# PostgreSQL'de NoSQL ve İlişkisel Veritabanı Birlikteliği

**JSON/JSONB** tipinde veri tutma, bu veriyi doğrudan ya da ilişkisel modelle birlikte kullanma yöntemleri.

@atifceylan

/in/atifceylan



M.Atıf Ceylan

# Ben Kimim?

 @atifceylan

 /in/atifceylan

- 25 Yıllık **BİLGİSAYARCI**
- 2000'den beri yazılımcı
  (yemek ve dil ayırt etmem, kuru-pilav ve **PHP**'den vazgeçmem!)
- 2002'den beri Linux, BSD, Solaris ve bunlar üzerindeki birçok servisle iştigal ediyorum.
- AWS, GCP ve bilimum bulutçuyla çalışıyorum.
- **PostgreSQL** Kullanıcıları ve Geliştiricileri Derneği üyesiyim.
- **AppstoniA /** appstonia.com
- **Cooksoft** / cooksoft.com.tr

# Sunum İçeriği

- JSON/JSONB veya Diğer Tiplerden Hangisini Kullanmalıyız?
- JSON/JSONB Veri Tipleri
- JSON Operatörleri
- JSON'da Index Kullanımı
- JSON Fonksiyonları
- İlişkisel Modelle Birlikte JSON Kullanım Örnekleri

# JSON/JSONB veya Diğer Tiplerden Hangisini Kullanmalıyız?

- En geniş veri tipi desteğine sahip veritabanı
- Her veri tipinde Array tip oluşturabilme (text[ ], json[ ] vb.)
- Amaca uygun veri tipi kullanma avantajları:
  - Kullanım kolaylıkları
  - Validasyon
  - Performans
  - Disk ve bellekteki boyut farkları

## PostgreSQL Standart Veri Tipleri (43 + range types)

bigint
bigserial
bit [ (n) ]
bit varying [ (n) ]
boolean
box
bytea
character [ (n) ]
character varying [ (n) ]
cidr
circle
date
double precision
inet
integer
interval [ fields ] [ (p) ]
**json**
**jsonb**
line
lseg
macaddr
macaddr8
money
numeric [ (p, s) ]
path
pg_lsn
pg_snapshot
point
polygon
real
smallint
smallserial
serial
text
time [ (p) ] [ without time zone ]
time [ (p) ] with time zone
timestamp [ (p) ] [ without time zone ]
timestamp [ (p) ] with time zone
tsquery
tsvector
txid_snapshot
uuid
**xml**

# PostgreSQL'de JSON/JSONB Tipler Nasıl?

- **JSON** ve **JSONB** iki farklı veri tipidir.
- Her iki tipte de veri geçerli bir **JSON** formatında olmalıdır (validation yapılıyor).
- **JSON** tipinde veri metin olarak, **JSONB**'de binary olarak tutulur.
- **JSON**'da veri orjinal haliyele tutulur, **JSONB**'de optimize edilir (anahtar/değer formatında tutulur).
- **JSON**'da her defasında**, JSONB**'de yalnızca girdi anında bir kez parse yapılır.
- **JSON**'da yazma daha hızlıdır.
- **JSONB** indexlenebilir (GIN/GIST, BTREE, HASH).
- **JSONB**'de aynı anahtardan 2 tane olmaz.

# Hangisini Ne Zaman Kullanmalıyız?

Girdi yapılan Json verisinin her defasında tümünün okunması gereken durumlarda (konfigürasyon, ayarlar vb.) **JSON**,

anahtar/değer okuması yapılması gereken tüm durumlarda **JSONB** olarak tutulmalıdır.

**JSON** neredeyse **TEXT** tipi ile aynı mantıktadır. Format doğrulaması (validation) için **TEXT** yerine **JSON** kullanın.

# JSON/JSONB Veri Tipleri

https://www.postgresql.org/docs/16/datatype-json.html

| JSON primitive type | PostgreSQL type | Notes |
|---|---|---|
| string | text | \u0000 is disallowed, as are Unicode escapes representing characters not available in the database encoding |
| number | numeric | **NaN** and **infinity** values are *disallowed* |
| boolean | boolean | Only lowercase **true** and **false** spellings are accepted |
| null | (none) | SQL NULL is a different concept |

# JSON/JSONB Veri Tipleri - Örnek

```
SELECT
  '5'::json AS number_type,
  '"foo"'::json AS text_type,
  'true'::json AS boolean_type,
  'null'::json AS null_value,
  '[1, 2, "foo", null]'::json AS json_array,
  '{"bar": "baz", "balance": 7.77, "active": false}'::json AS json_object,
  '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::json as json_object2
```

```
-[ RECORD 1 ]+--------------------------------------------------
number_type  | 5
text_type    | "foo"
boolean_type | true
null_value   | null
json_array   | [1, 2, "foo", null]
json_object  | {"bar": "baz", "balance": 7.77, "active": false}
json_object2 | {"foo": [true, "bar"], "tags": {"a": 1, "b": null}}
```

# JSON/JSONB Operatörleri

https://www.postgresql.org/docs/current/functions-json.html

| Operator | Right Operand Type | Description | Example | Example Result |
|---|---|---|---|---|
| -> | int | Get JSON array element (indexed from zero, negative integers count from the end) | `'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2` | `{"c":"baz"}` |
| -> | text | Get JSON object field by key | `'{"a": {"b":"foo"}}'::json->'a'` | `{"b":"foo"}` |
| ->> | int | Get JSON array element as text | `'[1,2,3]'::json->>2` | 3 |
| ->> | text | Get JSON object field as text | `'{"a":1,"b":2}'::json->>'b'` | 2 |
| #> | text[] | Get JSON object at specified path | `'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'` | `{"c": "foo"}` |
| #>> | text[] | Get JSON object at specified path as text | `'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'` | 3 |

# JSON/JSONB Operatörleri

- Postgres array tipinde indexler 1’den başlar, json array indexler 0’dan
  - `select ('{1,2,3}'::integer[])[1]  --1`
  - `select '[1,2,3]'::jsonb->1  --2`
- Array olmayan json’a index verirseniz null döner
  - `select '{"a":"foo","b":"bar","c":"baz"}'::jsonb->1 --null`

# JSON/JSONB Operatörleri - Karşılaştırma

| Operator | Right Operand Type | Description | Example | Result |
|---|---|---|---|---|
| @> | jsonb | Does the left JSON value contain the right JSON path/value entries at the top level? | `'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb` | t |
| <@ | jsonb | Are the left JSON path/value entries contained at the top level within the right JSON value? | `'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb` | t |
| ? | text | Does the string exist as a top-level key within the JSON value? | `'{"a":1, "b":2}'::jsonb ? 'b'` | t |
| ?\| | text[] | Do any of these array strings exist as top-level keys? | `'{"a":1, "b":2, "c":3}'::jsonb ?\| array['b', 'c']` | t |
| ?& | text[] | Do all of these array strings exist as top-level keys? | `'["a", "b"]'::jsonb ?& array['a', 'b']` | t |
| \|\| | jsonb | Concatenate two jsonb values into a new jsonb value | `'["a", "b"]'::jsonb \|\| '["c", "d"]'::jsonb` | `["a", "b", "c", "d"]` |

# JSON/JSONB Operatörleri - Karşılaştırma

| Operator | Right Operand Type | Description | Example | Result |
|---|---|---|---|---|
| - | text | Delete key/value pair or string element from left operand. Key/value pairs are matched based on their key value. | `'{"a": "b", "c": "d"}'::jsonb - 'a'` | `{"c": "d"}` |
| - | text[] | Delete multiple key/value pairs or string elements from left operand. Key/value pairs are matched based on their key value. | `'{"a": "b", "c": "d"}'::jsonb - '{a,c}'::text[]` | `{}` |
| - | integer | Delete the array element with specified index (Negative integers count from the end). Throws an error if top level container is not an array. | `'["a", "b"]'::jsonb - 1` | `["a"]` |
| #- | text[] | Delete the field or element with specified path (for JSON arrays, negative integers count from the end) | `'["a", {"b":1}]'::jsonb #- '{1,b}'` | `["a", {}]` |

# JSON/JSONB Operatör Kullanım Örnekleri

```
-- Simple scalar/primitive values contain only the identical value:
SELECT '"foo"'::jsonb @> '"foo"'::jsonb;

-- The array on the right side is contained within the one on the left:
SELECT '[1, 2, 3]'::jsonb @> '[1, 3]'::jsonb;

-- Order of array elements is not significant, so this is also true:
SELECT '[1, 2, 3]'::jsonb @> '[3, 1]'::jsonb;

-- Duplicate array elements don't matter either:
SELECT '[1, 2, 3]'::jsonb @> '[1, 2, 2]'::jsonb;

-- The object with a single pair on the right side is contained
-- within the object on the left side:
SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb": true}'::jsonb @>
'{"version": 9.4}'::jsonb;
```

# JSON/JSONB Operatör Kullanım Örnekleri

```
-- The array on the right side is not considered contained within the
-- array on the left, even though a similar array is nested within it:
SELECT '[1, 2, [1, 3]]'::jsonb @> '[1, 3]'::jsonb;  -- yields false

-- But with a layer of nesting, it is contained:
SELECT '[1, 2, [1, 3]]'::jsonb @> '[[1, 3]]'::jsonb;

-- Similarly, containment is not reported here:
SELECT '{"foo": {"bar": "baz"}}'::jsonb @> '{"bar": "baz"}'::jsonb;  -- yields false

-- A top-level key and an empty object is contained:
SELECT '{"foo": {"bar": "baz"}}'::jsonb @> '{"foo": {}}'::jsonb;

-- This array contains the primitive string value:
SELECT '["foo", "bar"]'::jsonb @> '"bar"'::jsonb;

-- This exception is not reciprocal -- non-containment is reported here:
SELECT '"bar"'::jsonb @> '["bar"]'::jsonb;  -- yields false
```

# JSON/JSONB Operatör Kullanım Örnekleri

```
-- String exists as array element:
SELECT '["foo", "bar", "baz"]'::jsonb ? 'bar';

-- String exists as object key:
SELECT '{"foo": "bar"}'::jsonb ? 'foo';

-- Object values are not considered:
SELECT '{"foo": "bar"}'::jsonb ? 'bar';  -- yields false

-- As with containment, existence must match at the top level:
SELECT '{"foo": {"bar": "baz"}}'::jsonb ? 'bar'; -- yields false

-- A string is considered to exist if it matches a primitive JSON string:
SELECT '"foo"'::jsonb ? 'foo';
```

# JSON/JSONB Sorgu Örnekleri

```
CREATE TABLE public.orders (
 id serial,
 customer_id integer,

 details JSONB
);

CREATE TABLE customers (
   id        serial,
   name      varchar(100),
   is_active boolean
);

CREATE TABLE product(
   id         serial,
   name       varchar(100),

   properties JSONB
);
```

# JSON/JSONB Sorgu Örnekleri

```
SELECT
      details->>'product_id' product id,
      sum((details->'qty')::integer) total_qty
FROM
      orders
WHERE
      details->'color' ? 'siyah'
GROUP BY product id
ORDER BY total_qty DESC;
```

| | product_id | total_qty |
|---|---|---|
| 1 | 103 | 31015 |
| 2 | 101 | 30782 |
| 3 | 102 | 29963 |

# JSON/JSONB Sorgu Örnekleri

```
SELECT
      details->>'product_id' product_id,
      details->>'qty' qty
FROM
      orders
WHERE
      details->>'qty' > '5';
```

| | product_id | qty |
|---|---|---|
| 1 | 102 | 7 |
| 2 | 102 | 8 |
| 3 | 101 | 6 |
| 4 | 101 | 9 |
| 5 | 103 | 8 |
| 6 | 102 | 8 |
| 7 | 101 | 9 |
| 8 | 102 | 8 |
| 9 | 102 | 9 |

# JSON/JSONB Sorgu Örnekleri

```
SELECT
      c.name, o.details
FROM
      orders o
JOIN
      customers c ON c.id=o.customer_id
WHERE
      details->>'color' = 'siyah';
```

| | name | details |
|---|---|---|
| 1 | Customer215 | {"qty": 10, "name": "Nita Cash", "note": "aliqua deserunt duis et", "color": "siyah", "phone": "+90 (894) 428-3373", "amount": 624, "address" |
| 2 | Customer772 | {"qty": 3, "name": "Tamara Pierce", "note": "enim aute minim anim", "color": "siyah", "phone": "+90 (901) 515-3851", "amount": 564, "address" |
| 3 | Customer237 | {"qty": 8, "name": "Ruby William", "note": "cillum exercitation voluptate ipsum", "color": "siyah", "phone": "+90 (984) 538-2357", "amount": |
| 4 | Customer630 | {"qty": 9, "name": "Medina Lane", "note": "Lorem exercitation duis in", "color": "siyah", "phone": "+90 (990) 522-3016", "amount": 602, "addr |
| 5 | Customer750 | {"qty": 9, "name": "Mcclure Lucas", "note": "esse excepteur qui aliquip", "color": "siyah", "phone": "+90 (999) 529-2245", "amount": 676, "ad |
| 6 | Customer952 | {"qty": 3, "name": "Macdonald Blake", "note": "cupidatat Lorem dolor quis", "color": "siyah", "phone": "+90 (982) 587-2329", "amount": 617, " |
| 7 | Customer223 | {"qty": 6, "name": "Hallie Sparks", "note": "ut magna veniam tempor", "color": "siyah", "phone": "+90 (936) 407-3247", "amount": 432, "addres |
| 8 | Customer717 | {"qty": 9, "name": "Aline Mcleod", "note": "do veniam culpa ipsum", "color": "siyah", "phone": "+90 (977) 534-2411", "amount": 430, "address" |
| 9 | Customer697 | {"qty": 6, "name": "Marcella Rodriguez", "note": "commodo amet ipsum veniam", "color": "siyah", "phone": "+90 (865) 600-3968", "amount": 983, |
| 10 | Customer721 | {"qty": 9, "name": "Hopkins Blanchard", "note": "in ad eu cupidatat", "color": "siyah", "phone": "+90 (868) 511-3163", "amount": 946, "addres |

# JSON/JSONB Sorgu Örnekleri

```
SELECT
      p.name, o.details->>'color' color, o.details->>'qty' qty
FROM
      orders o
JOIN
      product p on p.id=(o.details->>'product_id')::integer
WHERE
      details->>'qty' > '5';
```

| | name | color | qty |
|---|---|---|---|
| 1 | Polo Yaka Tişört | mavi | 6 |
| 2 | V Yaka Tişört | beyaz | 6 |
| 3 | Polo Yaka Tişört | mavi | 8 |
| 4 | V Yaka Tişört | siyah | 8 |
| 5 | Polo Yaka Tişört | beyaz | 7 |
| 6 | V Yaka Tişört | siyah | 7 |
| 7 | Bisiklet Yaka Tişört | mavi | 9 |
| 8 | Bisiklet Yaka Tişört | siyah | 6 |
| 9 | Bisiklet Yaka Tişört | siyah | 8 |
| 10 | V Yaka Tişört | siyah | 8 |

# JSON/JSONB'de Index Kullanımı

https://www.postgresql.org/docs/current/datatype-json.html#JSON-INDEXING

```
CREATE INDEX idx_1 on orders using  GIN (details);                    -- ?, ?&, ?|, @>
CREATE INDEX idx_2 on orders using  GIN (details jsonb_path_ops);     -- @>
CREATE INDEX idx_3 on orders using btree((details->'color'));         -- =, >, <
CREATE INDEX idx_4 on orders using btree((details->>'qty'));              -- =, >, <
```

# JSON/JSONB'de Index Kullanımı

```
schemaname | tablename | table_disc_size |  index | index_disc_size
-----------+-----------+-----------------+--------+----------------
public     | orders    |        19283968 |        |               0
public     | orders    |               0 | idx_1  |        22831104
public     | orders    |               0 | idx_2  |        12517376
public     | orders    |               0 | idx_3  |         1499136
public     | orders    |               0 | idx_4  |          344064
```

```
select * from orders where details ? 'color';                    --idx_1
select * from orders where details @> '{"color": "siyah"}';      --idx_2
select * from orders where details->'color' = '"siyah"';         --idx_3
select * from orders where details->>'qty'  > '5';               --idx_4
```

# JSON Functions

https://www.postgresql.org/docs/16/functions-json.html

- Creation Functions
- Testing Functions (16+ version)
- Processing Functions
- Aggregate Functions

# JSON Functions – Creation Functions

```
SELECT to_jsonb(i) FROM pg_stat_activity i;
```

| | to_jsonb |
|---|---|
| 1 | {"pid": 1470, "datid": null, "query": "", "state": null, "datname": null, |
| 2 | {"pid": 1472, "datid": null, "query": "", "state": null, "datname": null, |
| 3 | {"pid": 7734, "datid": "1216850", "query": "select to_jsonb(i) from pg_sta |

- to_jsonb()
- jsonb_build_array()
- jsonb_build_object()
- jsonb_object()
- row_to_json()

```
SELECT id,
       json_build_object('id', id, 'name', name) AS data
FROM product;
```

| | id | data |
|---|---|---|
| 1 | 101 | {"id" : 101, "name" : "V Yaka Tişört"} |
| 2 | 102 | {"id" : 102, "name" : "Polo Yaka Tişört"} |
| 3 | 103 | {"id" : 103, "name" : "Bisiklet Yaka Tişört"} |

# JSON Functions – Testing Functions (16+ version)

```
SELECT js,
 js IS JSON "json?",
 js IS JSON SCALAR "scalar?",
 js IS JSON OBJECT "object?",
 js IS JSON ARRAY "array?"
FROM (VALUES
      ('123'), ('"abc"'), ('{"a": "b"}'),
      ('[1,2]'),('abc'))
foo(js);
```

| | js | json? | scalar? | object? | array? |
|---|---|---|---|---|---|
| 1 | 123 | • true | • true | false | false |
| 2 | "abc" | • true | • true | false | false |
| 3 | {"a": "b"} | • true | false | • true | false |
| 4 | [1,2] | • true | false | false | • true |
| 5 | abc | false | false | false | false |

# JSON Functions – Processing Functions

```
SELECT
      id,key,value
FROM orders, jsonb_each_text(orders.details);
```

| | id | key | value |
|---|---|---|---|
| 1 | 15606 | qty | 9 |
| 2 | 15606 | name | Mcclure Petersen |
| 3 | 15606 | note | deserunt Lorem sint excepteur |
| 4 | 15606 | color | siyah |
| 5 | 15606 | phone | +90 (963) 585-2269 |
| 6 | 15606 | amount | 587 |
| 7 | 15606 | address | 352 Oceanic Avenue, Guthrie, New Mexico, 3637 |
| 8 | 15606 | is_sent | false |
| 9 | 15606 | order_date | 2024-02-13T02:56:16 -03:00 |
| 10 | 15606 | product_id | 101 |
| 11 | 15606 | product_name | tisort 1 |
| 12 | 15607 | qty | 4 |
| 13 | 15607 | name | Petra Prince |
| 14 | 15607 | note | ad excepteur aute dolor |
| 15 | 15607 | color | beyaz |
| 16 | 15607 | phone | +90 (997) 512-2468 |
| 17 | 15607 | amount | 555 |
| 18 | 15607 | address | 148 Elliott Place, Kent, Florida, 6572 |

- jsonb_path_query()
- jsonb_path_query_array()
- jsonb_path_query_first()
- jsonb_path_exists()
- jsonb_path_match()
- jsonb_extract_path()
- jsonb_extract_path_text()
- jsonb_insert()
- jsonb_set()
- jsonb_strip_nulls()
- jsonb_array_length()
- jsonb_array_elements()
- jsonb_array_elements_text()
- jsonb_each()
- jsonb_each_text()
- jsonb_object_keys()
- jsonb_to_record()

# JSON Functions – Processing Functions

```
SELECT
   jsonb_object_keys(details) keys

FROM orders
GROUP BY keys;
```

| | keys |
|---|---|
| 1 | address |
| 2 | amount |
| 3 | color |
| 4 | is_sent |
| 5 | name |
| 6 | note |
| 7 | order_date |
| 8 | phone |
| 9 | product_id |
| 10 | product_name |
| 11 | qty |

# JSON Functions – Aggregate Functions

```
SELECT jsonb_pretty(
            jsonb_object_agg(customer_id, c.name)
       ) customers
FROM orders o JOIN customers c on o.customer_id = c.id;
```

customers


```
{
    "6": "Customer180",
    "7": "Customer447",
    "8": "Customer81",
    "9": "Customer126",
    "10": "Customer488",
    "11": "Customer530",
    "12": "Customer69",
    "13": "Customer571",
    "14": "Customer719",
    "15": "Customer987",
    "16": "Customer350",
    "17": "Customer263",
    "18": "Customer750",
    "19": "Customer952",
    "20": "Customer118",
    "21": "Customer860",
    "22": "Customer501",
    "23": "Customer541",
```


- json_agg ()
- jsonb_agg ()
- json_objectagg ()
- json_object_agg ()
- jsonb_object_agg ()
- json_object_agg_strict()
- jsonb_object_agg_strict()
- json_object_agg_unique()
- jsonb_object_agg_unique()
- json_arrayagg()
- json_object_agg_unique_strict()
- jsonb_object_agg_unique_strict()



@atifceylan

/in/atifceylan

# Teşekkürler

# Sorular?