



Linux Sistemlerde PostgreSQL Yönetimi

İçindekiler

- ▶ [PostgreSQL Sunucusuna Bağlanma...](#)
- ▶ [Veritabanı İstemcisi / psql...](#)
- ▶ [Temel Veritabanı İşlemleri...](#)
- ▶ [Tablo İşlemleri...](#)
- ▶ [Veri İşlemleri...](#)
- ▶ [Tarih ve Zaman Fonksiyonları...](#)
- ▶ [Metin \(String\) Fonksiyonları...](#)
- ▶ [Kullanıcı Yönetimi...](#)

Debian tabanlı sistemler için repositoryden PostgreSQL kurulumu:

Paket indexlerini güncelle.

```
sudo apt update && sudo apt upgrade -y
```

PostgreSQL kur.

Debian/Ubuntu resmi depolarında PostgreSQL paketi hazır geliyor:

```
sudo apt install postgresql -y
```

PostgreSQL servisini kontrol et.

Linux'ta kontrol için terminale:

```
sudo systemctl status postgresql
```

Eğer çalışmıyorsa başlatmak için:

```
sudo systemctl start postgresql
```

Sistem açılışında otomatik olarak başlaması için:

```
sudo systemctl enable postgresql
```

PostgreSQL'in veritabanı kümesi (database cluster) dediğimiz şey aslında PostgreSQL'in tüm verilerini, ayarlarını ve iç yapısını tuttuğu bir dizin.

Ana klasörler

- `base/` → Tüm veritabanlarının tabloları burada durur.
Her veritabanı için bir alt klasör vardır. Her table, index, sequence dosya olarak saklanır.
- `global/` → Tüm cluster'a ait global veriler (ör. kullanıcılar, roller, transaction ID'ler).
- `pg_wal/` (eski adı `pg_xlog`) → Write Ahead Log dosyaları; veri bütünlüğünü sağlamak için yapılan değişikliklerin günlükleri.
- `pg_multixact/` → Çoklu transaction bilgileri.
- `pg_tblspc/` → Tablespace'lere (farklı disklere/veri yollarına ayrılan alanlar) sembolik linkler.
- `pg_stat/` → İstatistik bilgileri.
- `pg_logical/` → Mantıksal replikasyon için kullanılan bilgiler.
- `pg_commit_ts/` → Commit timestamp verileri.
- `pg_subtrans/` → Transaction alt-id bilgileri.

Önemli dosyalar

- `PG_VERSION` → Bu kümenin hangi PostgreSQL sürümüne ait olduğunu gösterir (ör. [15](#)).
- `postgresql.conf` → Sunucunun ana yapılandırma dosyası. (Port, shared_buffers, logging vs. ayarlar).
- `pg_hba.conf` → Kimlik doğrulama kuralları (hangi IP'den kim, hangi yöntemle bağlanabilir).
- `pg_ident.conf` → Sistem kullanıcıları ile PostgreSQL kullanıcılarını eşleştirme.
- `postmaster.pid` → Sunucu çalışırken PID (process ID) bilgisini tutar.

PostgreSQL varsayılan veritabanı kümesinin (data cluster) konumu işletim sistemine ve kurulum yöntemine göre değişir.

- Debian / Ubuntu / Pardus dağıtımlarında (apt ile kurulum):

```
/var/lib/postgresql/<version>/main
```

- RedHat / CentOS / Fedora dağıtımlarında (yum/dnf ile kurulum):

```
/var/lib/pgsql/<version>/data
```

- Kaynaktan derlediysen (`make install`): Kurulum sırasında `initdb` çalıştırırken verdigin `-D` parametresine göre belirlenir.

```
initdb -D /usr/local/pgsql/data
```

Not : Kesin konumu öğrenmek için `postgres` kullanıcısındayken terminale `psql -U postgres -c "SHOW data_directory;"` komutu girilir.

PostgreSQL'de veritabanı (DB) ve tablo (nesne) kimliklerini (OID) öğrenmek için:

```
postgres=# SELECT datname, oid FROM pg_database WHERE datname = 'db';
datname   |   oid
-----+-----
db        | 16448
(1 row)
```

-- PostgreSQL'in sistem kataloğu olan pg_database tablosundan bilgi çeker. pg_database tüm veritabanlarının kayıtlarını tutar.
-- /var/lib/postgresql/<version>/main/base/ konumunda ilgili veritabanın oid numarası ile ilgili klasörde veritabanı bilgileri bulunur.

```
postgres=# SELECT relname, oid FROM pg_class WHERE relname = 'tablo';
relname   |   oid
-----+-----
tablo     | 16449
(1 row)
```

-- pg_class adlı sistem kataloğuında sorgulama yapar. pg_class tabloların, görünümelerin, dizinlerin vs. meta verilerini tutar.

PostgreSQL hangi IP'den dinlediğini aşağıdaki komut ile sorgulunabilir!

```
sudo ss -ltnp | grep 5432
```

Not : Bu çıktı LISTEN eden adresleri gösterir. Örneğin: 127.0.0.1:5432 gibi olmalı. Eğer hiç çıkmıyorsa PostgreSQL çalışmıyor demektir.

`postgresql.conf` ayar dosyası

Dosya genelde `/etc/postgresql/<version>/main/postgresql.conf` yada `/var/lib/pgsql/data/postgresql.conf` konumunda bulunur:

`pg_hba.conf`

```
postgres=# \c vt ahmetp
connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed:
FATAL: Peer authentication failed for user "ahmetp"
Previous connection kept
```

Not : Bu hata peer authentication (kimlik doğrulama) seçeneği ile ilgilidir.

PostgreSQL'de pg_hba.conf dosyası bağlantıların nasıl doğrulanacağını belirler. Bu durumda "ahmetp" kullanıcısı için peer yöntemi geçerli.

Eğer peer aktifse: PostgreSQL, sistemde oturum açtığın Linux kullanıcı adı ile PostgreSQL kullanıcı adının aynı olmasını ister.

Linux'ta "ahmet" kullanıcısıyla, ama PostgreSQL'de "ahmetp" ile bağlanmak istiyorsun → eşleşme yok
→ FATAL: Peer authentication failed.

Bağlanabilmek için pg_hba.conf dosyasını düzenlemiz gereklidir.

Dosya genelde bu konumda olur:

```
/etc/postgresql/<sürüm>/main/pg_hba.conf
```

İçinde şuna benzer satır vardır:

```
local      all      all      peer
```

Bunu şu şekilde değiştir:

```
local      all      all      md5
```

Sonra PostgreSQL'i yeniden başlat:

```
sudo systemctl restart postgresql
```

Terminalden PostgreSQL sunucusuna bağlanmak için:

[Başa Dön...](#)

- `ahmet@pardus:~$ sudo su` Komutu ile root kullanıcısına geçilir.
- `root@pardus:~# su - postgres` Komutu ile postgres kullanıcısına geçilir.
- `postgres@pardus:~$ psql` Komutu ile PostgreSQL sunucusuna bağlanılır.

```
postgres@pardus:~$ psql  
Password for user postgres:  
psql (15.14 (Debian 15.14-0+deb12u1))  
Type "help" for help.  
  
postgres=#
```

Yada PostgreSQL oturumuna kısayoldan bağlanmak için:

```
ahmet@pardus:~$ sudo -u postgres psql
```

Not : PostgreSQL kurulunca varsayılan olarak "postgres" adında bir kullanıcı ve bu kullanıcıya ait "postgres" adında yeni bir veritabanı geliyor.

Veritabanı İstemcisi / psql

[Başa Dön...](#)

PostgreSQL sunucu interaktif terminal istemcisidir. PostgreSQL sunucuda sorgu çalışma, sorgu sonuçlarını görüntüleme, kabuk parametreleri ile dosya veya komut gönderme, betik içerisinde kullanarak otomatik işlemler yapabilir.

Genel Kullanımı

```
psql [seçenekler...] [veritabanı[kullanıcı]]
```

psql komutu için kullanılan parametreler:

Parametre	Açıklama	Örnek Kullanım
-h	Bağlanılacak sunucunun hostname/IP adresi	<code>psql -h 192.168.1.10</code>
-p	PostgreSQL port numarası (varsayılan: 5432)	<code>psql -p 5432</code>
-U	Bağlanılacak kullanıcı adı	<code>psql -U postgres</code>
-d	Bağlanılacak veritabanı adı	<code>psql -d testdb</code>
-W	Parolayı girmeye zorlar	<code>psql -U user -W</code>
-f	Bir SQL dosyasını çalıştırır	<code>psql -d db -f script.sql</code>
-c	Tek bir SQL komutu çalıştırır	<code>psql -d db -c "SELECT * FROM users;"</code>
-v	Değişken tanımlama	<code>psql -v var=123 -f script.sql</code>

Parametre	Açıklama	Örnek Kullanım
<code>-X</code>	psql başlangıç dosyası (.psqlrc) yüklenmesin	<code>psql -X</code>
<code>-A</code>	Hizalamayı kapatır (alignment off)	<code>psql -A -c "SELECT * FROM t"</code>
<code>-t</code>	Sadece satırları gösterir, başlık/format yok	<code>psql -t -c "SELECT now()"</code>
<code>-o</code>	Komut çıktısını dosyaya yazdırır	<code>psql -U postgres -d postgres -o sonuc.txt -c "SELECT * FROM ogrenciler;"</code>
<code>--help veya -?</code>	Yardım ekranı	<code>psql --help veya psql -?</code>
<code>--version veya -V</code>	Sürüm bilgisini gösterir	<code>psql --version veya psql -V</code>

Kullanıcı/parola ile TCP üzerinden veritabanına bağlanma:

```
$ psql -h 127.0.0.1 -U user -W -d db_name
Password for user user:
psql (11.5)
Type "help" for help.

db_name=>
```

Etkileşimli (interaktif) kabuk kullanma:

```
psql (11.5)
Type "help" for help.

postgres=# \c db_name
You are now connected to database "db_name" as user "postgres".
db_name=# SELECT * FROM table_name;
```

Etkileşimsiz kabuk kullanma (dışardan komut yollama):

```
$ psql -U user -c 'SELECT * FROM table_name;' db_name
```

Çıktıyı dosyaya kaydetme:

```
$ psql -U user -c 'SELECT * FROM table_name;' db_name > sonuc
```

Komut çıktısını kullanma (pipe):

```
$ echo '\c db_name \\ SELECT * FROM table_name;' | psql
```

Dosyayı girdi olarak kullanma:

```
$ psql -U user db_name < sorgu.sql
```

Öntanımlı olarak sql sorgularının çıktıları sql biçiminde gelir psql üzerinden csv biçiminde çıktı almak için:

```
$ psql -U user -d db_name -A -F"," -c "select * from table_name;" > dosya.csv
```

psql istemci temel komutları:

\l	Veritabanlarını listeleme	\q	Çıkış
\c	Belirtilen veritabanına geçme	\help (?)	Yardım
\dt	Tabloları listeleme	\copyright	Lisans bilgileri
\dT	Veri tiplerini listeleme	\conninfo	Sunucu bağlantı bilgileri
\du (\dg)	Veritabanı rol/kullanıcı listeleme	\password	Rol parolası belirleme
\dx	Yüklü olan eklentileri listeleme	\encoding	Tanımlı olan karakter kodlaması
\dn	Mevcut şemaları listeleme	\s	Geçmiş komutları listeleme

Temel Veritabanı İşlemleri

[Başa Dön...](#)

Mevcut veritabanlarını listeleme:

```
postgres=# \l
          List of databases
   Name    |  Owner   | Enc. | Collate |      Ctype      | Access privileges
-----+-----+-----+-----+-----+-----+
postgres | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 |
template0 | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | =c/postgres + postgres=CTc/postgres
template1 | postgres | UTF8 | en_US.UTF-8 | en_US.UTF-8 | =c/postgres + postgres=CTc/postgres
(3 rows)
```

Yeni bir veritabanı oluşturma:

```
postgres=# CREATE DATABASE db_name;
CREATE DATABASE
```

- `\c db_name` : Diğer veritabana geçiş için kullanılır.

- `\l+` : Mevcut veritabanlarının size, tablespace ve description alanlarınında listeler.
- `\i dosya` : PostgreSQL sunucusuna bağlandığı konumda bulunan script dosyasını çalıştırır.

Sahip belirterek veritabanı oluşturma:

```
postgres=# CREATE DATABASE db_name OWNER user;  
CREATE DATABASE
```

```
postgres=# CREATE DATABASE db_name  
WITH  
OWNER = postgres  
TEMPLATE = template0  
ENCODING = 'UTF8'  
LC_COLLATE = 'C'  
LC_CTYPE = 'C'  
CONNECTION LIMIT = 20;
```

Veritabanı sahipliğini değiştirmek için:

```
postgres=# ALTER DATABASE db_name OWNER TO user;  
ALTER DATABASE
```

Veritabanının ismini değiştirmek için:

```
postgres=# ALTER DATABASE db_name RENAME TO new_db_name;  
ALTER DATABASE
```

Veritabanını silmek için:

```
postgres=# DROP DATABASE db_name;  
DROP DATABASE
```

`SELECT datname FROM pg_database;` : Sistemdeki mevcut veritabanlarını listeleme sorgusu.

`SELECT username,usesysid FROM pg_user;` : Sistemdeki kullanıcıları ve id bilgileri listelenir.

`SELECT * FROM pg_stat_activity WHERE datname='postgres';` : Adı verilen veritabanına bağlı connectionları listeler.

Tablo İşlemleri

[↑ Başa Dön...](#)

Bir veritabanı içinde yeni bir tablo oluşturma:

```
postgres=# CREATE TABLE personel (
    ad          varchar(40),
    soyad       varchar(40),
    kidem      int,
    uid        int PRIMARY KEY
);
CREATE TABLE
```

Tabloları listeleme:

```
postgres=# \dt
           List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+
 public | personel | table | postgres
(1 row)
```

Tablonun ismini değiştirmek için:

```
postgres=# ALTER TABLE tablo_adi RENAME TO yeni_tablo_adi;
ALTER TABLE
```

Tablo silme:

```
postgres=# DROP TABLE table_name;
DROP TABLE
```

Tablo sahipliğini değiştirmek için:

```
postgres=# CREATE USER yildirim;
CREATE ROLE
postgres=# ALTER TABLE personel OWNER TO yildirim;
ALTER TABLE
postgres=# \dt
           List of relations
 Schema |   Name   | Type  | Owner
-----+-----+-----+
 public | personel | table | yildirim
```

Not : PostgreSQL'de bir tablo sahibini tablo oluşturulmadan belirlemek mümkün değildir. Tablo, onu oluşturan kullanıcıya aittir.

Tablo yapısını gösterme:

```
postgres=# \d personnel
      Table "public.personnel"
  Column |          Type          | Modifiers
-----+-----+-----+
  ad    | character varying(40) |
  soyad | character varying(40) |
  kidem | integer            |
  uid   | integer             | not null
Indexes:
"personnel_pkey" PRIMARY KEY, btree (uid)
```

Tabloyu düzenleme: Yeni sütun ekleme:

```
postgres=# ALTER TABLE public.personnel
ADD COLUMN yas INT;
```

Tabloyu düzenleme: Bir sütunun tipini değiştirme:

```
postgres=# ALTER TABLE public.personnel
ALTER COLUMN ad TYPE character varying (50);
```

Tabloyu düzenleme: Bir sütun silme:

```
postgres=# ALTER TABLE public.personnel
DROP COLUMN kidem;
```

Tabloyu düzenleme: Bir sütunun adını değiştirme:

```
postgres=# ALTER TABLE tablo_adi
RENAME COLUMN eski_isim TO yeni_isim;
```

Veri İşlemleri

 [Başa Dön...](#)

Tabloya bir satır ekleme:

```
postgres=# INSERT INTO personnel VALUES('John', 'Doe', 5, 01);
INSERT 0 1

-- Sadece belirli kolonlar için ekleme yapılacak ise:
postgres=# INSERT INTO personnel(ad, soyad) VALUES('John', 'Doe');
INSERT 0 1
```

Tabloya birden fazla satır ekleme:

```
postgres=# INSERT INTO personel VALUES
    ('Jane', 'Doe', 1, 02),
    ('Richard', 'Roe', 3, 03),
    ('Fred', 'Bloggs', 7, 04),
    ('Juan', 'Perez', 11, 05);
INSERT 0 4
```

Satır sorgulama:

```
postgres=# SELECT * FROM personel;
   ad    | soyad   | kidem | uid
-----+-----+-----+-----
 John  | Doe     |      5 |    1
 Jane  | Doe     |      1 |    2
 Richard | Roe    |      3 |    3
 Fred   | Bloggs  |      7 |    4
 Juan   | Perez   |     11 |    5
(5 rows)
postgres=# SELECT ad, soyad FROM personel;
   ad    | soyad
-----+
 John  | Doe
 Jane  | Doe
 Richard | Roe
 Fred   | Bloggs
 Juan   | Perez
(5 rows)
```

Sütun Güncelleme:

```
postgres=# UPDATE ogrenciler SET email='ersin-dari@yahoo.com' WHERE id=7;
UPDATE 1
```

Not : `WHERE` ile koşul belirtmezsek `ogrenciler` tablosundaki bütün `email` sütunları güncellenir.

Satır silme:

```
postgres=# DELETE FROM ogrenciler WHERE id=7;
DELETE 1
```

Not : `WHERE` ile koşul belirtmezsek `ogrenciler` tablosundaki bütün kayıtlar silinir.

İndeks İşlemleri

PostgreSQL Primary Key ya da Unique Constraint için indeksi otomatik olarak oluşturur.

```

postgres=# \d personnel
      Table "public.personnel"
   Column |          Type          | Modifiers
-----+-----+-----+
    ad   | character varying(40) |
  soyad | character varying(40) |
   kidem | integer            |
    uid  | integer             | not null
Indexes:
"personnel_pkey" PRIMARY KEY, btree (uid)

```

Standart indeks oluşturma:

```

postgres=# CREATE INDEX soyad_idx ON personnel (soyad);
CREATE INDEX

```

Referans Verme İşlemleri

Bir tablodan başka bir tabloya o tablonun Primary Key alanı aracılığıyla referans verilir.

```

pagila=# CREATE TABLE items
(
  code int PRIMARY KEY,
  name text,
  price numeric(10,2)
);
CREATE TABLE

```

```

pagila=# CREATE TABLE orders
(
  no int PRIMARY KEY,
  date date,
  amount numeric,
  item_code int REFERENCES items (code)
);
CREATE TABLE

```

Referans veren tablo:

```

postgres=# \d orders
      Table "public.orders"
 Column | Type   | Modifiers
-----+-----+
 no     | integer | not null
 date   | date    |
 amount | numeric |
 item_code | integer |

Indexes:
 "orders_pkey" PRIMARY KEY, btree (no)
Foreign-key constraints:
 "orders_item_code_fkey" FOREIGN KEY (item_code) REFERENCES items(code)

```

Çalışma Zamanı Parametreleri

`SHOW` ile belirli bir çalışma parametresinin bilgisi alınabilir:

```

postgres=# SHOW DateStyle;
DateStyle
-----
ISO, MDY
(1 row)

```

Tüm parametrelerin listesine ve bilgisine erişmek için:

```

postgres=# SHOW ALL;
          name       | setting | description
-----+-----+-----+
allow_system_table_mods | off      | Allows modifications of the structure
                           of ...
.
.
.
xmloption              | content | Sets whether XML data is implicit
                           parsing ...
zero_damaged_pages     | off      | Continues processing past damaged
                           page headers.
(290 rows)

```

`SET` komutu ile bir parametre çalışma zamanında değiştirilebilir:

```

postgres=# SET timezone='Europe/Rome';
SET

```

`SET` komutu ile değiştirilen parametre sadece o oturumda geçerlidir, oturum kapandığında geçerliliğini kaybeder. Parametrelerin kalıcı olması için `postgresql.conf` dosyası üzerinde ayarlama yapılmalıdır.

PostgreSQL Tarih ve Zaman Fonksiyonları

[Başa Dön...](#)

♦ Zaman Bilgisi Alma

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
NOW()	Şu anki tarih ve saatı döner (timestamp with time zone)	SELECT NOW();	2025-10-20 22:41:32.123+03
CURRENT_TIMESTAMP	NOW() ile aynıdır	SELECT CURRENT_TIMESTAMP;	2025-10-20 22:41:32.123+03
CURRENT_DATE	Sadece tarihi döner	SELECT CURRENT_DATE;	2025-10-20
CURRENT_TIME	Sadece saatı döner	SELECT CURRENT_TIME;	22:41:32.123+03
LOCALTIMESTAMP	Saat dilimi olmadan döner	SELECT LOCALTIMESTAMP;	2025-10-20 22:41:32.123

♦ Tarih Formatlama (TO_CHAR)

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
TO_CHAR(tarih, 'YYYY-MM-DD')	Tarihi belirtilen biçimde çevirir	SELECT TO_CHAR(NOW(), 'YYYY-MM-DD');	2025-10-20
TO_CHAR(tarih, 'DD Mon YYYY')	Ay adını içerir	SELECT TO_CHAR(NOW(), 'DD Mon YYYY');	20 Oct 2025
TO_CHAR(tarih, 'HH24:MI:SS')	Saat biçimini	SELECT TO_CHAR(NOW(), 'HH24:MI:SS');	22:45:30
TO_CHAR(tarih, 'Day, DD Mon YYYY')	Gün + tarih	SELECT TO_CHAR(NOW(), 'Day, DD Mon YYYY');	Monday, 20 Oct 2025

♦ Tarih Dönüşümü

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
TO_DATE(string, format)	String → Date	SELECT TO_DATE('2025-01-15', 'YYYY-MM-DD');	2025-01-15
TO_TIMESTAMP(string, format)	String → Timestamp	SELECT TO_TIMESTAMP('2025-01-15 10:30', 'YYYY-MM-DD HH24:MI');	2025-01-15 10:30:00

♦ Tarih Üzerinde İşlem (INTERVAL)

İşlem	Kullanım	Açıklama
Gün ekleme	SELECT NOW() + INTERVAL '5 days';	5 gün sonrası verir
Ay çıkarma	SELECT NOW() - INTERVAL '2 months';	2 ay öncesini verir
Saat ekleme	SELECT NOW() + INTERVAL '3 hours';	3 saat ekler
Dakika ekleme	SELECT NOW() + INTERVAL '30 minutes';	30 dakika ekler
Yıl çıkarma	SELECT NOW() - INTERVAL '1 year';	1 yıl önceki zamanı verir

♦ Tarih Parçalama (EXTRACT, DATE_PART)

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
EXTRACT(YEAR FROM tarih)	Yıl bilgisi	SELECT EXTRACT(YEAR FROM NOW());	2025
EXTRACT(MONTH FROM tarih)	Ay bilgisi	SELECT EXTRACT(MONTH FROM NOW());	10
EXTRACT(DAY FROM tarih)	Gün bilgisi	SELECT EXTRACT(DAY FROM NOW());	20
EXTRACT(DOW FROM tarih)	Haftanın günü (0=Pazar)	SELECT EXTRACT(DOW FROM NOW());	1
DATE_PART('hour', tarih)	Saat bilgisi	SELECT DATE_PART('hour', NOW());	22

♦ Tarih Farkı Hesaplama

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
AGE(t1, t2)	İki tarih arasındaki fark	SELECT AGE('2025-10-20', '2020-10-20');	5 years
AGE(NOW(), dogum_tarihi)	Yaş hesaplama örneği	SELECT AGE(NOW(), '2000-06-15');	25 years 4 mons 5 days

♦ Epoch (Unix Timestamp)

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
EXTRACT(EPOCH FROM NOW())	Şu anki zamanı saniye cinsinden verir	SELECT EXTRACT(EPOCH FROM NOW());	1730050000
TO_TIMESTAMP(epoch)	Epoch → Timestamp	SELECT TO_TIMESTAMP(1730050000);	2025-10-20 22:45:00+03

♦ Örnekler

```
-- 1. Yarının tarihi
SELECT CURRENT_DATE + INTERVAL '1 day';

-- 2. 10 gün sonra saat 12:00
SELECT CURRENT_DATE + INTERVAL '10 days' + TIME '12:00';

-- 3. Haftanın gününü öğren
SELECT TO_CHAR(NOW(), 'Day');

-- 4. Bugün Pazartesi mi?
SELECT EXTRACT(DOW FROM CURRENT_DATE) = 1;

-- 5. Ayın kaçinci haftası
SELECT EXTRACT(WEEK FROM CURRENT_DATE);

-----
SELECT *
FROM tablo_adi
```

```

WHERE EXTRACT(YEAR FROM doğum_tarihi) = 1990;
-- Bu sorgu, doğum tarihi 1990 olan tüm kayıtları getirir.

```

```

-----
SELECT *
FROM tablo_adi
WHERE EXTRACT(YEAR FROM doğum_tarihi) IN (1985, 1990, 1995);
-- Bu sorgu, doğum tarihi 1985, 1990 veya 1995 olan tüm kayıtları getirir.

```

```

-----
SELECT *
FROM tablo_adi
WHERE doğum_tarihi BETWEEN '1980-01-01' AND '1990-12-31';
-- Bu sorgu, doğum tarihi 1980 ile 1990 yılları arasında olan tüm kayıtları getirir.
/*
AGE() Fonksiyonu: Eğer doğum tarihinden yaş hesaplamak isterseniz, AGE() fonksiyonunu kullanabilirsiniz. Bu fonksiyon, iki tarih arasındaki farkı hesaplar.
*/

```

PostgreSQL Metin (String) Fonksiyonları

[Başa Dön...](#)

♦ Temel Fonksiyonlar

Fonksiyon	Açıklama	Örnek	Çıktı
LENGTH(text)	Metindeki karakter sayısını döner (byte sayabilir).	SELECT LENGTH('Ahmet');	5
CHAR_LENGTH(text) veya CHARACTER_LENGTH(text)	Gerçek karakter sayısını döner (UTF8 güvenli).	SELECT CHAR_LENGTH('Çağrı');	5
LOWER(text)	Tüm harfleri küçük yapar.	SELECT LOWER('AHMET');	ahmet
UPPER(text)	Tüm harfleri büyük yapar.	SELECT UPPER('ahmet');	AHMET
INITCAP(text)	Her kelimenin ilk harfini büyük yapar.	SELECT INITCAP('ahmet bedir');	Ahmet Bedir
REVERSE(text)	Metni ters çevirir.	SELECT REVERSE('Ahmet');	temhA

♦ Alt String Alma

Fonksiyon	Açıklama	Örnek	Çıktı
SUBSTRING(text FROM start FOR count)	Belirtilen aralıktaki karakterleri döner.	SELECT SUBSTRING('Ahmet' FROM 2 FOR 3);	hme
LEFT(text, n)	Soldan n karakter döner.	SELECT LEFT('Ahmet', 2);	Ah
RIGHT(text, n)	Sağdan n karakter döner.	SELECT RIGHT('Ahmet', 2);	et

♦ Metin Birleştirme

Fonksiyon	Açıklama	Örnek	Çıktı
<code>CONCAT(a, b, c...)</code>	Değerleri birleştirir.	<code>SELECT CONCAT('Postgre', 'SQL');</code>	PostgreSQL
<code>CONCAT_WS(delimiter, a, b, c...)</code>	Araya ayraç koyarak birleştirir.	<code>SELECT CONCAT_WS('-', 'Ahmet', 'Bedir');</code>	Ahmet-Bedir
<code>a b</code>	Metin birleştirme operatörü.		

♦ Arama ve Karşılaştırma

Fonksiyon / Operatör	Açıklama	Örnek	Çıktı
<code>POSITION(sub IN text)</code>	Alt dizinin pozisyonunu döner.	<code>SELECT POSITION('m' IN 'Ahmet');</code>	3
<code>LIKE</code>	Desene göre eşleşme	<code>SELECT 'Ahmet' LIKE 'Ah%';</code>	true
<code>ILIKE</code>	Harf duyarsız eşleşme	<code>SELECT 'ahmet' ILIKE 'AH%';</code>	true
<code>~</code>	Regex (büyük/küçük duyarlı)	<code>SELECT 'ahmet' ~ '^[a-z]+\$';</code>	true
<code>~*</code>	Regex (büyük/küçük duyarsız)	<code>SELECT 'Ahmet' ~* 'ahmet';</code>	true

♦ Metin Değiştirme ve Temizleme

Fonksiyon	Açıklama	Örnek	Çıktı
<code>REPLACE(text, from, to)</code>	Metin içindeki parçayı değiştirir.	<code>SELECT REPLACE('ahmet', 'a', 'o');</code>	ohmet
<code>TRIM(text)</code>	Baştaki ve sondaki boşlukları temizler.	<code>SELECT TRIM(' ahmet ');</code>	ahmet
<code>LTRIM(text)</code>	Sadece baştaki boşlukları siler.	<code>SELECT LTRIM(' ahmet');</code>	ahmet
<code>RTRIM(text)</code>	Sadece sondaki boşlukları siler.	<code>SELECT RTRIM('ahmet ');</code>	ahmet
<code>BTRIM(text, chars)</code>	Belirtilen karakterleri baştan ve sondan siler.	<code>SELECT BTRIM('xxahmetxx', 'x');</code>	ahmet

♦ Biçimlendirme ve Dönüşüm

Fonksiyon	Açıklama	Örnek	Çıktı
<code>TO_CHAR(value, format)</code>	Tarih veya sayıyı biçimlendirir.	<code>SELECT TO_CHAR(NOW(), 'YYYY-MM-DD');</code>	2025-10-20
<code>CAST(value AS TEXT)</code>	Veriyi metne dönüştürür.	<code>SELECT CAST(123 AS TEXT);</code>	'123'
<code>CAST(value AS INTEGER)</code>	Metni sayıya dönüştürür.	<code>SELECT CAST('456' AS INTEGER);</code>	456

♦ Faydalı Ek Fonksiyonlar

Fonksiyon	Açıklama	Örnek	Cıktı
SPLIT_PART(text, delimiter, field)	Belirtilen ayraçtan sonra n. parçayı döner.	SELECT SPLIT_PART('ahmet@bedir.com', '@', 1);	ahmet
REPEAT(text, number)	Metni belirtilen kadar tekrarlar.	SELECT REPEAT('Ha', 3);	HaHaHa
LPAD(text, length, fill)	Soldan belirtilen karakterle doldurur.	SELECT LPAD('7', 3, '0');	007
RPAD(text, length, fill)	Sağdan belirtilen karakterle doldurur.	SELECT RPAD('7', 3, '0');	700

♦ Örnek Tablo: ogrenciler

```
CREATE TABLE ogrenciler (
    id SERIAL PRIMARY KEY,
    ad VARCHAR(50),
    soyad VARCHAR(50),
    email VARCHAR(100),
    dtarihi DATE
);

INSERT INTO ogrenciler (ad, soyad, email, dtarihi) VALUES
('Ahmet', 'Bedir', 'ahmet.bedir@example.com', '1988-08-30'),
('Mehmet', 'Kaya', 'mehmet.kaya@example.com', '2013-01-22'),
('Ali', 'Çelik', 'ali.celik@example.com', '1979-04-30');
```

♦ Uzunluk ve Biçim Fonksiyonları

```
SELECT ad, LENGTH(ad) AS karakter_sayisi, UPPER(soyad) AS buyuk_harf
FROM ogrenciler;
```

ad	karakter_sayisi	buyuk_harf
Ahmet	5	BEDIR
Mehmet	6	KAYA
Ali	3	ÇELIK

♦ Birleştirme (Concatenation)

```
SELECT ad || ' ' || soyad AS tam_ad
FROM ogrenciler;
```

tam_ad
Ahmet Bedir
Mehmet Kaya

tam_ad

Ali Çelik

♦ Belirli Kısmı Alma

```
SELECT ad, SUBSTRING(email FROM 1 FOR 5) AS mail_parcasi  
FROM ogrenciler;
```

ad	mail_parcasi
Ahmet	ahmet
Mehmet	mehme
Ali	ali.c

♦ Ayraçla Bölme (SPLIT_PART)

```
SELECT ad, SPLIT_PART(email, '@', 1) AS kullanici_adi  
FROM ogrenciler;
```

ad	kullanici_adi
Ahmet	ahmet.bedir
Mehmet	mehmet.kaya
Ali	ali.celik

♦ Değiştirme (REPLACE)

```
SELECT ad, REPLACE(email, '.com', '.org') AS yeni_email  
FROM ogrenciler;
```

ad	yeni_email
Ahmet	ahmet.bedir@example.org
Mehmet	mehmet.kaya@example.org
Ali	ali.celik@example.org

♦ Trim ve Temizleme

```
SELECT TRIM('    ' || ad || '    ') AS temiz_ad  
FROM ogrenciler;
```

temiz_ad

Ahmet

Mehmet

Ali

♦ Pad (Soldan veya Sağdan Doldurma)

```
SELECT ad, LPAD(id::text, 3, '0') AS kod  
FROM ogrenciler;
```

ad	kod
Ahmet	001
Mehmet	002
Ali	003

♦ Küçük / Büyük Harf Dönüşümü

```
SELECT INITCAP(LOWER(ad || ' ' || soyad)) AS duzgun_isim  
FROM ogrenciler;
```

duzgun_isim

Ahmet Bedir

Mehmet Kaya

Ali Çelik

♦ Regex Arama (desen kontrolü)

```
SELECT ad, email  
FROM ogrenciler  
WHERE email ~ '^[a-z]+\.\.';
```

ad	email
Ahmet	ahmet.bedir@example.com
Mehmet	mehmet.kaya@example.com
Ali	ali.celik@example.com

♦ Biçimlendirme (TO_CHAR)

```
SELECT ad, TO_CHAR(NOW(), 'YYYY-MM-DD HH24:MI') AS kayit_tarihi
FROM ogrenciler;
```

ad	kayit_tarihi
Ahmet	2025-10-20 14:37
Mehmet	2025-10-20 14:37
Ali	2025-10-20 14:37

♦ Tarih Biçimlendirme (TO_CHAR)

```
SELECT ad, TO_CHAR(dtarihi, 'DD.MM.YYYY') AS dogum_tarihi FROM ogrenciler;
```

ad	dogum_tarihi
Ahmet	30.08.1988
Mehmet	22.01.2013
Ali	30.04.1979

PostgreSQL Kullanıcı Yönetimi

Başa Dön...

```
postgres=# ALTER USER postgres PASSWORD 'parola';
ALTER ROLE
```

Yukarıdaki komut ile `postgres` süper kullanıcı hesabının parolasını sıfırlamış olursun. Mevcut normal kullanıcıya parola atamak / değiştirmek için `ALTER USER user WITH PASSWORD 'new_password';` komutu kullanılır.

- `\du` : Komutu ile mevcut kullanıcılar listelenir.
- Oturum açıkken kullanıcı değiştirmek için:

```
postgres=# \c db_name user
```

- `CREATE USER new_user;` : Varsayılan olarak login yetkisi olan bir kullanıcı oluşturur.
- `CREATE ROLE new_user;` : Nologin bir kullanıcı oluşturur.

```
postgres=# CREATE ROLE yildirim;
CREATE ROLE
postgres=# CREATE USER bilgem;
CREATE ROLE
postgres=# \du
          List of roles
Role name | Attributes | Member of
-----+-----+-----+
yildirim | Cannot login | {}
bilgem   |           | {}
postgres | Superuser, Create role, Create DB,
          | Replication, Bypass RLS | {}
```

Kullanıcı oluşturulurken özellikde (attribute) belirlenebilinir:

```
postgres=# CREATE ROLE deploy SUPERUSER LOGIN;
CREATE ROLE
```

Kullanılabilecek attribute'lar:

```
LOGIN
SUPERUSER
CREATEDB
CREATEROLE
REPLICATION LOGIN
PASSWORD
```

Ya da sonradan değiştirilir:

```
postgres=# ALTER ROLE deploy NOSUPERUSER CREATEDB;
ALTER ROLE
```

- `CREATE USER new_user WITH PASSWORD 'parola';` : Yeni bir kullanıcı oluşturur ve ona şifre verir.
- `CREATE USER new_user WITH PASSWORD 'parola' CREATEDB;` : Yeni kullanıcı oluşturur ve veritabanı oluşturma yetkisi de verir.
- `CREATE DATABASE db_name OWNER user;` : İsmi verilen kullanıcıya veritabanı oluşturmak için kullanılır.
- `DROP USER user;` : Kullanıcı silmek için kullanılır. Silinmek istenen rol kullanımda ise önce

her bir veritabanında bu rolün sahiplendiği nesneler başka rollere devredilir ya da silinir, sonra kullanıcı silinir.
