



## Linux Sistemlerde PostgreSQL Yönetimi

Güncellemme : 11/2025

---

### İçindekiler

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

## Debian tabanlı sistemler için repositoryden PostgreSQL kurulumu:

Paket indexlerini güncelle.

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

PostgreSQL kurulumu için Debian/Ubuntu resmi depolarında PostgreSQL paketi hazır geliyor:

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

PostgreSQL servisini kontrol etmek 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 Veritabanı Kümesi

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 tablo, 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 kümelenin 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 kümelerinin (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
```

- Kaynakta derlediysen (`make install`) kurulum sırasında `initdb` çalıştırırken verdiği `-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, görünümülerin, 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 değiştirebilirsiniz:

```
local      all      all      md5
```

Sonra PostgreSQL'i yeniden başlatın:

```
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 çalıştırma, sorgu sonuçlarını görüntüleme, kabuk parametreleri ile dosya veya komut gönderme, betik içerisinde kullanarak otomatik işlemler yaptırabilir.

### 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>
-X	psql başlangıç dosyası (.psqlrc) yüklenmesin	<code>psql -X</code>

Parametre	Açıklama	Örnek Kullanım
<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</code> veya <code>-?</code>	Yardım ekranı	<code>psql --help</code> veya <code>psql -?</code>
<code>--version</code> veya <code>-V</code>	Sürüm bilgisini gösterir	<code>psql --version</code> veya <code>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 < soru.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 alanlarında listeler.
- `\i dosya` : PostgreSQL sunucusuna bağlandığın 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 listeleyen sorgusu.

`SELECT usename,usesysid FROM pg_user;` : Sistemdeki kullanıcı adı ve id bilgileri listelenir.

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

## PostgreSQL'de Veri Türleri (Data Types)

[Başa Dön...](#)

### 1) SAYISAL (NUMERIC) TİPLER

Veri Türü	Kapladığı Boyut	Min / Max Değeri	Örnek Kullanım
<b>smallint</b>	2 byte	-32768 → 32767	<code>age smallint</code>
<b>integer (int)</b>	4 byte	-2,147,483,648 → 2,147,483,647	<code>id int</code>
<b>bigint</b>	8 byte	-9,22e18 → 9,22e18	<code>population bigint</code>
<b>decimal / numeric(p,s)</b>	Değişken (yakl. 2 byte / 4 digit)	Hassasiyet sınırsız	<code>price</code> <code>numeric(12,2)</code>
<b>real (kayan noktalı) sayı</b>	4 byte	~6 hane hassasiyet	<code>temperature real</code>
<b>double precision (kayan noktalı) sayı</b>	8 byte	~15 hane hassasiyet	<code>rating double precision</code>

Veri Türü	Kapladığı Boyut	Min / Max Değeri	Örnek Kullanım
<b>serial</b>	4 byte (int)	Otomatik artan tamsayı	<code>id serial</code>
<b>bigserial</b>	8 byte	Daha büyük otomatik artan tamsayı	<code>id bigserial</code>

## 📌 2) METİN (TEXT) TİPLERİ

Veri Türü	Boyut	Max Uzunluk	Örnek
<b>text</b>	Değişken (1B–1GB)	1 GB (yaklaşık)	<code>description text</code>
<b>varchar(n)</b>	Değişken	n karakter	<code>name varchar(255)</code>
<b>char(n)</b>	n byte	n karakter (sabit)	<code>code char(10)</code>
<b>varchar</b> (sınırsız)	Değişken	1 GB	<code>name varchar</code>
<b>citext</b>	Değişken	1 GB	<code>email citext</code> ( <i>büyük/küçük duyarlı</i> )

## 📌 3) BOOLEAN

Tür	Boyut	Açıklama
<b>boolean</b>	1 byte	true / false

Örnek:

```
is_active boolean
```

## 📌 4) TARİH & SAAT TİPLERİ

Veri Türü	Boyut	Aralık	Örnek
<b>date</b>	4 byte	MÖ 4713 – MS 5874897	<code>birthdate date</code>
<b>time</b>	8 byte	00:00 → 24:00	<code>start_at time</code>
<b>time with time zone</b>	12 byte		<code>start_at timetz</code>
<b>timestamp</b>	8 byte	MÖ 4713 – MS 294276	<code>created_at timestamp</code>
<b>timestamptz</b>	8 byte		<code>created_at timestamptz</code>
<b>interval</b>	16 byte	±178 milyon yıl	<code>duration interval</code>

## 📌 5) JSON TİPLERİ

Tür	Boyut	Max	Örnek
json	Değişken	1 GB	data json
jsonb	Değişken	1 GB	meta jsonb

## 📌 6) ARRAY (DİZİ) TİPLERİ

Tür	Boyut	Limit	Örnek
int[], text[], varchar[]	Değişken	Her eleman max 1 GB	tags text[]

Dizi elemanları kendi veri türünün boyutuna bağlıdır.

## 📌 7) UUID

Tür	Boyut	Açıklama	Örnek
uuid	16 byte	Global benzersiz kimlik	id uuid DEFAULT gen_random_uuid()

## 📌 8) PARA TİPİ

Tür	Boyut	Örnek
money	8 byte	amount money

(Tavsiye edilen numeric(12, 2))

## 📌 9) BINARY / BYTEA

Tür	Boyut	Limit	Örnek
bytea	Değişken	1 GB	file bytea

Dosya, resim, video saklamak için.

## 📌 10) ÖZEL (SPECIAL) TİPLER

Tür	Boyut	Açıklama
inet	7-19 byte	IP adresi
cidr	7-19 byte	IP blokları
macaddr	6 byte	MAC adresi

Tür	Boyut	Açıklama
<b>macaddr8</b>	8 byte	
<b>tsvector</b>	Değişken	Full-text search
<b>tsquery</b>	Değişken	Text search
<b>point</b>	16 byte	(x,y)
<b>line</b>	32 byte	Sonsuz çizgi
<b>lseg</b>	32 byte	Çizgi parçası
<b>box</b>	32 byte	Dikdörtgen
<b>circle</b>	24 byte	Daire
<b>polygon</b>	Değişken	Çokgen
<b>enum</b>	4 byte	Sabit değerler

Örnek enum:

```
CREATE TYPE status AS ENUM ('active', 'passive');
```

## ■ 11) XML

Tür	Boyut	Limit
<b>xml</b>	Değişken	1 GB

## ■ 12) Object Identifier (OID) Türleri

Tür	Boyut	Açıklama
<b>oid</b>	4 byte	Sistem nesne ID'si
<b>regclass, regtype, regproc ...</b>	4 byte	Sistem referansları

# 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 olmadan belirlemek mümkün değildir. Tablo, onu oluşturan kullanıcıya aittir.

## Tablo yapısını gösterme:

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

## Tabloyu düzenleme: Yeni sütun ekleme:

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

## Tabloyu düzenleme: Bir sütunun tipini değiştirme:

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

#### Tabloyu düzenleme: Bir sütun silme:

```
postgres=# ALTER TABLE public.personel  
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 personel VALUES('John', 'Doe', 5, 01);  
INSERT 0 1  
  
-- Sadece belirli kolonlar için ekleme yapılacak ise:  
postgres=# INSERT INTO personel(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

 [Başa Dön...](#)

PostgreSQL'de index işlemleri; sorguları hızlandırmak, tablo içindeki belirli kolonlara göre hızlı arama yapabilmek için kullanılır.

Index, bir tablo içinde belirli sütunlara göre arama / filtreleme / sıralama işlemlerini hızlandıran veri yapılarıdır. Bir nevi kitabın arka dizini gibi çalışır.

### 1) Index Oluşturma (CREATE INDEX)

#### Temel kullanım

```
CREATE INDEX idx_adi ON tablo_adi (kolon_adi);
```

#### Örnek:

```
CREATE INDEX idx_users_email ON users (email);
```

 Bu, `users` tablosunda `email` üzerinden aramayı hızlandırır.

### 2) UNIQUE Index

Aynı değerin iki kez girilmesini engeller.

```
CREATE UNIQUE INDEX idx_users_tc ON users (tc_kimlik);
```

### 3) Birden Fazla Kolonlu (Composite) Index

```
CREATE INDEX idx_orders_user_date ON orders (user_id, order_date);
```

💡 Sorgu hem user\_id hem de order\_date içeriyorsa hızlanır.

### 4) Index Silme (DROP INDEX)

```
DROP INDEX idx_adi;
```

Örnek:

```
DROP INDEX idx_users_email;
```

### 5) Indexleri Listeleme

Sadece açıklayıcı yapmak istersen:

```
\d tablo_adi
```

veya

```
SELECT * FROM pg_indexes WHERE tablename = 'users';
```

### 6) Index Çalışıyor mu? — EXPLAIN ANALYZE

Sorgu index kullanıyor mu görmek için:

```
EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'a@b.com';
```

Çıktıda → `Index Scan` yazıyorsa index kullanılıyor demektir.

### 7) En Yaygın Index Türleri

Index Türü	Açıklama	Kullanım Alanı
B-Tree	Varsayılan index	Eşitlik, <, >, ORDER BY
Hash	Sadece eşitlik için	WHERE id = 5
GIN	JSONB, Array	JSON içi arama
GIST	Geometrik, tam metin	Konum / yakınlık
BRIN	Çok büyük (milyonlarca satır), sıralı veriler	Zaman serisi

## 8) JSONB için Index Örneği (GIN)

```
CREATE INDEX idx_products_data ON products USING GIN (data);
```

## 9) Partial (Koşullu) Index

Tablonun tamamı yerine sadece belirli bir kısmında index oluşturur.

```
CREATE INDEX idx_active_users ON users (email)
WHERE active = true;
```

## 10) Index Ne Zaman Kullanılmamalı?

- Tablo çok küçükse (1-2 bin satır)
- Kolon çok fazla tekrar eden değerler içeriyorsa (ör: cinsiyet)
- Sürekli güncellenen kolonlar (index güncelleme maliyeti yüksek)

# Referans Verme İşlemleri

 [Başa Dön...](#)

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)

```

PostgreSQL'de referans verme işlemi, yani FOREIGN KEY (yabancı anahtar) tanımlamak; bir tablodaki bir kolonun başka bir tablodaki PRIMARY KEY/UNIQUE bir kolona bağlı olmasını sağlar. Bu, veri bütünlüğü için çok önemlidir.

## 1) Temel FOREIGN KEY Kullanımı

✓ İki tablo düşünelim:

- **users** (ana tablo)
- **orders** (users tablosunu referanslayan alt tablo)

**users tablosu**

```

CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name TEXT
);

```

**orders tablosu (FOREIGN KEY ile)**

```

CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);

```

- ◆ Burada **orders.user\_id** → **users.id** şeklinde referans verildi.

## 2) FOREIGN KEY Sonradan Ekleme

Eğer tabloyu önceden oluşturduysan:

```

ALTER TABLE orders
ADD CONSTRAINT fk_orders_user
FOREIGN KEY (user_id) REFERENCES users(id);

```

### 3) FOREIGN KEY Silme

```
ALTER TABLE orders
DROP CONSTRAINT fk_orders_user;
```

### 4) ON DELETE / ON UPDATE Kuralları

Referans verilen veride değişiklik veya silme olunca ne yapılacağını belirler.

#### ✓ ON DELETE CASCADE

Ana tablo silinince alt tablodaki ilgili kayıtlar da otomatik silinir.

```
FOREIGN KEY (user_id)
REFERENCES users(id)
ON DELETE CASCADE;
```

#### ✓ ON DELETE SET NULL

Ana tablo silinince alt tablodaki değer NULL olur.

```
FOREIGN KEY (user_id)
REFERENCES users(id)
ON DELETE SET NULL;
```

#### ✓ ON DELETE RESTRICT / NO ACTION

Silme engellenir.

```
ON DELETE RESTRICT;
```

### 5) Composite (Çoklu kolon) FOREIGN KEY

Eğer tabloda iki kolon birlikte PRIMARY KEY ise:

Ana tablo

```
CREATE TABLE cities (
    country_code TEXT,
    city_code TEXT,
    PRIMARY KEY(country_code, city_code)
);
```

Referans veren tablo

```
CREATE TABLE people (
    id SERIAL PRIMARY KEY,
    country_code TEXT,
    city_code TEXT,
    FOREIGN KEY (country_code, city_code)
        REFERENCES cities(country_code, city_code)
);
```

## 6) FOREIGN KEY ile Index İlişkisi

PostgreSQL, referans veren kolonlara otomatik index oluşturmaz.

Örnek:

```
ALTER TABLE orders
ADD FOREIGN KEY (user_id) REFERENCES users(id);
```

💡 Bu durumda `orders.user_id` için index önerilir:

```
CREATE INDEX idx_orders_user_id ON orders(user_id);
```

## 7) Tabloları Listeleme + Foreign Key'leri Görme

```
\d orders
```

veya:

```
SELECT
    tc.table_name,
    kcu.column_name,
    ccu.table_name AS foreign_table,
    ccu.column_name AS foreign_column
FROM
    information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage AS kcu
    ON tc.constraint_name = kcu.constraint_name
JOIN information_schema.constraint_column_usage AS ccu
    ON ccu.constraint_name = tc.constraint_name
WHERE constraint_type = 'FOREIGN KEY';
```

## 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
<code>NOW()</code>	Şu anki tarih ve saatı döner (timestamp with time zone)	<code>SELECT NOW();</code>	2025-10-20 22:41:32.123+03
<code>CURRENT_TIMESTAMP</code>	<code>NOW()</code> ile aynıdır	<code>SELECT CURRENT_TIMESTAMP;</code>	2025-10-20 22:41:32.123+03
<code>CURRENT_DATE</code>	Sadece tarihi döner	<code>SELECT CURRENT_DATE;</code>	2025-10-20
<code>CURRENT_TIME</code>	Sadece saatı döner	<code>SELECT CURRENT_TIME;</code>	22:41:32.123+03
<code>LOCALTIMESTAMP</code>	Saat dilimi olmadan döner	<code>SELECT LOCALTIMESTAMP;</code>	2025-10-20 22:41:32.123

### ♦ Tarih Formatlama (TO\_CHAR)

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

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
<code>TO_CHAR(tarih, 'Day, DD Mon YYYY')</code>	Gün + tarih	<code>SELECT TO_CHAR(NOW(), 'Day, DD Mon YYYY');</code>	Monday, 20 Oct 2025

## ◆ Tarih Dönüşümü

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

## ◆ Tarih Üzerinde İşlem (INTERVAL)

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

## ◆ Tarih Parçalama (EXTRACT, DATE\_PART)

Fonksiyon	Açıklama	Örnek Kullanım	Örnek Çıktı
<code>EXTRACT(YEAR FROM tarih)</code>	Yıl bilgisi	<code>SELECT EXTRACT(YEAR FROM NOW());</code>	2025
<code>EXTRACT(MONTH FROM tarih)</code>	Ay bilgisi	<code>SELECT EXTRACT(MONTH FROM NOW());</code>	10
<code>EXTRACT(DAY FROM tarih)</code>	Gün bilgisi	<code>SELECT EXTRACT(DAY FROM NOW());</code>	20
<code>EXTRACT(DOW FROM tarih)</code>	Haftanın günü (0=Pazar)	<code>SELECT EXTRACT(DOW FROM NOW());</code>	1
<code>DATE_PART('hour', tarih)</code>	Saat bilgisi	<code>SELECT DATE_PART('hour', NOW());</code>	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
<code>LENGTH(text)</code>	Metindeki karakter sayısını döner (byte sayabilir).	<code>SELECT LENGTH('Ahmet');</code>	5
<code>CHAR_LENGTH(text)</code> veya <code>CHARACTER_LENGTH(text)</code>	Gerçek karakter sayısını döner (UTF8 güvenli).	<code>SELECT CHAR_LENGTH('Çağrı');</code>	5
<code>LOWER(text)</code>	Tüm harfleri küçük yapar.	<code>SELECT LOWER('AHMET');</code>	ahmet
<code>UPPER(text)</code>	Tüm harfleri büyük yapar.	<code>SELECT UPPER('ahmet');</code>	AHMET
<code>INITCAP(text)</code>	Her kelimenin ilk harfini büyük yapar.	<code>SELECT INITCAP('ahmet bedir');</code>	Ahmet Bedir
<code>REVERSE(text)</code>	Metni ters çevirir.	<code>SELECT REVERSE('Ahmet');</code>	temhA

### ♦ Alt String Alma

Fonksiyon	Açıklama	Örnek	Çıktı
<code>SUBSTRING(text FROM start FOR count)</code>	Belirtilen aralıktaki karakterleri döner.	<code>SELECT SUBSTRING('Ahmet' FROM 2 FOR 3);</code>	hme
<code>LEFT(text, n)</code>	Soldan n karakter döner.	<code>SELECT LEFT('Ahmet', 2);</code>	Ah
<code>RIGHT(text, n)</code>	Sağdan n karakter döner.	<code>SELECT RIGHT('Ahmet', 2);</code>	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
POSITION(sub IN text)	Alt dizinin pozisyonunu döner.	SELECT POSITION('m' IN 'Ahmet');	3
LIKE	Desene göre eşleşme	SELECT 'Ahmet' LIKE 'Ah%';	true
ILIKE	Harf duyarsız eşleşme	SELECT 'ahmet' ILIKE 'AH%';	true
~	Regex (büyük/küçük duyarlı)	SELECT 'ahmet' ~ '^[a-z]+\$';	true
~*	Regex (büyük/küçük duyarsız)	SELECT 'Ahmet' ~* 'ahmet';	true

## ♦ Metin Değiştirme ve Temizleme

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

## ♦ Biçimlendirme ve Dönüşürme

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

## ♦ Faydalı Ek Fonksiyonlar

Fonksiyon	Açıklama	Örnek	Çıktı
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

### **tam\_ad**

Mehmet Kaya

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	<a href="mailto:ahmet.bedir@example.org">ahmet.bedir@example.org</a>
Mehmet	<a href="mailto:mehmet.kaya@example.org">mehmet.kaya@example.org</a>
Ali	<a href="mailto:ali.celik@example.org">ali.celik@example.org</a>

## ♦ 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	<a href="mailto:ahmet.bedir@example.com">ahmet.bedir@example.com</a>
Mehmet	<a href="mailto:mehmet.kaya@example.com">mehmet.kaya@example.com</a>

ad	email
Ali	<a href="mailto:ali.celik@example.com">ali.celik@example.com</a>

#### ◆ 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'de Transaction (İşlem) Nedir?

### Başa Dön...

Transaction, bir grup SQL işleminin **tamamının başarıyla yapılması** veya **hiç yapılmaması** demektir. Yani **atomicity (bölünmezlik)** ilkesini sağlar.

 Özette:

**Ya hepsi olur, ya hiçbirı olmaz.**

### Temel Transaction Komutları

#### ► 1. BEGIN

Transaction başlatır.

## ▶ 2. COMMIT

Transaction içindeki tüm işlemleri kalıcı yapar.

## ▶ 3. ROLLBACK

Transaction içindeki tüm işlemleri iptal eder.

## ⌚ Basit Transaction Örneği

```
BEGIN;  
  
UPDATE hesap SET bakiye = bakiye - 500 WHERE id = 1;  
UPDATE hesap SET bakiye = bakiye + 500 WHERE id = 2;  
  
COMMIT;
```

İki sorgudan biri başarısız olursa işlem **ROLLBACK**; ile geri alınır ve bakiyeler değişmez.

## ⚠ Hata Olunca Otomatik Rollback

PostgreSQL şunu yapar:

- Transaction içinde bir hata olursa transaction **ERROR** durumuna geçer.
- Bundan sonra COMMIT edemezsin.
- Mutlaka ROLLBACK yapman gereklidir.

Örnek:

```
BEGIN;  
  
UPDATE users SET age = 'abc'; -- hata  
-- ERROR: invalid input syntax for type integer  
  
ROLLBACK; -- mecburi
```

## ✳ Savepoint (Ara Nokta) Kullanımı

Transaction içinde küçük geri dönüş noktaları.

### ✓ Savepoint Oluştur

```
BEGIN;  
  
UPDATE table1 SET x = 1;  
  
SAVEPOINT s1;  
  
UPDATE table2 SET y = 'aaa'; -- hata olabilir
```

### ✓ Hata olursa savepointe dön

```
ROLLBACK TO s1;
```

- ✓ Devam edeilsin

```
COMMIT;
```

## Transaction Isolation Levels (İzolasyon Seviyeleri)

PostgreSQL'de 4 seviye vardır:

Seviye	Açıklama
<b>READ UNCOMMITTED</b>	PostgreSQL desteklemez (otomatik READ COMMITTED olur)
<b>READ COMMITTED</b>	 Varsayılan. Yalnızca commit edilmiş veriyi görür.
<b>REPEATABLE READ</b>	Aynı transaction içinde tekrar sorguda aynı sonucu alırsın.
<b>SERIALIZABLE</b>	En güvenli ama en yavaş. Çakışmaları engeller.

Seviye seçimi:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

## Transaction ile Fonksiyon Örneği

PL/pgSQL fonksiyonları da otomatik olarak bir transaction içinde çalışır.

```
CREATE OR REPLACE FUNCTION para_transfer(a int, b int, miktar int)
RETURNS void AS $$
BEGIN
    UPDATE hesap SET bakiye = bakiye - miktar WHERE id = a;
    UPDATE hesap SET bakiye = bakiye + miktar WHERE id = b;
END;
$$ LANGUAGE plpgsql;
```

Fonksiyon hata alırsa otomatik rollback olur.

## Transaction Kullanım Senaryoları

- Banka işlemleri
- Sipariş oluşturma
- Çoklu tablo güncellemeleri
- Kritik log kayıtları
- Veri bütünlüğünün önemli olduğu her şey

# 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) belirlenebilir:

```
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 kullanımında ise önce her bir veritabanında bu rolün sahiplendiği nesneler başka rollere devredilir ya da silinir, sonra kullanıcı silinir.
-