

# SQL

## Eğitim Notları



Orta Seviye Java ile Web Development Patikası-

SQL Eğitimi Ders Notlarıdır / 02.01.2024

Eğitmen: Gürcan Çekiç / Notları Derleyen: Emre Er

# İÇİNDEKİLER

Veri ve Veri Tabanı .....	5
Veri .....	5
Veri Tabanı.....	5
Neden Veritabanı ?.....	5
Veri Tabanı Yönetim Sistemi .....	6
RDBMS.....	6
SQL Nedir ? .....	7
PostgreSQL Kurulumu.....	8
PGAdmin.....	11
SQL Temelleri.....	13
SELECT .....	13
WHERE.....	14
AND ve OR Bağlacı.....	14
WHERE NOT .....	15
Örnekler:.....	15
BETWEEN ve IN.....	16
LIKE ve ILIKE.....	17
DISTINCT .....	19
COUNT .....	19
Terminal Ekranı Kullanımı – PSQL Uygulaması .....	19
Postgre SQL Giriş .....	19
ORDER BY .....	20
LIMIT .....	21
OFFSET .....	21
Aggregate Fonksiyonlar .....	22
COUNT .....	22
MAX .....	22
MIN .....	22
SUM .....	23
AVG .....	23
GROUP BY .....	25

HAVING .....	26
ALIAS .....	27
CONCAT .....	27
Tablo Oluşturmak ve Silmek .....	28
CREATE .....	28
INSERT INTO .....	28
DROP .....	29
Verileri Güncellemek – Silmek .....	30
UPDATE .....	30
RETURNING .....	30
DELETE .....	31
PRIMARY KEY .....	32
FOREIGN KEY .....	33
Veri Tipleri .....	34
Temel Veri Tipleri .....	34
Karakter Veri Tipleri .....	35
Boolean Veri Tipleri .....	35
Zaman / Tarih Veri Tipleri .....	35
NOT NULL .....	36
NOT NULL Kullanımı .....	36
ALTER ve NOT NULL .....	36
UNIQUE .....	37
UNIQUE Kullanımı .....	37
ALTER ve UNIQUE .....	37
CHECK .....	38
CHECK Kullanımı .....	38
ALTER ve CHECK .....	38
psql Uygulaması-II .....	39
JOIN .....	40
INNER JOIN .....	40
LEFT JOIN .....	41
RIGHT JOIN .....	42
FULL JOIN .....	42

UNION.....	43
INTERSECT .....	43
EXCEPT.....	43
Alt Sorgular – Subqueries .....	44
Alt Sorgu Kullanımı .....	44
Any Operatörü.....	45
ALL Operatörü .....	45
Alt Sorgular ve JOIN Kullanımı .....	46

## **Veri ve Veri Tabanı**

---

### **Veri**

Veri, sayımlar, deneyler, gözlemler, araştırmalar ve ölçümler sonucu elde ettiğimiz ham bilgilerdir.

### **Veri Tabanı**

Veri tabanı ise, bu verileri organize bir şekilde tutup, depolayan ve bu verileri manipülasyonunu sağlayabilen sistemlerdir. Bilgisayarlarımız, cep telefonlarımız ve akıllı saatlerimiz veri tabanına sahiptirler.

### **Neden Veritabanı ?**

Excel ve benzeri yazılımlarla biz verilerimizi depolayabiliyoruz. Ancak veritabanları sayesinde;

- Tuttuğumuz verilerin aynı tipte olduğunu garanti edebiliriz,
- Veri tabanları diğer uygulamalar ile daha uyumlu çalışırlar,
- Veri tabanları çok büyük verileri depolayabilirler ve
- Çoklu kullanıcı yönetimi için veritabanları daha uygundur.

## Veri Tabanı Yönetim Sistemi

---

Verilerimizi, veri tabanı sayesinde tutuyoruz. Ancak, bu veri tabanını oluşturma, düzenleme ve SQL sayesinde sorgulama işlemlerini yapan yazılımlara Veri Tabanı Yönetim Sistemleri (DBMS) denir. İhtiyaçlara göre farklı veri tabanı yönetim sistemleri bulunur.

Temel Veri Tabanı Yönetim Sistemleri:

İlişkisel veritabanı (RDBMS)

Hiyerarşik Veri Tabanı

Ağ Veri Tabanı

Nesneye Yönelik Veri Tabanı

### RDBMS

İlişkisel Veri Tabanı Yönetim Sistemlerinde, veriler satır ve sütundan oluşan tablolarda tutulmaktadır. Bu tablolarda bulunan sütunlardaki veriler aynı veri türü ile tutulur. Dolayısıyla yüksek bir veri tutarlılığına sahiptir.

Popüler olan DBMS'ler aşağıdaki gibidir:

MySQL, SQL Server, MongoDB, PostgreSQL gibi.

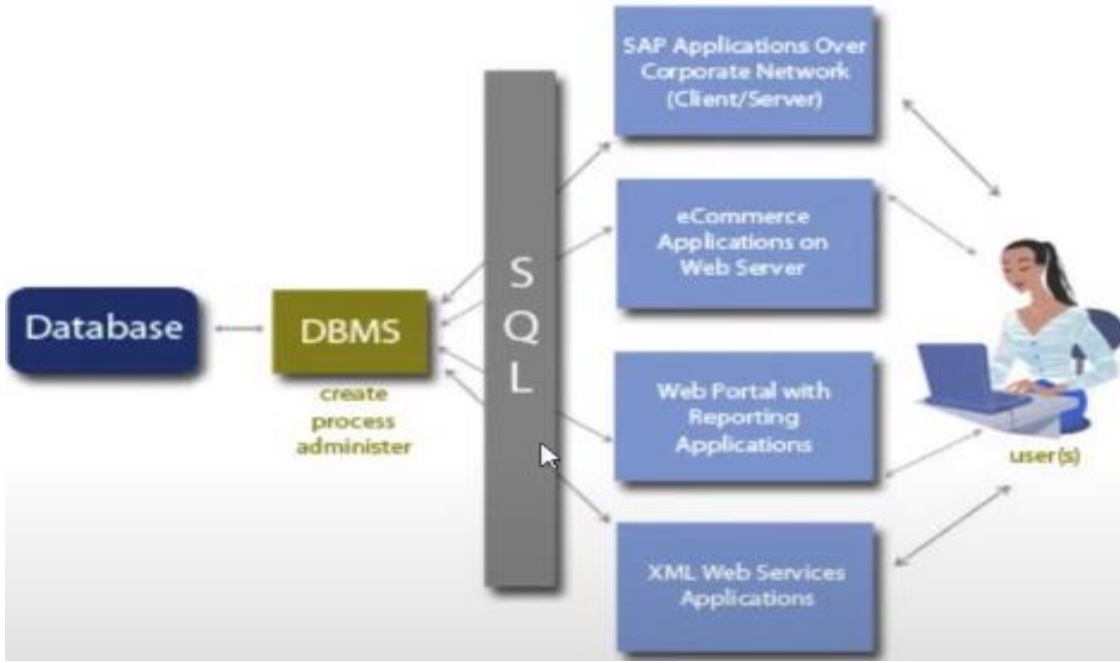


## SQL Nedir ?

Veritabanı yönetim sistemleri aracılığıyla SQL dilini kullanarak veri tabanından verileri alırız. SQL, veri tabanı yönetim sistemi ile iletişim kurmamızı sağlayan programlama dilidir.

SQL, declarative bir programlama dilidir. Declarative demek, sonuçta bizim için ne olacağı önemli demektir.

Imperative ve Declarative programlama mantığından bahsedecek olursak, Imperative (emredici) Programming'te, sonucun nasıl olacağı, hangi aşamaların gerçekleştirileceği vardır. Declarative (Beyan Edici) Programming'te ise, sonucun ne olacağı söz konusudur. Örneğin, "Film" tablosundan, "Title" ve "Release Year" sütunlarının çekilmesi, Beyan Edici bir durumdur.



SQL, 4.Nesil bir programlama dilidir. Burada Nesil'den kasıt, o programlama dilinin giderek İnsan diline yakınlaşması ile alakalıdır.

1.NESİL: Makine dili seviyesindeki Programlama Dilleri 1.Nesil (1GL) olarak sınıflandırılırlar.

2.NESİL: Assembly programlama dilleri 2.Nesil (2GL) olarak sınıflandırılır.

3.NESİL: C, C++, Java, Python, PHP, Perl, C#, BASIC, Pascal, Fortran, ALGOL, COBOL gibi filleri 3.Nesil (3GL) olarak sınıflandırılırlar.

4.NESİL: SQL, ABAP, R gibi diller 4Nesil (4GL) Programlama Dilleri olarak sınıflandırılırlar.

## PostgreSQL Kurulumu

Veri tabanı yönetim sistemi olarak Postgre SQL kurulumu için aşağıdaki link'e tıklanır.

<https://www.postgresql.org/download/>

Buradan, İşletim Sistemi seçilir. Windows kullanıcıları için Windows kısmından indirebilir.

### Quick Links

- Downloads
  - Packages
  - Source
- Software Catalogue
- File Browser

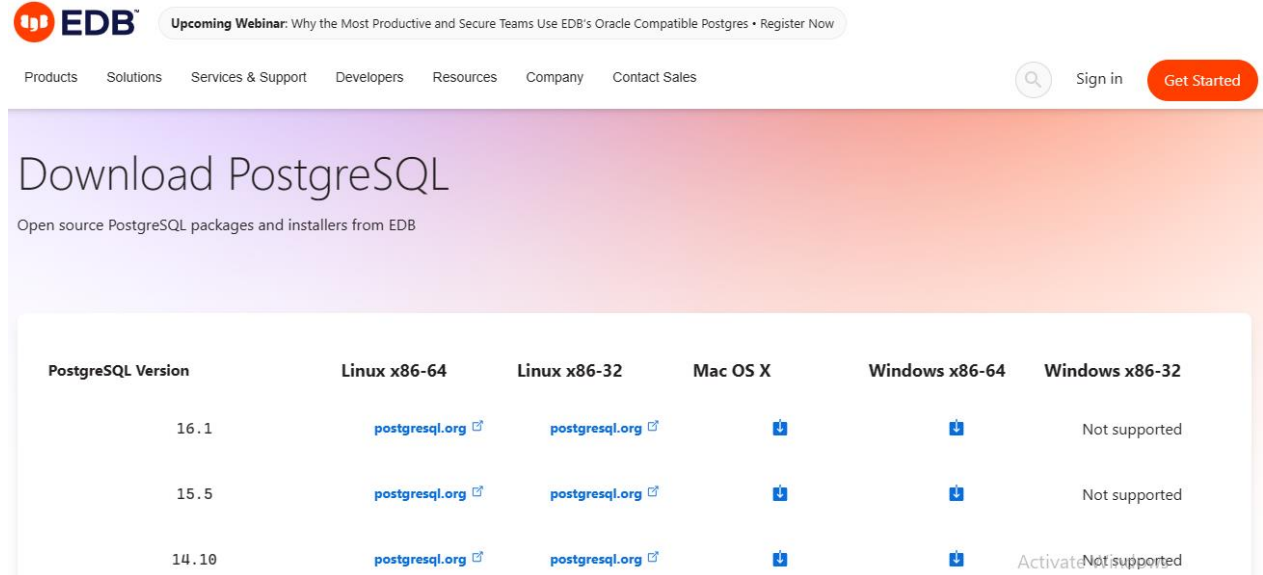
### Windows installers







#### Interactive installer by EDB

**Download the installer** certified by EDB for all supported PostgreSQL versions.

**Note!** This installer is hosted by EDB and not on the PostgreSQL community servers. If you have any issues, please contact [webmaster@enterprisedb.com](mailto:webmaster@enterprisedb.com).

Download the installer linkine tıklanır.



PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
16.1	<a href="https://www.postgresql.org/">postgresql.org</a>	<a href="https://www.postgresql.org/">postgresql.org</a>			Not supported
15.5	<a href="https://www.postgresql.org/">postgresql.org</a>	<a href="https://www.postgresql.org/">postgresql.org</a>			Not supported
14.10	<a href="https://www.postgresql.org/">postgresql.org</a>	<a href="https://www.postgresql.org/">postgresql.org</a>			Activated Not supported

Bilgisayar sistemi kaç bit'lik ise, ona göre en sonki verisyon tıklanarak indirme işlemi başlatılır:

- İnen .exe kurulum dosyası açılır.
- Next denerek bir sonraki kısımda Installation Directory olarak dosya/klasör konumu belirtilir, uygunsa devam edilir.
- Select Components kısmında hepsi seçili olarak bırakılır, Next'e tıklanır. Data Directory kısmı kontrol edilir, uygunsa devam edilir.
- Super User için bir password belirlememiz gerekiyor. Parola belirlenir ve tekrar girilir.



- Port 5432 olarak gelir, Next diyoruz.
- Locale kısmıda aynı kalacak şekilde Next diyoruz.
- Pre Installation Summary kısmında kurulumun nereye yapılacağını söylüyor buna da Next deyip kurulumu başlayabiliriz.

Kurulumdan sonra:

- Kurulum dosyasının olduğu postgresql klasörünün içine girilir. Buradan da "bin" klasörünün içerisine girilir. Burada, PostgreSQL komutlarını çalıştıran programcıklar bulunur.
- Path kopyalanıp, terminal başlatılır (Windows + R tuşları ile "Çalıştır" ekranı açılır. Cmd yazılıp Tamam'a tıklanır.) Psql, PostgreSQL'in komut ekranıdır. Klasördeki herhangi bir komutu çalıştırmak/çalıştırabilmek için bunu/bu path'i tanımlamamız gereklidir.
- Terminal'e `cd` yazdıktan sonra boşluk bırakıp kopyalanan path'i yapıştırdıktan sonra Enter'a basılır.
- Burada bizden şifre ister ancak bunun olmaması gerekir. Ctrl+C ile çıkılır.
- Postgres kullanıcısı ile veritabanına giriş yapacağımızı belirtmek için: `psql -U postgres` yazılır Enter'a basılır.
- Ekranda şifre kısmına Super User için istenilen şifre girilir.

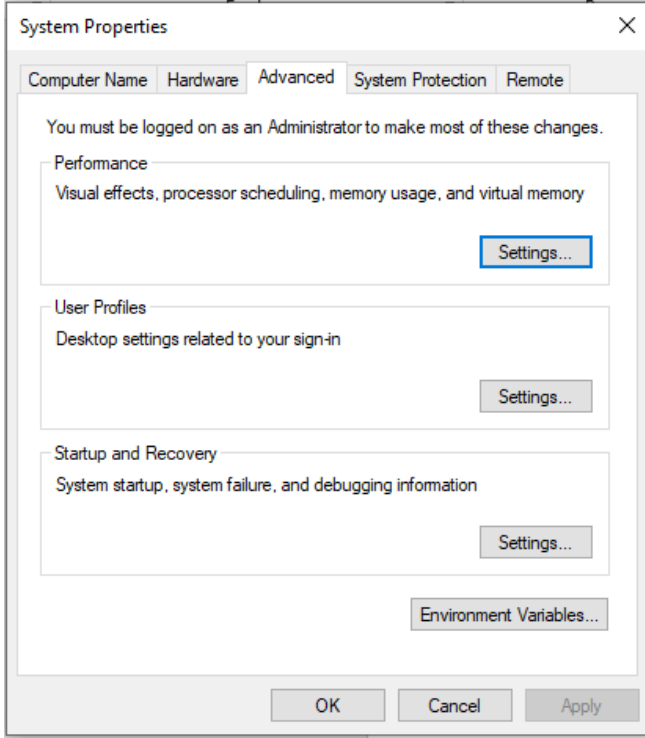
```
C:\Program Files\PostgreSQL\16\bin>psql -U postgres
postgres kullanıcısının parolası:
psql (16.1)
UYARI: Üçbirimin kod sayfası (857), Windows kod sayfasından (1254) farklıdır
      8-bitlik karakterler doğru çalışmayabilir. Ayrıntılar için psql referans
      belgelerinde "Windows kullanıcılarına notlar" bölümüne bakın.
Yardım için "help" yazınız.

postgres=#
```

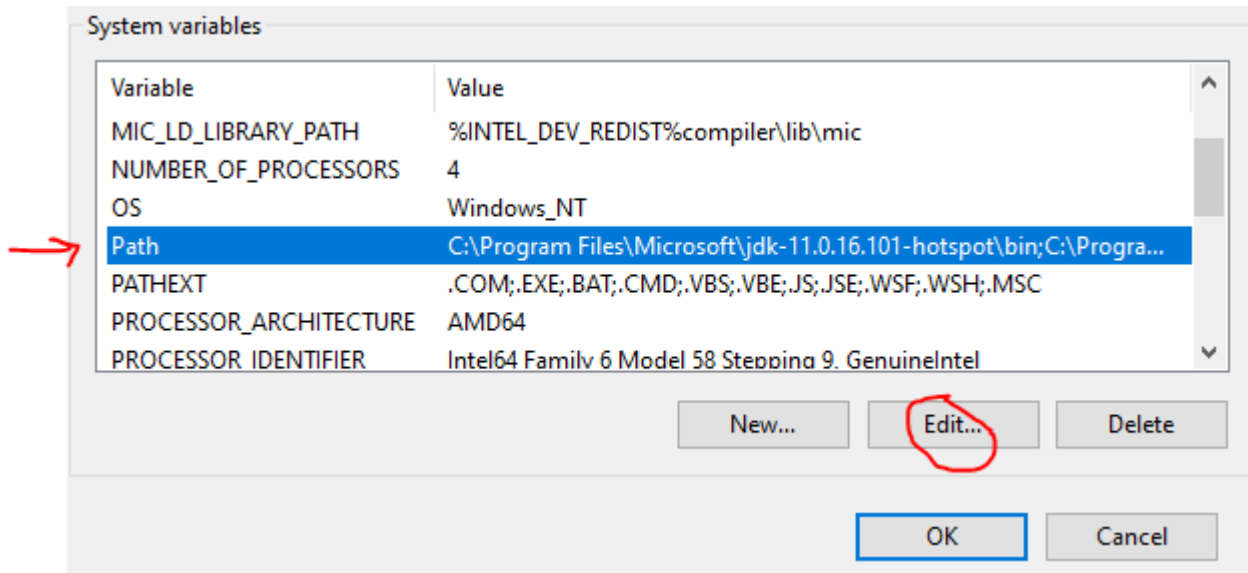
*Bu şekilde Veri Tabanı Yönetim Sistemi ile bağlantımızı kurmuş olduk.*

Bağlantıyı her seferinde bu şekilde yapmayacağız. Bunun için aşağıdaki adımlar izlenecektir.

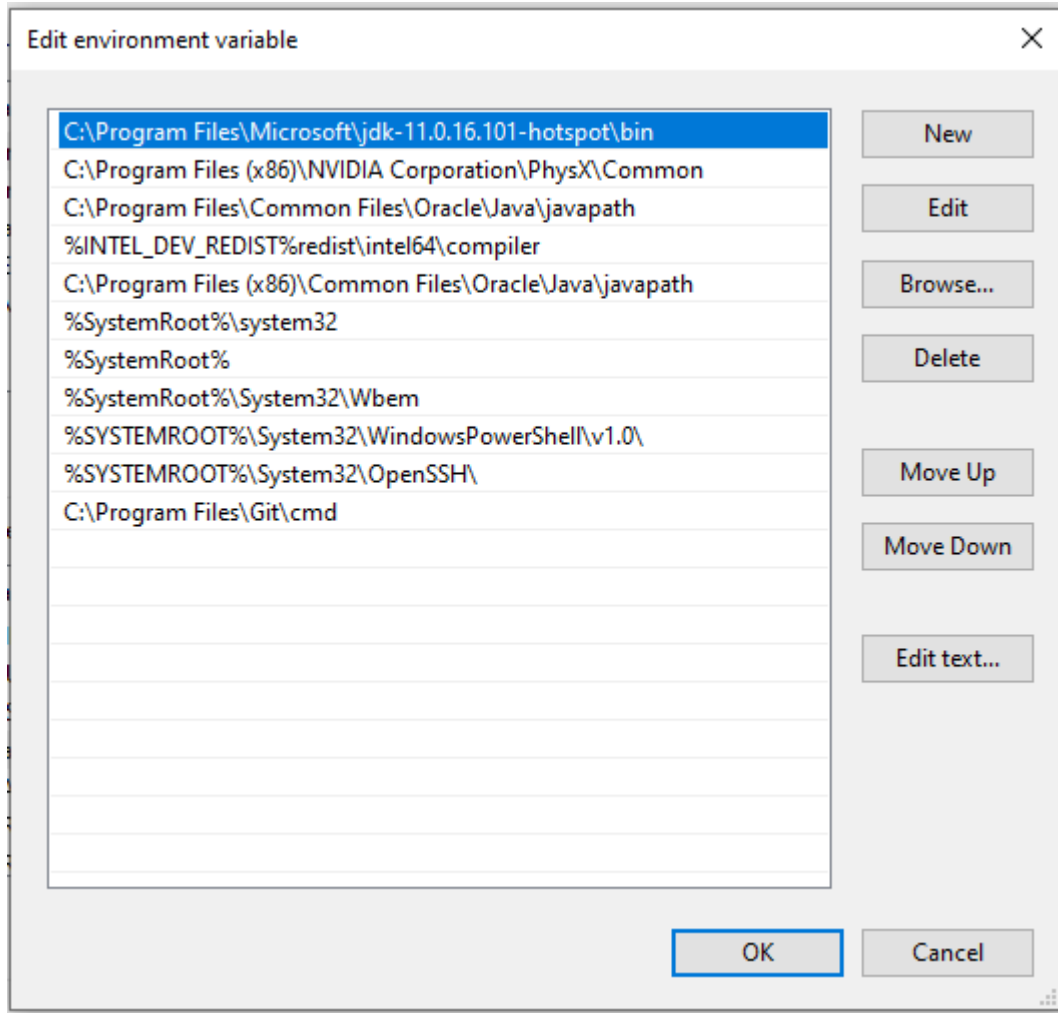
- Dosya yolu tekrar kopyalanır.
- Sol alt Windows menüsündeki arama çubuğuna “Edit the Systems Environment” yazılır, tıklanır. Environment Variables’a tıklanır.



- System variables kısmından path seçilip, Edit’e tıklanır.



- Çıkan ekranda New butonuna tıklanır ve altta çıkan boşluğa kopyalanan path yapıştırılır. OK'ye tıklanır.



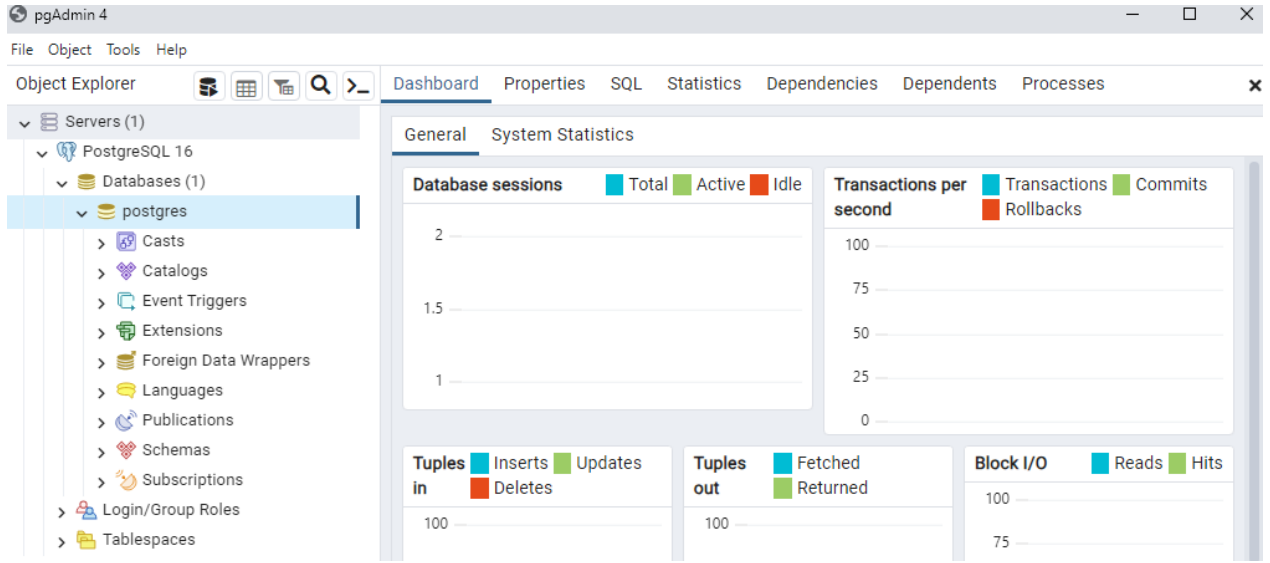
- Diğer pencerelerde de OK'ye tıklanılarak çıkılır.
- Tekrar komut satırı (Terminal) açılır.
- `psql -U postgres` denilerek Enter'a basılır.
- Bu sefer Super User şifresi isteyecektir, şifre girilir. Böylece psql tanımlaması tamamlanmıştır.

## PGAdmin

PGAdmin'e tıklanılarak, postgresSQL arayüzünü çalıştırırız.

Psql, terminal arayüzümüz, PostgreSQL kullanıcı dostu olan bir grafik arayüzüdür. Biz, PGAdmin ile yaptığımız işlerin tamamını psql ile de yaparız.

PGAdmin'e giriş yapar yapmaz ilk çıkan ekranda şifre isteyecektir. Şifreyi girip, kaydedelim.



Sol tarafta görüldüğü gibi, postgres veritabanı da otomatik olarak geldi. Bu postgresql'in bize vermiş olduğu örnek bir veritabanıdır. Ancak SQL komutları üzerine çalışırken hazır bir veritabanı kullanacağız.

Aşağıdaki adreste, sample olarak zip formatında bir database mevcuttur. Bunu indirebiliriz.

<https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/>

- Zip dosyasını açalım. İçinden .tar uzantılı bir dosya çıkacaktır. Bu bir arşiv dosyasıdır.
- PGAdmin'e gelerek, PostgreSQL simgesine Sağ Tıklayıp, sırayla Create ve Database'e tıklanır. Database'e isim verilir, Kaydedilir.
- Oluşturduğumuz veri tabanının simgesine Sağ Tıklayıp Restore'a tıklanır. Select File'a gelinerek .tar uzantılı dosya seçilir. Restore'a tıklanır.
- Veri tabanımız yüklenmiş oldu. Sağ Tıklayıp Refresh edelim.
- Veri tabanımızdan Schemas'a gelelim. Buradan Tables'a gelelim.
- film'in üzerine sağ tıklayıp View Data ve All Columns'a tıklayalım.

## SQL Temelleri

---

### SELECT

SELECT bizim istediğimiz verileri görmemizi sağlar.

**SELECT** release\_year **from** film; ifadesi, film isimli tablo'dan(obje) release\_year kolonunu göster anlamına gelmektedir. Bu satırı yorum olarak dönüştürmek için, başına iki tane - işareti koyarız.

--SELECT release\_year from film; (Yorum satırı oldu)

Sadece tek kolonu değil birden fazla kolonu da görebiliriz. Bunun için araya virgül koyarız.

**SELECT** release\_year, title **from** film;

SQL komutları Case Sensitive değildir. Yani bu sorguları küçük harfle de yazsak, komutumuz / sorgumuz çalışacaktır. Ancak sorgunun /Query okunurluğu açısından büyük harfler ile yazmakta fayda vardır.

**select** release\_year, title **from** film;

Soru ifadesinin sonuna noktalı virgül kymasak da sorgumuz gerçekleşir ancak, farklı sorgularda bunu koymak zorundayız.

Tüm sütunları getirmek için, sütun isimlerini tek tek yazmak yerine \* asteriks işaretini koyarız.

**SELECT** \* **FROM** film;

Böylece film tablosundaki tüm verileri bize getirir.

## WHERE

**SELECT** komutu ile yaptığımız çalışmalarda bizler tüm sütunların veya ilgili sütunlarda bulunan verilerin tamamını çekmek isteriz. Çoğu durumda ise verilerin tamamını değil belirli koşulları sağlayan verileri görmek isteriz. Bunun için **WHERE** anahtar kelimesini kullanırız.

```
SELECT * from actor
```

```
WHERE first_name='Penelope'
```

Bu şartlı sorguya göre, actor isimli tablodaki tüm verilerden, first\_name'i Penelope olanları getiriyoruz.

Karşılaştırma operatörleri:

Büyüktür >	Büyük Eşit >=	
Küçüktür <	Küçük Eşit <=	
Eşit =	Eşit Değil !=	Eşit Değil <>

film isimli tablodan, rental\_rate oranı 1'e eşit ve küçük olanları getirmek için;

```
SELECT * FROM film
```

```
WHERE rental_rate<=1;
```

## AND ve OR Bağlacı

```
SELECT * FROM film
```

```
WHERE rental_rate>4 AND replacement_cost >20.0;
```

Hem rental\_rate 4'ten büyük, hem de replacement\_cost 20'den büyük olanları getirecektir.

```
SELECT * FROM film
```

```
WHERE rental_rate>4 OR rental_rate < 2;
```

rental\_rate 4'ten büyük ya da rental\_rate 2'den küçük olanları getirecektir.

Başka bir örnek;

```
SELECT * FROM film
```

```
WHERE rental_rate=4 OR rental_rate = 2;
```

rental\_rate 4'e veya rental\_rate 2'ye eşit olanları getirecektir.

## WHERE NOT

```
SELECT * FROM film
```

```
WHERE NOT (rental_rate=4.99 AND replacement_cost = 14.99);
```

Burada, parantez içindeki koşul sağlanmamış oldu.

```
SELECT * FROM film
```

```
WHERE NOT NOT rental_rate=4.99;
```

Burada ise, rental\_rate=4.99'a eşit olmayanların tersi, yani 4.99'a eşit olma durumu söz konusudur.

## Örnekler:

1.

```
SELECT * FROM actor
```

```
WHERE first_name='Penelope' AND last_name='Guiness' OR first_name='Nick';
```

Burada, OR'dan itibaren sanki yeni bir sorgu başlarmış gibi oluyor.

Query

Query History

1

SELECT \* from actor

2

WHERE first\_name='Penelope' AND last\_name='Guiness' OR first\_name='Nick';

Data Output

Messages

Notifications

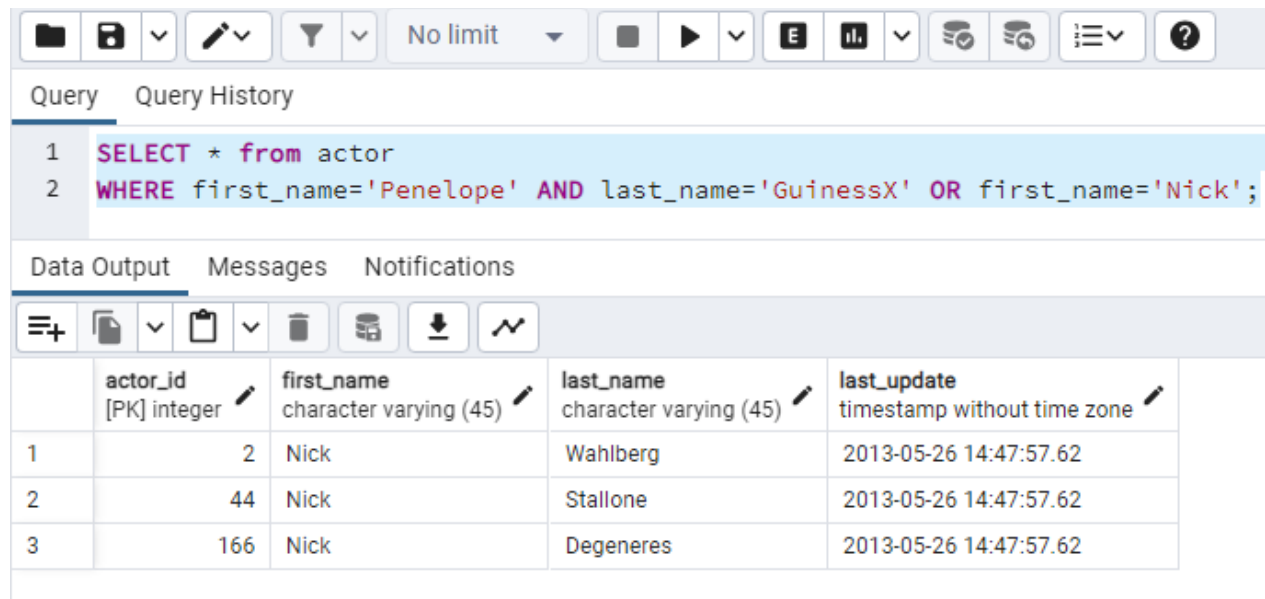
	actor_id [PK] integer	first_name character varying (45)	last_name character varying (45)	last_update timestamp without time zone
1	1	Penelope	Guiness	2013-05-26 14:47:57.62
2	2	Nick	Wahlberg	2013-05-26 14:47:57.62
3	44	Nick	Stallone	2013-05-26 14:47:57.62
4	166	Nick	Degeneres	2013-05-26 14:47:57.62

2.

```
SELECT * FROM actor
```

```
WHERE first_name='Penelope' AND last_name='GuinessX' OR first_name='Nick';
```

Bu örnektei last\_name'i GuinessX olan bir soyadı yok, yani koşul sağlanmaz ancak, OR'dan sonraki sorgu çalışır.



The screenshot shows a SQL query editor interface. At the top, there's a toolbar with icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is active, displaying the following SQL query:

```
1 SELECT * from actor
2 WHERE first_name='Penelope' AND last_name='GuinessX' OR first_name='Nick';
```

Below the query, the 'Data Output' tab is active, showing the results of the query. The results are displayed in a table with the following columns: actor\_id, first\_name, last\_name, and last\_update. The table contains three rows of data:

	actor_id [PK] integer	first_name character varying (45)	last_name character varying (45)	last_update timestamp without time zone
1	2	Nick	Wahlberg	2013-05-26 14:47:57.62
2	44	Nick	Stallone	2013-05-26 14:47:57.62
3	166	Nick	Degeneres	2013-05-26 14:47:57.62

## BETWEEN ve IN

```
SELECT * FROM film
```

```
WHERE length BETWEEN 50 AND 60; --50 ile 60 arasında olanlar
```

50 ile 60 arasındaki length değerine sahip verileri getirir. (50 ve 60 dahildir)

NOT ifadesi ile de kullanmak mümkündür. Örneğin;

```
SELECT * FROM film
```

```
WHERE length NOT BETWEEN 50 AND 60; --50 ile 60 arasında olmayanlar
```



```
SELECT * FROM film
```

```
WHERE length IN (50,60,70); --50,60 ve 70 olan değerler gelir
```

Belli bir veri kümesini seçmek için, çok fazla **OR** ifadesi kullanmak yerine **IN** ifadesini kullanırız. 1 veya 1'den fazla kullanabiliriz.

Burada da **NOT** ifadesi kullanılabilir.

```
SELECT * FROM film
```

```
WHERE length NOT IN (50,60,70); --50,60,70 olmayan değerler gelir.
```

### LIKE ve ILIKE

```
SELECT * FROM customer
```

```
WHERE first_name LIKE 'A%'; --A ile başlayan isimleri getirdi. *
```

```
SELECT * FROM customer
```

```
WHERE first_name LIKE '%y'; --y harfi ile biten isimleri getirdi.
```

```
SELECT * FROM customer
```

```
WHERE first_name LIKE 'J%d'; --J ile başlayıp d harfi ile biten isimleri getirdi.
```

\*Burada % işareti 1'den fazla karakter için yer tutmaktadır.

\*\*Sadece 1 karakterlik yer tutmak için alt tire “\_” işareti kullanılmaktadır.

```
SELECT * FROM customer
WHERE first_name LIKE 'A_y'; **
```

Query		Query History				
1	SELECT * FROM customer					
2	WHERE first_name LIKE 'A_y';					
Data Output		Messages				
	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character var	
1	32	1	Amy	Lopez	amy.lopez@e	

**LIKE** komutu Case Sensitive'dir. Case sensitive arama olmaması için **ILIKE** komutu kullanılır.

```
SELECT * FROM customer
WHERE first_name LIKE '%Y'; --Burada sonu büyük Y harfi ile biten isim
--olmadığı için bir veri getirmeyecektir. Ancak, ILIKE ile bunun önüne
--geçebiliriz.
SELECT * FROM customer
WHERE first_name ILIKE '%Y';
```

**NOT** deyimini **LIKE** ve **ILIKE**'da da kullanabiliriz.

```
SELECT * FROM customer
WHERE first_name NOT LIKE 'A%'; --A harfi ile başlamayanları getirdi
```

Diğer Şekilde Kullanım:

```
~~ LIKE
~~* ILIKE
!~~ NOT LIKE
!~~* NOT ILIKE
```

```
SELECT * FROM customer
WHERE first_name !~~* 'J%'; --J harfi ile başlamayanları getirdi
```

## DISTINCT

**DISTINCT** bizim veri listemizdeki benzersiz verileri sıralar.

```
SELECT DISTINCT first_name
```

```
FROM customer;
```

## COUNT

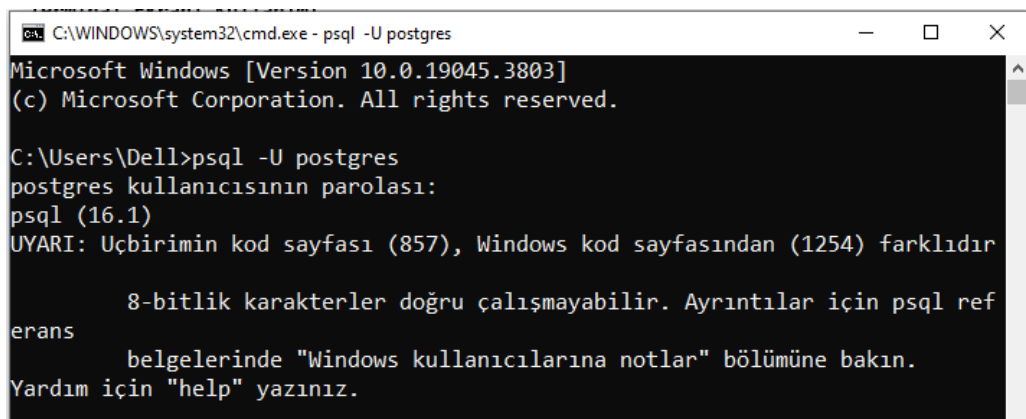
**COUNT** ise aranılan veri sayısını bize döndürür. **DISTINCT** ile birlikte kullanılabilir.

```
SELECT COUNT (DISTINCT rental_rate)
```

```
FROM film;
```

## Terminal Ekranı Kullanımı - PSQL Uygulaması Postgre SQL Giriş

- `psql -U postgres` //Kullanıcı adımızla giriş
- Şifreyi gireriz.
- `\l` ile veritabanlarını listeleriz
- `connect` ile veya `\c` ile istediğimiz listeye bağlarız.
- `\c dvdrental` --dvdrental isimli veri listesine ulaşırız.
- Buradan itibaren, SQL komutlarını arayüzdeki gibi (PGAdmin)çalıştırabiliriz.
- Bir alt satıra geçmek için “`;`” koymayız.
- Bir geri çıkmak için Ctrl + C ‘ye basarız.
- Daha fazla basarsak başa döner.
- En son `exit` komutu ile terminalimizi kapatabiliriz.



```
C:\WINDOWS\system32\cmd.exe - psql -U postgres
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>psql -U postgres
postgres kullanıcısının parolası:
psql (16.1)
UYARI: Üçbirimin kod sayfası (857), Windows kod sayfasından (1254) farklıdır
      8-bitlik karakterler doğru çalışmayabilir. Ayrıntılar için psql ref
erans
      belgelerinde "Windows kullanıcılarına notlar" bölümüne bakın.
Yardım için "help" yazınız.
```

## ORDER BY

**ORDER BY** ile verilerimizi belirli bir sütuna göre sıralayabiliriz.

```
SELECT * FROM film
```

```
ORDER BY title;
```

Böylelikle, film tablomuzdaki verilerimiz, title sütununa göre sıralandı. title'da veriler alfabetik sıraya göre sıralanmıştır. Bu tarz bir sıralama

'Ascending' yani artan bir sıralamadır.

Yine sorgumuzu aşağıdaki gibi yazarsak, aynı şekilde alfabetik olarak title sütununu sıralamış oluruz:

```
SELECT * FROM film
```

```
ORDER BY title ASC;
```

Ancak verilerimizi biz azalacak şekilde 'Descending' sıralamak istersek de aşağıdaki gibi sorgumuzu yazarız.

```
SELECT * FROM film
```

```
ORDER BY title DESC;
```

Sadece title ve length'i görelim ve length'i de azalan şekilde sıralayalım:

```
SELECT title,length FROM film
```

```
ORDER BY length DESC;
```

Karmaşık bir şekilde yapacak olursak, örneğin,

rental\_rate artacak şekilde ama length azalacak şekilde listeleyelim:

```
SELECT title,rental_rate,length FROM film
```

```
ORDER BY rental_rate ASC,length DESC;
```

Araya **WHERE** ifadesi koyarak bir filtreleme de yapabiliriz:

```
SELECT title,rental_rate, length FROM film
```

```
WHERE title LIKE 'A%'
```

```
ORDER BY rental_rate ASC,length DESC;
```

## LIMIT

**LIMIT** ile verilerimizi sınırlandırabiliriz. Örneğin biz 1000 satırlık bir veri topluluğundan sadece belirli bir sayıda veri alabiliriz.

```
SELECT * FROM film
```

```
LIMIT 25; --tablomuza 25 tane veri gelir.
```

## OFFSET

**OFFSET** ile de, verilerimizi hangi satırdan itibaren almak / getirmek istiyorsak yanında bir sayı belirterek kullanırız.

Bir örnek yapalım. actor tablomuzu seçelim, buradan da sadece ismi 'Penelope' olanları getirelim.

```
SELECT * FROM actor
```

```
WHERE first_name = 'Penelope';
```

	actor_id [PK] integer	first_name character varying (45)	last_name character varying (45)	last_update timestamp without time zone
1	1	Penelope	Guiness	2013-05-26 14:47:57.62
2	54	Penelope	Pinkett	2013-05-26 14:47:57.62
3	104	Penelope	Cronyn	2013-05-26 14:47:57.62
4	120	Penelope	Monroe	2013-05-26 14:47:57.62

Bu tablodan, sadece, last\_name'i Monroe olanı getirmek istersek, önce last\_name'i ascending olarak sıralarız, sonrasında da **OFFSET**'i 2 veririz. En son da **LIMIT** değerimizi 1 olarak veririz.

```
SELECT * FROM actor
```

```
WHERE first_name = 'Penelope'
```

```
ORDER BY last_name
```

```
OFFSET 2
```

```
LIMIT 1;
```

Data Output Messages Notifications				
	actor_id [PK] integer	first_name character varying (45)	last_name character varying (45)	last_update timestamp without time zone
1	120	Penelope	Monroe	2013-05-26 14:47:57.62










## Aggregate Fonksiyonlar

Aggregate fonksiyonlar yardımıyla bir veri küme verisini bir araya getiririz ve bu verilerin toplamını, maksimum ve minimum değerlerini, adetini, ortalamasını bulabiliriz. Örneğin film tablomuzdaki rental\_rate değerlerinin toplamını bulabiliriz. Veya, length verilerinin minimum ve maksimum değerlerini görebiliriz. Sonuç itibarıyla bu aggregate fonksiyonları sayesinde tek bir sonuç elde ederiz. Aggregate fonksiyonlardan sonra parantez konulur.

### COUNT

Daha önce gördüğümüz COUNT da bir aggregate fonksiyondur.










```
SELECT COUNT(*) FROM film
```

Data Output	Messages	Notifications
        		
	count bigint	
1	1000	

### MAX

Maksimum değeri elde ederiz.










```
SELECT MAX(length) FROM film;
```

Data Output	Messages	Notifications
        		
	max smallint	
1	185	

### MIN

Minimum değeri elde ederiz.

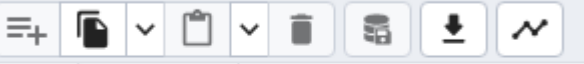
```
SELECT MIN(length) FROM film;
```

Data Output	Messages	Notifications
        		
	min smallint	
1	46	

## SUM

Toplam değeri elde edriz.


```
SELECT SUM(length) FROM film;
```

Data Output Messages Notifications		
		
	sum bigint	🔒
1	115272	

## AVG


Ortalama değeri elde edriz.

```
SELECT AVG(rental_rate) FROM film;
```

Data Output Messages Notifications		
		
	avg numeric	🔒
1	2.9800000000000000	


Burada, noktadan sonraki sayıları silmek için / yuvarlama yapmak için **ROUND** ifadesini kullanırız. Noktadan sonra kaç hane olacağını da belirtiriz.

```
SELECT ROUND (AVG(rental_rate),3) FROM film;
```

Data Output Messages Notifications		
		
	round numeric	🔒
1	2.980	

Veya tamamen yuvarlamak istiyorsak, hane kısmı için 0 yazarız.

```
SELECT ROUND (AVG(rental_rate),0) FROM film;
```

Data Output Messages Notifications		
		
	round numeric	🔒
1	3	

Aggregate fonksiyonlarını birden çok kullanabiliriz.

```
SELECT MAX(length), MIN(length),SUM(rental_rate) FROM film;
```

Query

Query History










1

SELECT MAX(length), MIN(length),SUM(rental\_rate) FROM film;

Data Output

Messages

Notifications



	max smallint	min smallint	sum numeric
1	185	46	2980.00

Aggregate fonksiyonlar ile sorgularımızı yaparken, **WHERE** ifadesi de kullanabiliriz.

```
SELECT MAX(length), MIN(length), SUM(rental_rate) FROM film
```

```
WHERE title LIKE 'A%';
```

Data Output		Messages	Notifications
	max smallint	min smallint	sum numeric
1	181	46	131.54

Peki, rental\_rate oranı 4.99 olan en uzun film hangisidir, veya rental\_rate oranı 0.99 olan en kısa film hangisidir bunu getirmek için aşağıdaki sorguları yazarsak **GROUP BY** hatası alırız.

Bunun için **GROUP BY** kullanırız.

Query	Query History	
1	SELECT MAX(length), MIN(length),SUM(rental_rate), title FROM film	
2		
Data Output	Messages	Notifications
ERROR: aggregate fonksiyonu kullanmak için "film.title" sütununu GROUP BY listesine eklemelisiniz LINE 1: SELECT MAX(length), MIN(length),SUM(rental_rate), title FROM... 		



## GROUP BY

Veritabanımızdaki bi tabloda, örneğin *dvdrental* veritabanının *film* tablosunda, bir sütundaki verinin karşılığı olan bir verinin maksimum veya minimum değerini getirmek istersek bunu **WHERE** komutunu kullanarak bir grupta yapabiliriz. Ancak, bunun yerine **GROUP BY** kullanırız.

Örnek verecek olursak:

dvdrental veritabanında rental\_rate sütununda bizim 3 farklı değerimiz var (0.99, 2.99, 4.99). Biz bu 3 farklı değer için en uzun filmi bulmaya çalışalım.

```
SELECT MAX(length) FROM film
```

```
WHERE rental_rate = 0.99;
```

```
SELECT MAX(length)
```

```
FROM film
```

```
WHERE rental_rate = 2.99;
```

```
SELECT MAX(length)
```

```
FROM film
```

```
WHERE rental_rate = 4.99;
```

İstediğimiz sonuçları elde ediyoruz ancak şöyle bir sorunun var 3 farklı değer yerine 30 farklı değer olsaydı? İşte bu şekilde senaryolar için yani verileri grupta için **GROUP BY** anahtar kelimesi kullanılır.

```
SELECT rental_rate, MAX(length) FROM film
```

```
GROUP BY rental_rate;
```

```
1 SELECT rental_rate, MAX(length) FROM film
2 GROUP BY rental_rate;
```

	rental_rate numeric (4,2)	max smallint
1	2.99	185
2	4.99	185
3	0.99	184

Burada, **SELECT**'ten sonra rental\_rate'i yazmamızın amacı listedeki uzunlukların hangi rental rate'e karşılık geldiğini göstermek içindir.

## HAVING

Gruplanmış verilerimize koşul sağlamak için kullanılır. Hemen **GROUP BY**'dan sonra kullanılır.

Örneğin rental\_rate oranlarına gre bir gruplama yapalım; ve bunların listede kaç adet olduğunu verelim:

```
SELECT rental_rate, COUNT(*) FROM film
GROUP BY rental_rate;
```

	rental_rate numeric (4,2)	count bigint
1	2.99	323
2	4.99	336
3	0.99	341

Şimdi de, sayısı 323'ten fazla olanları getirelim, yani **HAVING** ile şart uygulayalım:

```
SELECT rental_rate, COUNT(*) FROM film
GROUP BY rental_rate
HAVING COUNT(*) > 323;
```

Data Output Messages Notifications		
	rental_rate numeric (4,2)	count bigint
1	4.99	336
2	0.99	341

```
SELECT customer_id, SUM(amount) FROM payment
GROUP BY customer_id
HAVING SUM(amount) >100
ORDER BY SUM(amount) DESC
LIMIT 3;
```

Data Output Messages Notifications		
	customer_id smallint	sum numeric
1	148	211.55
2	526	208.58
3	178	194.61

## ALIAS

Sütunlarımıza geçici olarak isim verebiliriz. Bunun için **AS** anahtar kelimesini kullanırız. Sütunlara isim verebildiğimiz gibi tablolarımıza da geçici isimler verebiliriz. **AS** anahtar kelimesiyle sütunlarımıza isim vererek daha anlamlı hale getiririz.

```
SELECT first_name AS isim, last_name AS soyisim FROM customer;
```

	isim character varying (45)	soyisim character varying (45)
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson

Görüldüğü gibi, `first_name` sütunu ile `last_name` sütunu isimleri değiştirilmiştir.

Sütun isimlerinin kelime sayısı birden fazla olacaksa çift tırnak ("" ) kullanırız.

```
SELECT first_name AS "isim sütunu", last_name AS "soyisim sütunu" FROM customer;
```

	isim sütunu character varying (45)	soyisim sütunu character varying (45)
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson

## CONCAT

Alias konusunda devam edecek olursak; iki farklı sütunu birleştirmemiz mümkündür. **CONCAT** anahtar kelimesi ile bunu yapabiliriz. Oluşturulan bu yeni sütun ismini de **AS** anahtar kelimesi ile geçici olarak veririz.

```
SELECT CONCAT (first_name, ' ', last_name) AS müşteri FROM customer;
```

--arada boşluk olması için tek tırnaklar konuldu

	müşteri text
1	Jared Ely
2	Mary Smith
3	Patricia Johnson

## Tablo Oluşturmak ve Silmek

---

### CREATE

```
CREATE TABLE <tablo_adı> (  
    <sütun_adı> <veri_tipi> <constraint_name>,  
    <sütun_adı> <veri_tipi> <constraint_name>,  
    ....  
);
```

```
CREATE TABLE author (  
    id SERIAL PRIMARY KEY,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    email VARCHAR(100)  
    birthday DATE  
);
```

### INSERT INTO

```
INSERT INTO author (first_name,last_name,email,birthday)
```

### VALUES

```
('Sabahattin','Ali','Sabahattin.Ali@gmail.com','1956-11-06'),  
('Edip','Cansever','Edip.Cansever@gmail.com','1959-10-15'),  
('Oğuz','Atay','Oguz.Atay@gmail.com','1949-05-07');
```

```
CREATE TABLE author2 (LIKE author); --author tablosuyla aynı veri tipinde  
author2 tablosunu oluşturdu
```

```
INSERT INTO author2
```

```
SELECT*FROM author; --author2'ye, author tablosunun hepsini getirdik.
```

```
CREATE TABLE author3 (LIKE author2); --author2 tablosu gibi, author3 tablosu oluşturduk
```

```
SELECT*FROM author3;--author3 tablosunu sorguladık, içinde hiç bi değer yok ama aynı author2 veri tiplerini barındırıyor.
```

```
INSERT INTO author3
```

```
SELECT*FROM author
```

```
WHERE first_name='Sabahattin';--author3 tablosuna sadece 'Sabahattin' isimli satırın olduğu kısmı getirdik.
```

## DROP

```
DROP TABLE author3;--author3 tablosunu siler.
```

```
DROP TABLE IF EXISTS author2;--Silme işleminde hata almamak için IF EXISTS anahtar kelimesi kullanıldı.
```

## Verileri Güncellemek - Silmek

### UPDATE

Verileri güncellemeye başlamadan önce, otomatik olarak bir tablo oluşturalım. Bunun için Google'a "Generate dummy data for sql" yazalım veya direk olarak aşağıdaki linke tıklayalım.

<https://www.mockaroo.com/>

Buradan oluşturacağımız verileri veri tipleri ile birlikte seçeriz.

Formatını SQL olarak belirledikten sonra Preview kısmına tıklarız, buradaki kodların hepsini kopyaladıktan sonra PGAdmin'de çalıştırırız. Sonrasında otomatik olarak bizim için oluşturulmuş tablomuzu görebiliriz.

UPDATE author

SET first\_name = 'Emrah',

last\_name = 'Safa',

email='emrah.safa@gmail.com',

birthday= '1990-05-25'

WHERE id=10;

--id numarası 10 olan satırın özelliklerini değiştirmiş olduk

Query		Query History				
1		SELECT * FROM author;				
Data Output		Messages				
id		first_name				
[PK] integer		character varying (50)				
90	99	Wessie				
99	100	Benn				
100	10	Emrah				
last_name		character varying (50)				
		Bartore				
		Bessom				
		bbessom2r@princeton.edu				
		emrah.safa@gmail.com				
email		character varying (100)				
		[null]				
birthday		date				
		1924-00-29				
		1925-05-15				
		1990-05-25				

### RETURNING

UPDATE author

SET last\_name = '\*\*\*'

WHERE first\_name='Alex'

RETURNING \* ; --Değiştirilenlerin hepsini getirir.

Query Query History

```
1 UPDATE author
2 SET last_name = '***'
3 WHERE first_name='Alex'
4 RETURNING*; --Değiştirilenlerin hepsini getirir.
```

Data Output Messages Notifications



	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	birthday date
1	90	Alex	***	aocrevy2h@kickstarter.com	1987-09-19

## DELETE

**DELETE FROM** author

**WHERE** id>10 -id numarası 10'dan büyük olanları sildi

**RETURNING** \*;

Query Query History

```
1 SELECT * FROM author;
```

Data Output Messages Notifications



	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	birthday date
1	1	Saxe	Itzhak	sitzhak0@qq.com	1993-09-06
2	2	Carrol	Caveau	ccaveau1@arstechnica.com	1990-11-06
3	3	Morna	Nulty	mnulty2@comsenz.com	[null]
4	4	Thorndike	Danilewicz	tdanilewicz3@cpanel.net	1909-07-17
5	5	Elysha	Wheal	ewheal4@indiegogo.com	1901-10-18
6	6	Yorgos	Simond	ysimond5@hostgator.com	1966-05-11
7	7	Shaylyn	Winslet	swinslet6@360.cn	1913-02-09
8	8	Gay	Maryska	gmaryska7@example.com	1949-01-28
9	9	Mireielle	De Beauchemp	mdebeauchemp8@eepurl.com	1905-01-06
10	10	Emrah	Safa	emrah.safa@gmail.com	1990-05-25

### PRIMARY KEY

Bir tabloda bulunan veri sıralarını birbirinden ayırmamızı sağlayan bir kısıtlama (constraint) yapısıdır. O tabloda bulunan veri sıralarına ait bir "benzersiz tanımlayıcıdır".

- Benzersiz (Unique) olmalıdır.
- NULL değerine sahip olamaz.
- Bir tabloda en fazla 1 tane bulunur.

#### Primary Keys



<u>StudentId</u>	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

Yukarıda bulunan görselimizde de gördüğünüz gibi STUDENT tablosunda bulunan **StudentId** sütunu PRIMARY KEY yapısındadır ve her satırı (veri kaydını) diğer satırlardan ayırmamızı sağlar.

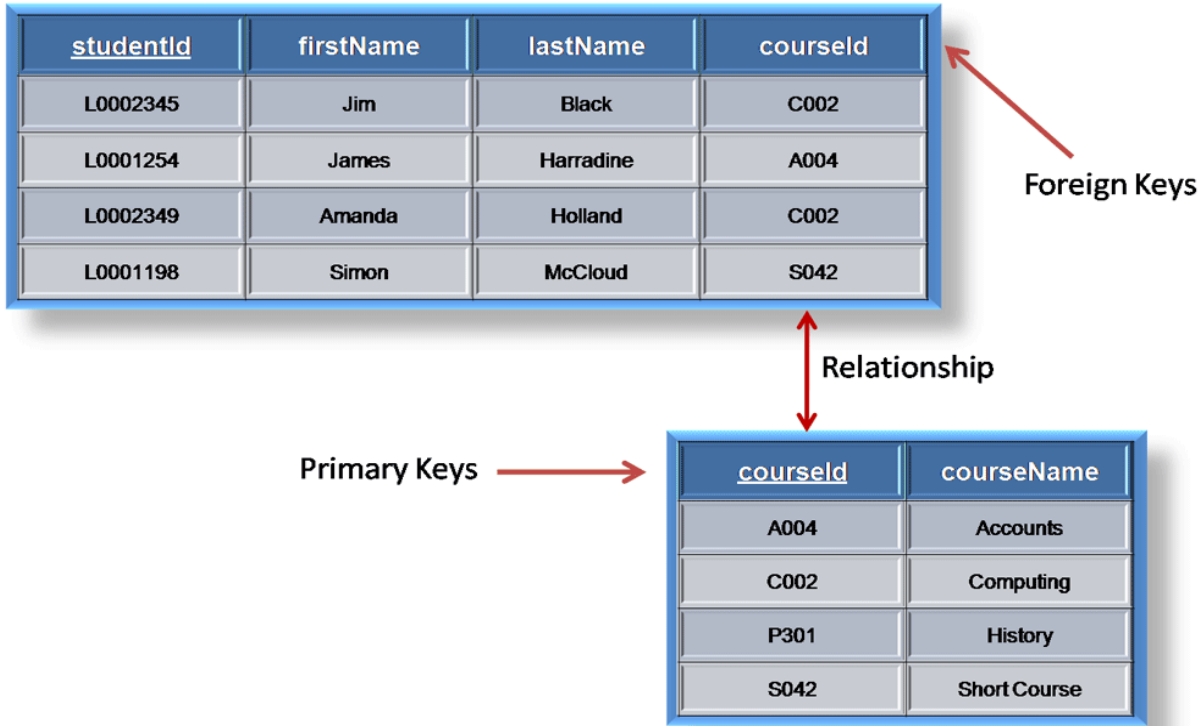


## **FOREIGN KEY**

FOREIGN KEY bir tabloda bulunan herhangi bir sütundaki verilerin genelde başka bir tablo sütununa referans vermesi durumudur, tablolar arası ilişki kurulmasını sağlar.

Bir tabloda birden fazla sütun FK olarak tanımlanabilir.

Aynı sütunun içerisinde aynı değerler bulunabilir.



Yukarıda bulunan görselimizde de gördüğünüz gibi STUDENT tablosunda bulunan courseId sütunu FOREIGN KEY yapısındadır ve başka bir tablo olan "Course" tablosundaki courseId sütununa referans verir.

## Veri Tipleri

---

### Temel Veri Tipleri

- Sayısal Veri Tipleri
- Karakter Veri Tipleri
- Boolean Veri Tipleri
- Date / Time Veri Tipleri

### Sayısal Veri Tipleri

İsim	Range
smallint	-32768 to +32767
integer	-2147483648 to +2147483647
bigint	-9223372036854775808 to +9223372036854775807
decimal	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	6 decimal digits precision
double precision	15 decimal digits precision
smallserial	1 to 32767
serial	1 to 2147483647
bigserial	1 to 9223372036854775807

### Karakter Veri Tipleri

İsim	Tanım
character varying(n), varchar(n)	variable-length with limit
character(n), char(n)	fixed-length, blank padded
text	variable unlimited length

Sınırlı sayıda karakter kullanımı için VARCHAR veya CHAR veri tipleri kullanılır. VARCHAR veri tipi doldurulmayan karakterleri yok sayar, CHAR veri tipi ise doldurulmayan karakterler için boşluk bırakır. Sınırsız karakter kullanımı için ise TEXT veri tipi kullanılır.

### Boolean Veri Tipleri

TRUE, FALSE veya NULL (Bilinmeyen) değerlerini alabilirler.

TRUE: true, yes, on, 1

FALSE: false, no, off, 0

### Zaman / Tarih Veri Tipleri

İsim	Tanım
timestamp [ (p) ] [ without time zone ]	both date and time (no time zone)
timestamp [ (p) ] with time zone	both date and time, with time zone
date	date (no time of day)
time [ (p) ] [ without time zone ]	time of day (no date)
time [ (p) ] with time zone	time of day (no date), with time zone
interval [ fields ] [ (p) ]	time interval

### **NOT NULL**

Birçok durumda bizler herhangi bir sütuna yazılacak olan verilere belirli kısıtlamalar getirmek isteriz. Örneğin yaş sütununda sadece sayısal verilerin olmasını isteriz ya da kullanıcı adı sütununda bilinmeyen (NULL) değerlerin olmasını istemeyiz. Bu gibi durumlarda ilgili sütunda **CONSTRAINT** kısıtlama yapıları kullanılır.

**NULL** bilinmeyen veri anlamındadır. Boş string veya 0 verilerinden farklıdır. Şu şekilde bir senaryo düşünelim bir kullanıcının email hesabı yoksa buradaki veriyi boş string şeklinde düşünebiliriz. Acak eğer kullanıcının maili var ancak ne olduğunu bilmiyorsak bu durumda o veri **NULL** (bilinmeyen) olarak tanımlanabilir.

### **NOT NULL Kullanımı**

Employees şeklinde bir tablomuzu oluşturalım. Tablodaki first\_name ve last\_name sütunlarında bilinmeyen veri istemiyoruz, bu sütunlarda **NOT NULL** kısıtlama yapısı kullanabiliriz.

```
CREATE TABLE Employees (  
    id SERIAL PRIMARY KEY,  
    first_name VARCHAR(100) NOT NULL,  
    last_name VARCHAR(100) NOT NULL,  
    age INTEGER  
);
```

### **ALTER ve NOT NULL**

**ALTER** anahtar kelimesini varolan bir tabloda değişiklik yapmak için kullanılır. Aşağıdaki senaryoda bir sütuna **NOT NULL** kısıtlaması vermek için aşağıdaki söz dizimi yapısı kullanılır.

```
ALTER TABLE <tablo_adi> ALTER COLUMN <sütun_adi>
```

```
SET NOT NULL;
```

## UNIQUE

**UNIQUE** kısıtlaması ile uyguladığımız sütundaki verilerin birbirlerinden farklı benzersiz olmalarını isteriz.

PRIMARY KEY kısıtlaması kendiliğinden **UNIQUE** kısıtlamasına sahiptir.

NOT NULL kısıtlamasında olduğu gibi tablo oluştururken veya ALTER komutu ile beraber tablo oluştuktan sonra da kullanabiliriz.

## UNIQUE Kullanımı

Employees şeklinde bir tablomuzu oluşturalım. Tablodaki email sütununda bulunan verileri **UNIQUE** olarak belirlemek istersek.

```
CREATE TABLE Employees (  ---  
    emaile VARCHAR(100) UNIQUE,  
    ----  
);
```

## ALTER ve UNIQUE

```
ALTER TABLE <tablo_adı> ADD UNIQUE <sütun_adı>
```

Bu arada herhangi bir sütuna **UNIQUE** kısıtlaması getirirsek ve öncesinde **UNIQUE** olmayan verileri kaldırmamız gerekir.

## CHECK

**CHECK** kısıtlaması ile uyguladığımız sütundaki verilere belirli koşullar verebiliriz. Örneğin age (yaş) olarak belirlediğimiz bir sütuna negatif değerler verebiliriz veya web portaline üye olan kullanıcıların yaşlarının 18 yaşından büyük olması gibi kendi senaryolarımıza uygun başka kısıtlamalar da vermek isteyebiliriz.

CHECK kısıtlamasını da tablo oluştururken veya ALTER komutu ile beraber tablo oluştuktan sonra kullanabiliriz.

## CHECK Kullanımı

Employees şeklinde bir tablomuzu oluşturalım. Tablodaki age sütununda bulunan verilerin 18'e eşit veya büyük olmasını istiyoruz.

```
CREATE TABLE Employees ( ---
```

```
    age INTEGER CHECK (age>=18)
```

```
----
```

```
);
```

## ALTER ve CHECK

```
ALTER TABLE <tablo_adı> ADD CHECK (age>=18)
```

## psql Uygulaması-II

Postgre sql veri tabanına bağlanalım.

```
psql -U postgres
```

Veri tabanınıma bağladıktan sonra "testdb" adında yeni bir tablo oluşturalım.

```
CREATE TABLE testdb;
```

```
\c
```

 ile bu tablomuza bağlanalım.

testdb'ye bağlandıktan sonra users adında bir tablo oluşturalım. Bu tablomuzda sırasıyla, id, username ve birthday sütunları olacak. Sırasıyla da bunların isimleri, veri tipleri ve constraint'lerini yazacağız.

```
postgres=# \c testdb
```

```
Şu anda "testdb" veritabanına "postgres" kullanıcısı ile bağlısınız.
```

```
testdb=# CREATE TABLE users(id SERIAL PRIMARY KEY,
```

```
testdb(# username VARCHAR(50) NOT NULL,
```

```
testdb(# birthday DATE);
```

```
CREATE TABLE --Tablomuzun oluştuğuna dair bildirim
```

```
testdb=# \dt --\dt ile tablomuzun içeriğini görelim.
```

**Nesnelerin listesi**

İema	Ad <sup>2</sup>	Veri tipi	Sahibi
------	-----------------	-----------	--------

-----+-----+-----+-----

public	users	tablo	postgres
--------	-------	-------	----------

(1 sat<sup>2</sup>r)

```
testdb=# \d+ ile tablomuzun ayrıntılarını göreürüz.
```

```
\d+ users
```

 ile de tablomuzun parametrelerini, veri tiplerini görürüz.(tablo adı users olduğundan tablo adını da yazdık)

Şimdi bu tablomuza verileri girelim. Yine mockaroo sitesinden verilerimizi tipleri ile birlikte SQL formatında 50 satırlık olacak şekilde oluşturalım.

Verileri Ön İzleme ile (Preview) açıp, kopyaladıktan sonra, terminal ekranımıza sağ click ile kopyalayalım.

Verileri getirmek için aşağıdaki komutu yazalım.

```
testdb=# SELECT * FROM users;
```

```
testdb=# SELECT * FROM users;
 id |  username  | birthday
-----+-----+-----
  1 | Rhiamon    | 2023-01-03
  2 | Nannie     | 2023-07-06
  3 | Donnell    | 2023-09-10
  4 | Edgard     | 2023-10-10
  5 | Gilbert    | 2023-02-04
 10 | Eliza      | 2023-07-07
 11 | Wesley     | 2023-02-06
 12 | Corine     | 2023-03-12
 14 | Arluene    | 2023-04-04
 17 | Zena       | 2023-05-09
 18 | Johannah   | 2023-10-07
 24 | Gay        | 2023-06-02
 27 | Shellysheldon | 2023-08-10
 28 | Jemmie     | 2023-07-12
 30 | Alick      | 2023-01-11
```

## JOIN

---

### INNER JOIN

Birden fazla tabloları birleştirmek için **JOIN** anahtar sözcüğünü kullanırız. Bu birleştirme işlemi kesişim ile mümkündür.

products ve categories adında tablolarımız olsun. İkinci olan categories isimli tablomuzun category\_id bilgisi, products tablosundaki product\_id aracılığı ile (FOREIGN KEY) birleştirebiliriz.

```
SELECT product_name, price, category_name
```

```
FROM products --birleşim olacak tablo
```

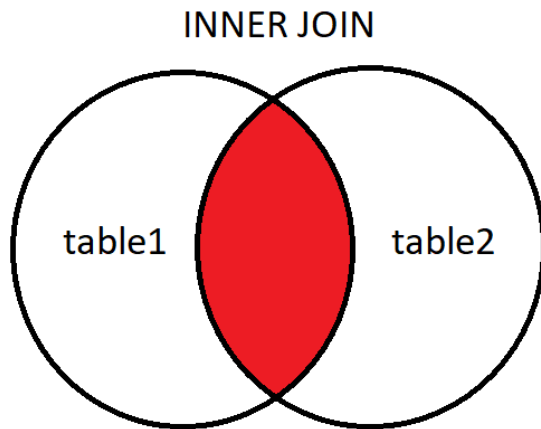
```
INNER JOIN categories ON products.product_id = categories.category_id;
```

INNER JOIN yapısıyla, birbirleri ile kesişimde bulunan tabloları listeleyebiliriz.

```
SELECT <sütun_adı>, <sütun_adı> ... FROM <tablo1_adı> INNER JOIN <tablo2_adı> ON  
<tablo1_adı>.<sütun_adı> = <tablo2_adı>.<sütun_adı>;
```



INNER JOIN - Venn şemasında gösterimi:

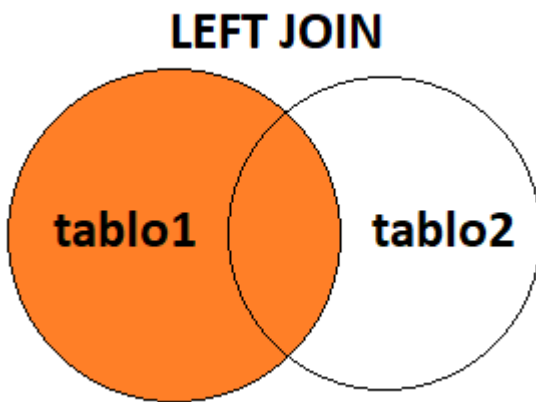


Buradaki kesişim simetriktir, yani, biz sorgumuzda **FROM**'dan sonraki kısma ikinci tabloyu da yazsak bize aynı liste gelecektir.

### LEFT JOIN

Bu sefer, soldaki birinci tablomuzda olan ama sağdaki ikinci tablomuzda olmayan verileri listeleriz. Sağ taraftaki tabloda olmayan veriler null olarak listelenecektir.

LEFT JOIN - Venn şemasında gösterimi:

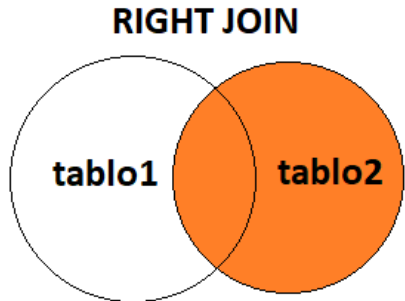


```
SELECT <sütun_adı>, <sütun_adı> ... FROM <tablo1_adı> LEFT JOIN <tablo2_adı> ON  
<tablo1_adı>.<sütun_adı> = <tablo2_adı>.<sütun_adı>;
```

### RIGHT JOIN

Birleştirme işlemi bu kez sağ taraftaki, ikinci tablo üzerinden gerçekleştirilir.

RIGHT JOIN – Venn şemasında gösterimi:



```
SELECT book.title, author.first_name, author.last_name FROM book RIGHT JOIN  
author ON author.id = book.author_id;
```

### FULL JOIN

FULL JOIN yapısıyla, her iki tablodaki tüm verileri getiririz.

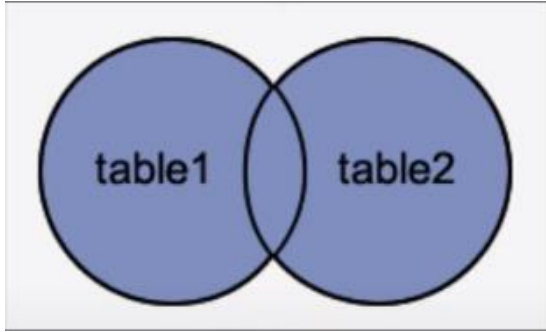
Her iki tabloyu da getirelim:

```
SELECT * FROM products
```

```
FULL JOIN categories ON products.product_id = categories.category_id;
```

	id integer	product_name character varying (50)	price integer	discounted_price integer	product_id integer	id integer	category_name character varying (50)	category_id integer
1	3	Elbise	25	22	1	1	A	1
2	4	Şapka	5	4	2	2	B	2
3	5	Pantolon	100	90	3	3	C	3
4	6	Ayakkabı	125	110	4	4	D	4
5	7	Eldiven	25	20	5	5	E	5

FULL JOIN – Venn şemasında gösterimi:



### UNION

**UNION** anahtar kelimesini iki farklı sorgumuzu birleştirmek için kullanırız. Çift olan / mükerrer olanları da görmek için **UNION ALL** anahtar kelimesini yazarız.

```
SELECT id,product_name FROM products
```

**UNION ALL**

```
SELECT id,category_name FROM categories;
```

Burada dikkat edilmesi gereken husus, birleştirilecek olan iki tablonun sütun sayısı aynı olmalı. Yani, products tablosunda id ve product\_name sütunlarını alırken, categories tablosundan da id ve category\_name sütunlarını alıp birleştirmek istedik. Sütun veri tipleri de aynı olmalıdır, yoksa hata alırız.

### INTERSECT

**INTERSECT** anahtar kelimesi de, iki farklı sorgudaki kesişimleri bize verir.

```
SELECT id,product_name FROM products
```

**INTERSECT**

```
SELECT id,category_name FROM categories;
```

### EXCEPT

**EXCEPT** anahtar kelimesi de, birinci sorguda olup ikinci sorguda olmayan verleri bize listeler.

```
SELECT id,product_name FROM products
```

**EXCEPT**

```
SELECT id,category_name FROM categories;
```

## Alt Sorgular - Subqueries

---

Bir sorgu içerisinde, o sorgunun ihtiyaç duyduğu veri veya verileri getiren sorgulardır.

### Alt Sorgu Kullanımı

**bookstore** veritabanında "Gülün Adı" isimli kitabımızın sayfa sayısı 466 dır. Bu kitaptan daha fazla sayıda sayfası bulunan kitapları aşağıdaki sorgu yardımıyla sıralayabiliriz.

```
SELECT * FROM book  
WHERE page_number > 466;
```

Ancak yukarıdaki sorgumuzda şöyle bir sorun var. 466 ifade statik bir ifade ve biz bu değeri bilmiyor olabiliriz. Bu nedenle buradaki 466 rakamını aşağıdaki sorgu yardımıyla bulabiliriz.

```
SELECT page_number FROM book  
WHERE title = 'Gülün Adı'
```

İşte bu ikinci sorgumuz ilk sorgumuzda bir alt sorgu görevi görebilir. Her iki sorguyu da birleştirelim.

```
SELECT * FROM book  
WHERE page_number >  
(  
SELECT page_number  
FROM book  
WHERE title = 'Gülün Adı'  
);
```

### Any Operatörü

Any ve All operatörleri alt sorgularda sıklıkla kullanılır ve tek bir sütunda bulunan bir değerle bir değer dizisinin karşılaştırılmasını sağlar.

Alt sorgudan gelen herhangi bir değer koşulu sağlaması durumunda TRUE olarak ilgili değerın koşu sağlamasını sağlar. **bookstore** veritabanında yapmış olduğumuz aşağıdaki sorguyu inceleyelim.

```
SELECT first_name, last_name FROM author
WHERE id = ANY
(
  SELECT id
  FROM book
  WHERE title = 'Abe Lincoln in Illinois' OR title = 'Saving Shiloh'
)
```

Yukarıda görmüş olduğunuz gibi alt sorgudan gelebilecek potansiye iki id değeri var, bu id değerinin her ikisi de birbirinden bağımsız olarak ana sorgudaki id sütununda bulunan değerler ile eşleştiği için sorgu sonucunda oluşan sana tabloda id değeri 4 ve 5 olan yazarlara ait first\_name ve last\_name değerlerini göreceğiz.

### ALL Operatörü

Alt sorgudan gelen tüm değerlerin koşulu sağlaması durumunda TRUE olarak döner.

**bookstore** veritabanındaki yine aynı sorguyu inceleyelim.

```
SELECT first_name, last_name FROM author
WHERE id = ALL
(
  SELECT id
  FROM book
  WHERE title = 'Abe Lincoln in Illinois' OR title = 'Saving Shiloh'
)
```

Burada ne söylemiştik alt sorgu tarafından 4 ve 5 id leri gelecek burada eşlik olduğu için aynı anda 4 ve 5 in bu şulu sağlaması olanaksız olduğu için herhangi bir değer dönülmeyecektir.

### Alt Sorgular ve JOIN Kullanımı

Altsorgular ve JOIN kavramları birlikte çok sık kullanılırlar. Aşağıdaki iki senaryoda bu iki yapıyı birlikte kullanacağız.

İlk senaryomuz: **bookstore** veri tabanında kitap sayfası sayısı ortalama kitap sayfası sayısından fazla olan kitapların isimlerini, bu kitapların yazarlarına ait isim ve soyisim bilgileriyle birlikte sıralayınız.

```
SELECT author.first_name, author.last_name, book.title FROM author
```

```
INNER JOIN book ON book.author_id = author.id
```

```
WHERE page_number >
```

```
(
```

```
    SELECT AVG(page_number) FROM book
```

```
);
```

Yukarıdaki sorgumuzda kitaplara ait yazar bilgilerini JOIN kullanarak elde ediyoruz. Ortalama sayfa sayısını da alt sorgudan getiriyoruz.

İkinci senaryomuz: **dvdrental** veritabanında en uzun filmlerin isimlerini aktör isim ve soyisimleriyle birlikte sıralayalım.

```
SELECT actor.first_name, actor.last_name, film.title FROM actor
```

```
JOIN film_actor ON film_actor.actor_id = actor.actor_id
```

```
JOIN film ON film.film_id = film_actor.film_id
```

```
WHERE film.length =
```

```
(
```

```
    SELECT MAX(length) FROM film
```

```
)
```

Burada da görmüş olduğumuz gibi film lerin aktör bilgilerini ikili JOIN yapısı kullanarak elde ediyoruz. En uzun film süresini de alt sorgudan getiriyoruz.