

# Veritabanı Yönetim Sistemleri (335)

---

Dr. Öğr. Üyesi Ahmet Arif AYDIN

L16-

PostgreSQL Veri Tipleri

GÜZ -2022

# PostgreSQL Veri Tipleri

---

- NUMERIC
- UUID
- ARRAY
- JSON
- HSTORE
- UDT (user defined types)

# PostgreSQL: Numeric

---

Numeric veri tipi  
bir sayının tam ve ondalıklı kısmı  
tanımlanmasını sağlar.

`NUMERIC(precision, scale)`

# PostgreSQL: Numeric

---

Numeric veri tipi  
bir sayının tam ve ondalıklı kısmı  
tanımlanmasını sağlar.

`NUMERIC(precision, scale)`

```
CREATE TABLE products (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    price NUMERIC(5,2)  
);
```

# PostgreSQL: Numeric

---

Numeric veri tipi  
bir sayının tam ve ondalıklı kısmı  
tanımlanmasını sağlar.

**NUMERIC(precision, scale)**

```
CREATE TABLE products (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(100) NOT NULL,  
    price NUMERIC(5,2)  
);
```

```
INSERT INTO  
    products (name, price)  
VALUES  
    ('Phone', 500.215),  
    ('Tablet', 500.214);
```

<https://www.postgresqltutorial.com/postgresql-numeric/>

# PostgreSQL : UUID

---

<https://tools.ietf.org/html/rfc4122>

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

128bit uzunluğunda uluslararası unique olan bir veri tipidir.

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

128bit uzunluğunda uluslararası unique olan bir veri tipidir.

40e6215d-b5c6-4896-987c-f30f3678f608  
6ecd8c99-4036-403d-bf84-cf8400f67836  
3f333df6-90a4-4fda-8dd3-9485d27cee36

32digit hexadecimal

Dağıtık sistemlerde kullanılmaktadır aynı anda üretilmesi gereken id lerin benzersiz olmasını garanti eder.

# PostgreSQL : UUID

---

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

**CREATE EXTENSION IF NOT EXISTS**

**"uuid-osp";**

Unique id üretilmesini sağlayan modül



# PostgreSQL : UUID

---

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

**CREATE EXTENSION IF NOT EXISTS "uuid-ossf";** Unique id üretilmesini sağlayan modül

```
CREATE TABLE contacts (  
    contact_id uuid DEFAULT uuid_generate_v1(),  
    first_name VARCHAR NOT NULL,  
    last_name VARCHAR NOT NULL,  
    email VARCHAR NOT NULL,  
    phone VARCHAR,  
    PRIMARY KEY (contact_id)  
);
```

Random sayı = UUID

# PostgreSQL : UUID

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

**CREATE EXTENSION IF NOT EXISTS "uuid-ossf";** Unique id üretilmesini sağlayan modül

```
CREATE TABLE contacts (  
    contact_id uuid DEFAULT uuid_generate_v1(),  
    first_name VARCHAR NOT NULL,  
    last_name VARCHAR NOT NULL,  
    email VARCHAR NOT NULL,  
    phone VARCHAR,  
    PRIMARY KEY (contact_id)  
);
```

Random sayı = UUID

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossf";  
SELECT uuid_generate_v1();
```

	uuid_generate_v1 uuid
1	7badad02-39aa-11ec-9cd3-9801a78fa0c1

<https://www.postgresqltutorial.com/postgresql-uuid/>

# PostgreSQL : UUID

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

**CREATE EXTENSION IF NOT EXISTS "uuid-ossf";** Unique id üretilmesini sağlayan modül

```
CREATE TABLE contacts (  
    contact_id uuid DEFAULT uuid_generate_v4(),  
    first_name VARCHAR NOT NULL,  
    last_name VARCHAR NOT NULL,  
    email VARCHAR NOT NULL,  
    phone VARCHAR,  
    PRIMARY KEY (contact_id)  
);
```

Sistemin MAC adresini +  
Zaman (timestamp) +  
Random sayı = UUID

# PostgreSQL : UUID

**UUIDs** (Universal Unique Identifier)

**GUIDs** : Globally Unique Identifier

**CREATE EXTENSION IF NOT EXISTS "uuid-ossf";** Unique id üretilmesini sağlayan modül

```
CREATE TABLE contacts (  
    contact_id uuid DEFAULT uuid_generate_v4(),  
    first_name VARCHAR NOT NULL,  
    last_name VARCHAR NOT NULL,  
    email VARCHAR NOT NULL,  
    phone VARCHAR,  
    PRIMARY KEY (contact_id)  
);
```

Sistemin MAC adresini +  
Zaman (timestamp) +  
Random sayı = UUID

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossf";  
SELECT uuid_generate_v4();
```

	uuid_generate_v4 uuid
1	47c64326-714c-48be-9773-c837335ec44b

<https://www.postgresqltutorial.com/postgresql-uuid/>

# PostgreSQL : ARRAY

---

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

PostgreSQL de bütün veri tiplerinin  
ARRAY karşılığı bulunmaktadır.

Char [], Text [], Date []...

```
ARRAY [  
    '(408)-589-5846',  
    '(408)-589-5555'  
]
```

# PostgreSQL : ARRAY

---

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

PostgreSQL de bütün veri tiplerinin  
ARRAY karşılığı bulunmaktadır.

Char [], Text [], Date []...

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

# PostgreSQL : ARRAY

---

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

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

## PostgreSQL : ARRAY

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

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

	name	phones
▶	John Doe	(408)-589-5846

ARRAY tipi ile eklenen veriye erişirken **index 1** den başlamaktadır



# PostgreSQL : ARRAY

	id integer	name character varying (20)	grade numeric (4,2)	phone text[]
1	7	mehmet ayhan	82.37	{(424)-333-3333,(555)-222-4444}
2	3	ayhan aydin	72.57	{(333)-344-5533,(543)-211-4456}

```
INSERT INTO testarray
VALUES (3, 'ayhan aydin', 72.57,
        ARRAY ['(333)-344-5533', '(543)-211-4456'] ) ;
```

# PostgreSQL : ARRAY

	id integer	name character varying (20)	grade numeric (4,2)	phone text[]
1	7	mehmet ayhan	82.37	{(424)-333-3333,(555)-222-4444}
2	3	ayhan aydin	72.57	{(333)-344-5533,(543)-211-4456}

```
SELECT name, phone[1] FROM testarray;
```

	name character varying (20)	phone text
1	mehmet ayhan	(424)-333-3333
2	ayhan aydin	(333)-344-5533

# PostgreSQL : JSON

- PostgreSQL 9.2 versiyonundan itibaren kullanılmaktadır.
- İki tip kullanılır JSON ve JSONB
- JSON (JavaScript Object Notation)

```
"1248053020379627520": {  
  "created_at": "Thu Apr 09 01:00:02 +0000 2020",  
  "id_str": "1248053020379627520",  
  "full_text": "Facebook is working on “Campus”, a new s|  
  "source": "<a href=\\\"https://mobile.twitter.com\\\" rel='  
  "user_id_str": "337119125",  
  "retweet_count": 150,  
  "favorite_count": 803,  
  "reply_count": 106,  
  "quote_count": 609, // @wongmjane  
  "conversation_id_str": "1248053020379627520",  
  "lang": "en",  
  "self_thread": {  
    "id_str": "1248053020379627520"  
  }  
},
```

Örnek bir tweet

# PostgreSQL : JSON

- Kaydedilecek JSON'ın içerdiği alanlar farklı kolonlar için farklı olabilir ve string olarak kaydedilir.
- JSON içindeki her bir alana direct erişim sağlanır.

JSON formatı veriyi

Key-> Value

(Anahtar-> Deger)

biçiminde kaydeder.

```
"1248053020379627520": {  
  "created_at": "Thu Apr 09 01:00:02 +0000 2020",  
  "id_str": "1248053020379627520",  
  "full_text": "Facebook is working on “Campus”, a new s  
  "source": "<a href=\"https://mobile.twitter.com\" rel='  
  "user_id_str": "337119125",  
  "retweet_count": 150,  
  "favorite_count": 803,  
  "reply_count": 106,  
  "quote_count": 609, // @wongmjane  
  "conversation_id_str": "1248053020379627520",  
  "lang": "en",  
  "self_thread": {  
    "id_str": "1248053020379627520"  
  }  
},
```

# PostgreSQL : JSON

---

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

```
INSERT INTO orders (info)  
VALUES  
( '  
    { "customer": "John Doe",  
      "items": {"product": "water"  
                , "qty": 6}  
    }  
,  
);
```

# PostgreSQL : JSON

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

```
INSERT INTO orders (info)  
VALUES  
( '  
    { "customer": "John Doe",  
      "items": {"product": "water"  
                , "qty": 6}  
    }  
,  
);
```

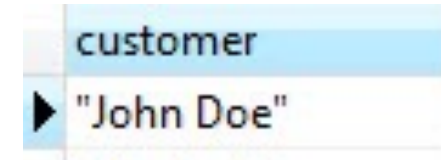
```
SELECT  
info -> 'customer'  
AS customer  
FROM orders;
```

# PostgreSQL : JSON

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

```
INSERT INTO orders (info)  
VALUES  
( '  
    { "customer": "John Doe",  
      "items": {"product": "water"  
                , "qty": 6}  
    }  
,  
);
```

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



customer
▶ "John Doe"

- -> operatörü json

- ->> operatörü text

formatında sonuc gönderir

# PostgreSQL: HSTORE

---

```
CREATE EXTENSION hstore;
```

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

Hstore veri tipinin kullanılmasını sağlayan modül

Sütun içerisinde bir anahtar → değer yapısı oluşturmayı sağlar.



# PostgreSQL: HSTORE

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

```
INSERT INTO books (title, attr)  
VALUES
```

Hstore

```
( 'PostgreSQL Tutorial',  
  "paperback" => "243",  
  "publisher" => "postgresqltutorial",  
  "language"  => "English",  
  "ISBN-13"   => "978-1449370000",  
  "weight"    => "11.2 ounces" ) ;
```

Key

Value

<http://www.postgresqltutorial.com/postgresql-hstore/>

# PostgreSQL: HSTORE

- Sütun içerisinde bir **anahtar**→ **değer** (key → value) yapısı oluşturmayı sağlar.
- Anahtar ve değer ikilileri **string** olarak saklanır.
- Semi-structured data,
- Sürekli kullanılmayan değerlerin saklanması açısından faydalıdır.
- Yeni alan eklenip silinmesini sağlar .
- Performans açısından tavsiye edilir.

	id [PK] integer	isim character varying	nitelik hstore
1	1	Geek Love: A Novel	"pages"=>"368", "author"=>"Katherine Dunn", "category"=>"fiction"
2	2	Geek Love: A Novel	"yıl"=>"2018", "pages"=>"368", "author"=>"Katherine Dunn", "category"=>"fiction"

<http://www.postgresqltutorial.com/postgresql-hstore/>

- `CREATE DOMAIN` kısıtlamalar (`NOT NULL`, `CHECK`) yardımıyla veri tipi oluşturma imkanı sağlar .

- `CREATE DOMAIN` kısıtlamalar (`NOT NULL`, `CHECK`) yardımıyla veri tipi oluşturma imkanı sağlar .

```
CREATE TABLE mailing_list (  
    id SERIAL PRIMARY KEY,  
    first_name VARCHAR NOT NULL,  
    last_name VARCHAR NOT NULL,  
    email VARCHAR NOT NULL,  
    CHECK (  
        first_name !~ '\s'  
        AND last_name !~ '\s'  
    )  
);
```

- **CREATE DOMAIN** kısıtlamalar (**NOT NULL**, **CHECK**) yardımıyla veri tipi oluşturma imkanı sağlar .

```
CREATE TABLE mailing_list (  
    id SERIAL PRIMARY KEY,  
    first_name VARCHAR NOT NULL,  
    last_name VARCHAR NOT NULL,  
    email VARCHAR NOT NULL,  
    CHECK (  
        first_name !~ '\s'  
        AND last_name !~ '\s'  
    )  
);
```

```
CREATE DOMAIN contact_name AS  
VARCHAR NOT NULL CHECK (value !~ '\s');  
  
CREATE TABLE mailing_list2  
    ( id serial PRIMARY KEY,  
      first_name contact_name,  
      last_name contact_name,  
      email VARCHAR NOT NULL  
    );
```

**ALTER DOMAIN:** domain değiştirilir

**DROP DOMAIN** (domain silinir )

**\dD** kayıtlı domainler listelenir

- `CREATE TYPE` composit (birleştirilmiş) bir tip oluşturma imkanı sağlar ve bir fonksiyonun (stored procedure) geri gönderdiği değer olarak kullanılabilir.

```
CREATE TYPE film_summary AS (  
    film_id INT,  
    title VARCHAR,  
    release_year SMALLINT  
);
```

`ALTER TYPE:` type değiştirilir    `DROP TYPE` (type silinir)    \dT kayıtlı tip ler listelenir

<https://www.postgresqltutorial.com/postgresql-user-defined-data-types/>

- **CREATE TYPE** composit (birleştirilmiş) bir tip oluşturma imkanı sağlar ve bir fonksiyonun (stored procedure) geri gönderdiği değer olarak kullanılabilir.

```
CREATE TYPE film_summary AS (  
    film_id INT,  
    title VARCHAR,  
    release_year SMALLINT  
);
```

### CREATE OR REPLACE FUNCTION

```
get_film_summary (f_id INT)  
    RETURNS film_summary AS  
$$  
SELECT film_id, title, release_year  
FROM film  
WHERE film_id = f_id ;  
$$  
LANGUAGE SQL;
```

**ALTER TYPE:** type değiştirilir    **DROP TYPE** (type silinir)    \dT kayıtlı tip ler listelenir

<https://www.postgresqltutorial.com/postgresql-user-defined-data-types/>

- `CREATE TYPE` composit (birleştirilmiş) bir tip oluşturma imkanı sağlar ve bir fonksiyonun (stored procedure) geri gönderdiği değer olarak kullanılabilir.

```
CREATE TYPE film_summary AS (  
    film_id INT,  
    title VARCHAR,  
    release_year SMALLINT  
);
```

```
SELECT * FROM  
get_film_summary (40);
```

### CREATE OR REPLACE FUNCTION

```
get_film_summary (f_id INT)
```

```
RETURNS film_summary AS
```

```
$$
```

```
SELECT film_id, title, release_year
```

```
FROM film
```

```
WHERE film_id = f_id ;
```

```
$$
```

```
LANGUAGE SQL;
```

film_id	title	release_year
40	Army Flintstones	2006

<https://www.postgresqltutorial.com/postgresql-user-defined-data-types/>



---

Dinlediğiniz için  
Teşekkürler...  
İyi çalışmalar...