO books (title, attr) VALUES (
'PostgreSQL Tutorial',
'"paperback" => "243",
 "publisher" => "postgresqltutorial.com",
"language" => "English",
"ISBN-13" => "978-1449370000",
 "weight" => "11.2 ounces"' );
```

HSTORE: Query/Update/Delete

```
SELECT
        title, attr -> 'weight' AS weight
FROM
        books
WHERE
        attr -> 'ISBN-13' = '978-1449370000';
```

```
UPDATE books
SET attr = attr || '"freeshipping"=>"yes"' :: hstore;
```

```
UPDATE books
SET attr = attr || '"freeshipping"=>"no"' :: hstore;
```

```
UPDATE books
SET attr = attr - 'weight'
WHERE title = 'PostgreSQL Cheat Sheet';
```

HSTORE – Advanced Queries

```
SELECT
  title,
  attr->'publisher' as publisher,
  attr
FROM
            books
WHERE
            attr ? 'publisher';
```

returns all rows with attr contains key publisher.

HSTORE - HSTORE – Advanced Queries

```
SELECT title
FROM books WHERE
attr @> '"weight"=>"11.2 ounces"':: hstore;
```

```
SELECT
        title
FROM
        books
WHERE
        attr ?& ARRAY [ 'language', 'weight' ];
```

ARRAY

In PostgreSQL, an array of a collection of elements that have the same data type.

Every data type has its companion array type e.g., integer has an integer[] array type, character has character[] array type.

```
CREATE TABLE contacts (
  id SERIAL PRIMARY KEY,
  name VARCHAR (100),
  phones TEXT []
);
```

```
column_name data_type [][]
```

ARRAY – Insert Data

```
INSERT INTO contacts (name, phones)
VALUES('John Doe',ARRAY [ '(408)-589-5846','(408)-589-5555' ]);
```

```
INSERT INTO contacts (name, phones)
VALUES('Lily Bush','{"(408)-589-5841"}'),
      ('William Gate','{"(408)-589-5842","(408)-589-58423"}');
```

ARRAY – Query

```
SELECT
  name,
  phones [ 1 ]
FROM
  contacts;
```

```
SELECT
  name,
  phones
FROM
  contacts
WHERE
  '(408)-589-5555' = ANY (phones);
```

Output:

```
   name    |              phones
-----------+---------------------------------
 John Doe | {(408)-589-5846,(408)-589-5555}
(1 row)
```

ARRAY

```
UPDATE contacts
SET phones [2] = '(408)-589-5843'
WHERE ID = 3
RETURNING *;
```

phones = '{"(408)-589-5843"}'

Output:

```
 id |     name     |                phones
----+--------------+---------------------------------
  3 | William Gate | {(408)-589-5842,(408)-589-5843}
(1 row)
```

Expanding ARRAY

```
SELECT
  name,
  unnest(phones)
FROM
  contacts;
```

Output:

```
     name      |     unnest
---------------+----------------
 John Doe      | (408)-589-5846
 John Doe      | (408)-589-5555
 Lily Bush     | (408)-589-5841
 William Gate  | (408)-589-5843
(4 rows)
```