## Kaynakça

http://sql.sitesi.web.tr/

**Murat ELİÇALIŞKAN**
Webmaster

**Düzenleme**

**Burak Kıymaz**
http://www.burakkiymaz.com/

# MySQL Komutları

## SQL Nedir?

"Structured Query Language" yani Yapısal Sorgulama Dili'nin baş harflerinden oluşmuş SQL sorgulamaya dayanan bir veritabanı anlaşma yoludur. Belirli komutlar yoluyla veritabanına veri gönderilmesi, verinin işlenmesi, düzenlenmesi, silinmesi gibi işlemler yapılır.

MySQL sunucusu veritabanlarından oluşur. Her veritabanı çeşitli tablolar barındırır. Bu tablolarsa sütunlardan oluşur. Aşağıda bir tablo görülmekte:

```
+-----+--------+
| id  | isim   |
+-----+--------+
|  1  | Ali    |
|  2  | Sevim  |
|  3  | Gözde  |
+-----+--------+
```

Bu sorgulama komutlarını kullandığınız belli başlı veritabanı sistemleri; MySQL, Microsoft SQL, PostgresSQL, Oracle'dır.

Elbette bu sorguları yaparken veritabanı girişini gerçekleştirmiş olmalısınız. Ayrıca veritabanı kullanıcısı olarak ilgili komutları kullanabilme yetkisine de sahip olmalısınız.

### Neden MySQL?

Dünyada en çok kullanılan, açık kaynak kodlu bir veritabanı yazılımıdır. Ücretsiz olarak faydalanılabilen MySQL ayrıca Linux işletim sistemi kullanan web sunucularının vazgeçilmezidir. Diğer bir açık kaynak kodlu programlama dili olan PHP, MySQL ile işbirliği içerisinde çalışmakta ve kullanıcı kolaylığı sağlamaktadır.

### Neden Veritabanlarına İhtiyaç Duyarım?

Veritabanı, adı üstünde verilerin saklı olduğu yerdir. Bilgiyi hızlı bir şekilde kullanıcılara ulaştırırken, sıralarken, sınıflandırırken ve benzeri işlemlerle dinamik sayfalar oluştururken size lazım olacak temel şeydir.

Şimdi veritabanı ile anlaşma yolu olan SQL dilinin belli başlı komutları üzerinde duralım.

## MySQL Kurmak

İşletim sisteminize göre ve bilgisayarınızın x86/64bit olmasına bağlı olarak aşağıda yer alan yüklemelerden sizin için doğru olanı seçin. Buradaki Binaries program kurulumunu içerir. ZIP Archive olanlar ise kurulumunu kendiniz yapmanız gereken sıkıştırılmış program dosyalarıdır. Eğer bilginiz yoksa Windows Binaries'in bilgisayarınıza uygun olanını yüklemeniz tavsiye edilir.

Eğer fazla bir bilginiz yoksa, kurulum sonrasında karşınıza çıkacak ayarlar bölümünde

sürekli devam ederek Kullanıcı Adı ve Şifre belirtmenizi istediği yere kadar gidin. Burada yazacağınız kullanıcı adı ve şifre, veritabanına bağlanmanız için gerekli anahtardır. Ayrıca MySQL'in windows servislerine eklendiğinden de emin olun. Böylelikle bilgisayar her açıldığında otomatik MySQL sunucusu da açılacaktır.

### Yüklenen Program Neleri İçeriyor?

Öncelikle MySQL sunucusu (Server), verileri saklayan ve işleyen program. MySQL İstemcisi (Client) yardımıyla da sorguları gerçekleştirdiğimiz alanla karşılaşıyoruz. Ayrıca çeşitli kütükler (Library) yardımıyla .NET Framework, C/C++ gibi programlama dilleri yoluyla veritabanına bağlanabileceğimiz ek dosyalarla karşılaşıyoruz.

MySQL sorgularını gerçekleştirirken gelen paketle birlikte MySQL Command Line Client dışında MySQL Front, Navcat gibi programlardan ve web sunucularında yer alan phpMyAdmin gibi eklentilerden de yardım alabiliriz. Bunlar da bir nevi client görevi görecektir.

## Komut Kullanımı

MySQL komutlarını kullanırken dikkat edilecek bir iki nokta vardır. Bunlar ilerleyen derslerde öğreneceğiniz komutların tamamında geçerli değişmez kurallardır.

(Örneklerde yer alan komutları dikkate almayın, bu sayfa genel kurallar içindir.)

1. Bir MySQL komutu mutlaka noktalı virgül ile biter:

```
SELECT * FROM tablo;
```

2. Komut kullanırken satır atlayabilirsiniz, noktalı virgül kullanmadığınız sürece bir önceki satırın devamı sayılacaktır:

```
SELECT * FROM tablo
WHERE id < "25"
ORDER BY id
LIMIT 0, 10;
```

3. Sorgulama yaparken iç sorgular için parantez kullanmalısınız. Sorgularda öncelikle dikkate alınacak yer parantezlerdir:

```
SELECT * FROM tablo1
   WHERE id = ( SELECT mesaj_no FROM tablo2 WHERE mesaj_id = "1" );
```

4. Bir tabloya bir isim atayabilirsiniz. Bunu iki yolla yaparsınız ya direkt olarak atadığınız ismi yazarsınız ya da "as" ekleyerek atadığınız ismi yazarsınız:

```
1. SELECT t.id FROM tablo t;
2. SELECT t.id FROM tablo as t;
```

5. Değişkenler mutlaka tırnak (") ya da tek tırnak (') işareti içine alınır. Fakat hangisiyle başlarsa onunla kapanmak zorundadır:

```
1. SELECT * FROM tablo WHERE isim = "Ali";

2. SELECT * FROM tablo WHERE isim = 'Ali';
```

6. Eğer değişken (") ya da (') içeriyorsa taksim konularak değişkenin kapanmadığı sunucuya bildirilmelidir (yoksa hata verecektir).

```
1. UPDATE tablo
   SET mesaj = 'Bugün Ali\'nin Doğum Günü!' WHERE id = "1";

2. UPDATE tablo
   SET mesaj = "Bana döndü ve \"Neredesin!\" dedi.."
   WHERE id = "1";
```

7. Eğer toplu bir komut dosyasında (SQL File) açıklama yazmak istersek bunu üç şekilde yaparız:

```
# (diyez) Buraya açıklama gelecek... ya da

-- (iki çizgi) Buraya açıklama gelecek... ya da

/* fazla satırlı
   açıklamalar için
   bu kullanılabilir */
```



## Veritabanı Oluşturmak

İçerisinde tabloların bulunacağı bir veritabanı oluştururken **CREATE DATABASE db_adi**komutundan yararlanırız. Burada yer verdiğimiz db_adi, oluşturacağımız veritabanın adını ifade eder:

```
CREATE DATABASE veritabanim;
```

## Veritabanlarını Listeletmek

Yukarıdaki tabloyu oluşturduktan sonra SHOW DATABASES komutu ile veritabanlarını listeletiyoruz. Yarattığımız veritabanı listede görünüyor:

```
SHOW DATABASES;

+----------------+
| Database       |
+----------------+
| veritabanim    |
| mysql          |
| test           |
+----------------+
```

## Veritabanlarını Listeletmek

Yukarıdaki tabloyu oluşturduktan sonra SHOW DATABASES komutu ile veritabanlarını listeletiyoruz. Yarattığımız veritabanı listede görünüyor:

Sıra geldi veritabanımız içerisinde bir tablo oluşturmaya. Bunun için öncelikle kullanacağımız veritabanını seçmeliyiz.

Aşağıdaki komutu kullandığımızda MySQL sunucusu seçili tabloyu bize verecektir. Eğer seçili bir tablo yoksa boş görünecektir.

```
SELECT DATABASE();

+----------------+
| Database()     |
+----------------+
|                |
+----------------+
```

"veritabanim" adındaki veritabanını seçmek için **USE db_adi** komutunu kullanırız:

```
USE veritabanim;
```

Böylelikle kullanacağımız veritabanını seçtik. PHP'de mysql_select_db() fonksiyonu bu iş için işimize yarayacaktır. Şimdi oluşturmak istediğimiz tabloyu hazırlayalım:

| no | isim | yas | email |
|---|---|---|---|
| 1 | Ali | 39 | ali8840@hotmail.com |
| 2 | Sevim | 26 | sevim4420@msn.com |
| 3 | Gözde | 26 | gozde1234@yahoo.com |

Oluşturacağımız tablo 4 sütun içerecektir. Bunlar no, isim, yaş, e-mail verilerinin barınacağı sütunlar. Burada kullandığımız no (id) tüm kayıtlar için ayrı olacak ve o tabloda ilgili veri ile ilgili işlem yapmamızı kolaylaştıracaktır. Bu sebeple no (id) kullanmamız gerekli. Tabiki bu olmak zorunda anlamına gelmiyor.

**CREATE TABLE tablo_adi (sutunlar <ozellikleri>);** komutu yardımıyla bu tabloyu oluşturalım. Sütun adlarını yazarken virgül kullandığımıza (son sütun adı hariç) ve sütun adından sonra o sütunun özelliklerini sıraladığımıza dikkat edin;

```
CREATE TABLE tablom
(
   no int unsigned not null auto_increment primary key,
   isim varchar(45),
   yas int,
   email varchar(60)
);
```

Evet, ilk başta karışık gelebilir. Fakat veri türleri ve tablo özelliği olarak adlandıracağımız (primary key, not null vs) kullanımları öğrendikten sonra kolaylıkla tablo oluşturacaksınız.

Sütun adının hemen yanındaki değer (veri türü):

| | |
|---|---|
| **int** | Veri sayısal bir değerdir. |
| **varchar(n)** | Veri n kadar harf içeren bir alfasayısal değerdir. |

"no" sütununda belirtilen özellikler;

| | |
|---|---|
| **unsigned** | Sayı pozitif bir tamsayı olmak zorundadır. |
| **not null** | Veri tanımsız (null) olamaz. |
| **auto_increment** | Yeni veri eklendiğinde artar. |
| **primary key** | Tablonun ana sütunudur ve her zaman farklıdır. |



### Tabloları Listeletmek

Bunun için **SHOW TABLES** komutu kullanılır.

```
SHOW TABLES;

+---------------------+
| Tables in employees |
+---------------------+
| tablom              |
+---------------------+
```

Oluşturduğumuz tablo listede görünmektedir. Şimdi bu tablo hakkında daha detaylı bilgi alalım. Bunun için **DESCRIBE tablo_adi** komutunu kullanacağız:

```
DESCRIBE tablom;

+-------+------------------+------+-----+---------+----------------+
| Field | Type             | Null | Key | Default | Extra          |
+-------+------------------+------+-----+---------+----------------+
| id    | int(10) unsigned |      | PRI | 0       | auto_increment |
| isim  | varchar(45)      | YES  |     | NULL    |                |
| yas   | int(10)          | YES  |     | NULL    |                |
| email | varchar(60)      | YES  |     | NULL    |                |
+-------+------------------+------+-----+---------+----------------+
```

## Veri Eklemek

Bir tabloya veri eklemek için **INSERT INTO tablo_adi** komutu kullanılır. Bu komut iki şekilde kullanılır. Ya tüm sütunların içereceği değerler aralarına virgül konularak sıralanır ya da öncelikle sütun adları sıralanır sonra VALUES() parantez içinde veriler listelenir. Örnekleri inceleyelim:

```
INSERT INTO tablom ("1", "Ali", "39",
   "a.kececi8440@hotmail.com");

INSERT INTO tablom (isim, yas, email)
   VALUES("Gözde","24","gozde1440@msn.com");
```

Görüldüğü üzere üstteki örnekte sıralı bir şekilde tüm sütunlarda geçecek verileri yazdık. Alttaki örnekte ise "no" sütununu yazmadan diğer üç tanesini yazacağımızı belirttik ve ilgili değerleri girdik. Buna bağlı olarak "no" sütununa sunucu tarafından sıradaki değer verilecektir (2).



## Sorgulama Yapmak

Sorgulama yapmamızı yani bir tablodan belirli koşullara uyan veya tüm kayıtları listelememizi sağlayan komut **SELECT**'dir. Bu **SELECT sütunlar FROM tablo_adı WHERE koşullar**şeklinde kullanılır.

```
SELECT isim, yas FROM tablom;

+-------+-----+
| isim  | yas |
+-------+-----+
| Ali   | 39  |
| Sevim | 24  |
| Gözde | 28  |
+-------+-----+
```

Eğer bir tablodaki tüm sütunları seçeceksek yıldız (*) karakterini sütunları belirttiğimiz SELECT'ten sonraki kısma yazabiliriz:

```
SELECT * FROM tablom;
```

```
+----+-------+-----+----------------+
| id | isim  | yas | email          |
+----+-------+-----+----------------|
| 1  | Ali   | 39  | ali@mail.com   |
| 2  | Sevim | 24  | sevim@mail.com |
| 3  | Gözde | 28  | gozde@mail.com |
+----+-------+-----+----------------+
```

## Koşullara Bağlı Listeleme

Eğer sadece belli koşullara uyan kayıtları listeletmek ve seçmek istiyorsak WHERE koşullarşeklinde komuta ekleme yaparız.

```
SELECT isim, yas, email FROM tablom WHERE id = "1";
```

```
+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Ali   | 39  | ali@mail.com   |
+-------+-----+----------------+
```

Yukarıdaki örnekte id sütunu 1 olan kaydı bize vermesini WHERE id = "1" eklemesiyle belirttik. Sayısal değerlerde büyüktür, küçüktür, küçük eşittir, büyük eşittir kullanabiliriz:

```
SELECT isim, yas, email FROM tablom WHERE yas < "30";
```

```
+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Sevim | 24  | sevim@mail.com |
| Gözde | 28  | gozde@mail.com |
+-------+-----+----------------+
```

Yukarıdaki sorgu bize 30 yaşından küçük üyeleri verdi. Büyük eşittir (>=) ve küçük eşittir (<=) dersek ilgili sayıyı da sorgu sonuçlarına dahil etmiş oluruz.

Sayısal olmayan bir değeri koşul olarak öne sürüyorsak eşittir (=) kullanırız.

```
SELECT isim, yas, email FROM tablom WHERE isim = "Ali";

+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Ali   | 39  | ali@mail.com   |
+-------+-----+----------------+
```

Burada isim = "ali" ya da isim = "ALI" gibi bir kullanım da olabilirdi. Sunucu büyük-küçük harf duyarsız olarak koşula bakacaktır. Eşitsizlik durumunu ise <> ile belirtiriz:

```
SELECT isim, yas, email FROM tablom WHERE yas <> "24";

+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Ali   | 39  | ali@mail.com   |
| Gözde | 28  | gozde@mail.com |
+-------+-----+----------------+
```

Bu sorgu 24 yaşında olmayan üyeleri listelememizi sağladı.

## Kalıp İfadelerle Sorgu (pattern)

Eğer bir sözel verinin sadece belli kriterlere uyanlarını seçeceksek sütun_adı LIKE kalıp koşulunu kullanırız. Kalıp kısmında belirteceğimiz yüzde (%) işareti, orada herhangi bir harf ya da kelime olabileceğini belirtir.

```
SELECT isim, yas, email FROM tablom WHERE isim LIKE "a%";

+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Ali   | 39  | ali@mail.com   |
+-------+-----+----------------+
```

"a%" ifadesi A harfiyle başlayan ve herhangi bir şekilde devam eden kayıtları listelememizi sağladı. Eğer yüzde başta olsaydı:

```
SELECT isim, yas, email FROM tablom WHERE isim LIKE "%m";
```

```
+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Sevim | 24  | sevim@mail.com |
+-------+-----+----------------+
```

Bu ifade sonu isim sütununda sonu m harfiyle biten kayıtları listelememizi sağladı. Yüzde işaretini her iki tarafta da kullanabiliriz. Bu durumda içinde ilgili harf ya da kelime geçen kayıtlar listelenir:

```
SELECT isim, yas, email FROM tablom WHERE isim LIKE "%e%";
```

```
+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Sevim | 24  | sevim@mail.com |
| Gözde | 28  | gozde@mail.com |
+-------+-----+----------------+
```

Yukarıdaki her iki kayıtta e harfi içeriyor. Burada harf ile ilgili örnekler verdik fakat buraya kelime de yazabileceğinizi unutmayın.

```
SELECT isim, yas, email FROM tablom WHERE isim LIKE "%evi%";
```

```
+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Sevim | 24  | sevim@mail.com |
+-------+-----+----------------+
```

## Bağlaçlar (Operatörler)

**WHERE koşul** yapısında birden fazla koşul belirteceksek AND, OR ve NOT bağlaçlarından uygun olanı kullanırız.

**AND** bağlacı, iki koşula da uyan sonuçları listelememizi sağlar.

```
SELECT isim, yas, email FROM tablom
   WHERE isim = "Ali" AND yas = "39";

+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Ali   | 39  | ali@mail.com   |
+-------+-----+----------------+
```

**OR** bağlacı iki durumdan birine uyan kayıtları listeler:

```
   SELECT isim, yas, email FROM tablom
   WHERE isim = "Ali" OR isim = "Sevim";

+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Ali   | 39  | ali@mail.com   |
| Sevim | 24  | sevim@mail.com |
+-------+-----+----------------+
```

**NOT** ifadesi bir bağlaçtan ziyade bir koşulun tersinin gerçekleştiğinde geçerli olacağını söylemektir. Yani eğer var olan bir koşulun önüne NOT eklersek, o koşul gerçekleşmiyorsa doğru olarak kabul edilecektir. Örneğin:

```
   SELECT isim, yas, email FROM tablom
   WHERE isim NOT LIKE "%a%";

+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Sevim | 24  | sevim@mail.com |
| Gözde | 28  | gozde@mail.com |
+-------+-----+----------------+
```

Görüldüğü üzere LIKE teriminin önüne gelen NOT yardımıyla isim sütunundaki değerde A harfi bulunmayanları listelettik.

Koşul alanında kullanacağımız parantez, tıpkı matematikteki gibi öncelikle dikkate alınacaktır. Parantez dışındaki kısımsa parantez ile karşılaştırılır. Örneğin tablomuza göre Adı Ali ya da Sevim olup yaşı 30 dan az olanları listeleteceğimiz bir sorgu yapacak olsaydık:

```
SELECT isim, yas, email FROM tablom
WHERE ( isim = "Ali" OR isim = "Sevim" ) AND yas < 30;
```

```
+-------+-----+----------------+
| isim  | yas | email          |
+-------+-----+----------------|
| Sevim | 24  | sevim@mail.com |
+-------+-----+----------------+
```

### IN ve BETWEEN Kullanımı

Aşağıdaki örnekte görülen bir sorguyu:

```
SELECT * FROM uyeler WHERE id = 3 OR id = 6 OR id = 7;
```

Bu şekilde uzun yazmak yerine IN(...) kullanabiliriz:

```
SELECT * FROM uyeler WHERE id IN(3, 6, 7);
```

Kullanırken parantez içine virgüllerle ayrılarak değerler yazılır. Eğer parantez içinde belirtilen değerleri içermeyenleri seçeceksek **id NOT IN()** kullanabiliriz.

```
SELECT * FROM uyeler WHERE id NOT IN(3, 6, 7);
```

**BETWEEN** (arasında) ise belli sayı aralıklarındaki verileri vermemizi sağlar, lafın gelişi:

```
SELECT * FROM uyeler WHERE id >= 60 AND id <= 100;
```

Yukarıdaki sorgu aşağıdaki ile ifade edilebilir:

```
SELECT * FROM uyeler WHERE id BETWEEN 60 AND 100;
```

Aynı şekilde NOT eklenerek ilgili aralıktaki sayıların seçilmediği sorgular yapılabilir:

```
SELECT * FROM uyeler WHERE id NOT BETWEEN 60 AND 100;
```

Yukarıdaki sorgu da 60-100 arasında olmayan sayıları seçecektir.

### IS NULL ve IS NOT NULL

Bir sütun hiçbir değer içermeyebilir. Bir sütun eğer null (tanımsız) değer alabiliyorsa bunu **WHERE** koşul kısmında koşul olarak sorgulatabiliriz. Örneğin:

```
SELECT * FROM uyeler WHERE email IS NULL;
```

Ya da tanımsız değer değilse koşulunu NOT getirerek yapabiliriz:

```
SELECT * FROM uyeler WHERE email IS NOT NULL;
```

## Sıralama Yapmak

**SELECT sütunlar FROM tablo_adı** ve varsa koşulları **WHERE koşul** şeklinde yazdıktan sonra **ORDER sütun_adı** ile listenin neye göre sıralanacağını belirtiriz.

Aşağıdaki verilere sahip bir "uyeler" tablomuz olduğunu düşünürsek:

```
SELECT * FROM uyeler;
```

```
+----+-------+-----+
| id | isim  | yas |
+----+-------+-----+
| 1  | Ali   | 39  |
| 2  | Gözde | 28  |
| 3  | Sevim | 24  |
| 4  | Ayşe  | 32  |
| 5  | Halil | 48  |
| 6  | Ece   | 24  |
+----+-------+-----+
```

Sıralamanın -farklı bir kural belirtilmemişse- kayıt sırasına göre yapıldığını görebiliriz. Bu listeyi yaşa göre sıralamak istersek sorgunun sonuna **ORDER BY yas** ekleriz:

```
SELECT * FROM uyeler ORDER BY yas;
```

```
+----+-------+-----+
| id | isim  | yas |
+----+-------+-----+
| 3  | Sevim | 24  |
| 6  | Ece   | 24  |
| 2  | Gözde | 28  |
| 4  | Ayşe  | 32  |
| 1  | Ali   | 39  |
| 5  | Halil | 48  |
+----+-------+-----+
```

Dikkat edilirse 24 yaşında iki üye var. Fakat Sevim'in kayıt numarası daha küçük olduğundan yukarıda görünüyor. Bu durumda ikinci bir sıralamada dikkat edilecek husus belirtmek istersek virgül koyarak diğer geçerli sütun adını yazarız:

```
SELECT * FROM uyeler ORDER BY yas, isim;

+----+-------+-----+
| id | isim  | yas |
+----+-------+-----+
| 6  | Ece   | 24  |
| 3  | Sevim | 24  |
| 2  | Gözde | 28  |
| 4  | Ayşe  | 32  |
| 1  | Ali   | 39  |
| 5  | Halil | 48  |
+----+-------+-----+
```

Nitekim bu örnekte eğer "yas" sütunundaki veriler eşitse dikkat edilecek ikinci sütunun "isim" sütunu olacağını bildirdik. Yaptığımız 3 örnekte de sıralamanın sayılarda küçükten büyüğe, harflerde alfabetik olarak listelendiğini gördük. Eğer bu durumun tersini yapmak istiyorsak sütun adından sonra DESC eki getirmemiz yeterlidir. Örneğin yaşı büyükten küçüğe sıralayacak olursak:

```
SELECT * FROM uyeler ORDER BY yas DESC, isim;

+----+-------+-----+
| id | isim  | yas |
+----+-------+-----+
| 5  | Halil | 48  |
| 1  | Ali   | 39  |
| 4  | Ayşe  | 32  |
| 2  | Gözde | 28  |
| 6  | Ece   | 24  |
| 3  | Sevim | 24  |
+----+-------+-----+
```



### Sonuçları Sınırlamak (LIMIT)

SELECT, FROM, WHERE, ORDER gibi kullanacağımız ne varsa kullandıktan sonra en son olarak sorgunun kaç kayıt göstereceğini öğrenmeye geldi. Normalde **LIMIT baslangic, adet**belirtilmemişse ilgili tüm kayıtlar listelenecektir. Aşağıdaki sorguda sadece ilk 3 kaydı göstermesini LIMIT kullanarak sağladık:

```
SELECT * FROM uyeler
ORDER BY yas DESC, isim
LIMIT 0, 3;
```

```
+----+-------+-----+
| id | isim  | yas |
+----+-------+-----+
| 5  | Halil | 48  |
| 1  | Ali   | 39  |
| 4  | Ayşe  | 32  |
+----+-------+-----+
```

LIMIT'te belirttiğimiz ilk değer başlangıç değeridir, kurala uyan sonuçlar bu sıradakinden başlar. Sonraki değer ise kaç adet kayıt gösterileceğidir.

## Çeşitli Fonksiyonlar

Bu sayfada yer verdiğimiz tüm fonksiyonlar SELECT ile FROM arasında yer alan sütunları listelediğimiz alanda kullanılabilirler. Genel olarak sütunlardaki en yüksek, en düşük, toplam değerleri bulmamıza yararlar.

### Benzerleri Ayırmak (DISTINCT)

Bazen veriler arasındaki benzerlikleri kaldırarak sadece benzer olmayanları listeletmek isteyebiliriz. Böyle durumlarda **SELECT DISTINCT** yapısı kullanılır:

Aşağıdaki verilere sahip bir "uyeler" tablomuz olduğunu düşünürsek:

```
SELECT * FROM uyeler;
```

```
+----+-------+----------+
| id | isim  | meslek   |
+----+-------+----------+
| 1  | Hasan | Avukat   |
| 2  | Merve | Öğretmen |
| 3  | Nazım | Öğretmen |
| 4  | Efe   | Polis    |
| 5  | Caner | Polis    |
+----+-------+----------+
```

Ve bu tabloda kaç meslek dalı olduğunu sorgulatmak istiyorsak:

```
SELECT DISTINCT meslek FROM uyeler;

+----------+
| meslek   |
+----------+
| Avukat   |
| Öğretmen |
| Polis    |
+----------+
```

Bu komut ile aynı işlevi gören **GROUP BY sütun_adı** ile de benzerler ayıklanabilir:

```
SELECT meslek FROM uyeler GROUP BY meslek;

+----------+
| meslek   |
+----------+
| Avukat   |
| Öğretmen |
| Polis    |
+----------+
```

### Kayıtların İçerdiği En Düşük ve En Yüksek Değerler

Bir sütundaki kayıtların en düşük değerini bulmak için **MIN(sütun_adı)**, en yüksek değerini bulmak için **MAX(sütun_adı)** fonksiyonlarını kullanırız:

Aşağıdaki verilere sahip bir "uyeler" tablomuz olduğunu düşünürsek:

```
mysql> SELECT * FROM uyeler;

+----+-------+-----+
| id | isim  | yas |
+----+-------+-----+
| 1  | Ali   | 39  |
| 2  | Gözde | 28  |
| 3  | Sevim | 24  |
| 4  | Ayşe  | 32  |
| 5  | Halil | 48  |
| 6  | Ece   | 24  |
+----+-------+-----+
```

Buradaki en düşük yaşı bulmak için:

```
SELECT MIN(yas) FROM uyeler;
```

```
+----+-------+
| MIN(yas)   |
+----+-------+
| 24         |
+------------+
```

En yüksek yaşı bulmak için:

```
SELECT MAX(yas) FROM uyeler;
```

```
+----+-------+
| MAX(yas)   |
+----+-------+
| 48         |
+------------+
```

### Sütunlardaki Değerlerin Toplamını ya da Ortalamasını Bulmak

Yine yukarıdaki tabloya göre herkesin yaşları toplamını hesaplatmak istiyorsak (sütunların içerdiği değerler toplamı) **SUM(sütun_adı)** fonksiyonundan yararlanırız:

```
SELECT SUM(yas) FROM uyeler;
```

```
+----+-------+
| SUM(yas)   |
+----+-------+
| 195        |
+------------+
```

Yaşların ortalamasını hesap etmek isteseydik AVG(sütun_adı) fonksiyonu istediğimiz ortalamayı bize verecekti.

```
SELECT SUM(yas) FROM uyeler;
```

```
+----+-------+
| AVG(yas)   |
+----+-------+
| 32,5       |
+------------+
```

Burada anlattığımız MAX, MIN, SUM, AVG fonksiyonları ayrıca işlem yapmak için de kullanılabilir. Bu durumda sonuçlar yaptığımız işlemin sonucu olarak görünecektir:

```
SELECT SUM(yas) / AVG(yas) FROM uyeler;
```

```
+----+-------+
| AVG(yas)   |
+----+-------+
| 6          |
+------------+
```

### Toplam Kayıt Sayısını Bulmak

Bunun için **COUNT(sütun_adı)** kullanılır:

```
SELECT COUNT(id) FROM uyeler;
```

```
+----+------+
| COUNT(id) |
+----+------+
| 6         |
+-----------+
```

### Değişkenli Yapılar Oluşturmak

MySQL'de **CONCAT(yazilar)** komutu yardımıyla sonuçların istediğimiz bir formatta olmasını sağlayabiliriz.

Aşağıdaki gibi bir "uyeler" tablomuz olsaydı:

```
SELECT * FROM uyeler;
```

```
+----+-------+----------+
| id | isim  | soyad    |
+----+-------+----------+
| 1  | Seda  | Ekinci   |
| 2  | Merve | Demirkan |
| 3  | Cemil | Hikmet   |
+----+-------+----------+
```

Ve eğer amacımız isim ve soyadların birleştiği bir sorgu sütunu oluşturmak olsaydı, şu tarz bir sorgu işimizi görecekti:

```
SELECT CONCAT(isim, " ", soyad) FROM uyeler;

+--------------------------+
| CONCAT(isim, " ", soyad) |
+--------------------------+
| Seda Ekinci              |
| Merve Demirkan           |
| Cemil Hikmet             |
+--------------------------+
```

## Sütunları Adlandırmak

Komut Kullanımı sayfasında belirttiğimiz gibi istersek tablo adlarını ve sütun adlarını adlandırma yoluna gidebiliriz. Aynı anda birden fazla tablo kullanarak sorgu oluşturmak için bize yardımcı olacaktır.

Bu sayfadaki örnekler aşağıdaki "notlar" adlı tabloya göre yapılacaktır:

```
SELECT * FROM notlar;

+----+-------+--------+--------+
| id | isim  | sinav1 | sinav2 |
+----+-------+--------+--------+
| 1  | Ceren | 40     | 60     |
| 2  | Dilek | 50     | 70     |
| 3  | Mert  | 60     | 70     |
| 4  | Murat | 80     | 90     |
| 5  | Oğuz  | 70     | 80     |
| 6  | Sevgi | 30     | 40     |
+----+-------+--------+--------+
```



Aşağıdaki örnekte sinav1 sütunundaki not ortalamasını AVG() fonksiyonu ile aldık ve çıkan sonucu içeren sütunun adını "ortalama" koyduk:

```
SELECT AVG(sinav1) as ortalama FROM notlar;

+----------+
| ortalama |
+----------+
| 55       |
+----------+
```

Şimdi 1. sınavın %40'ı ile 2. sınavın %60 ını alarak final notu oluşturalım ve öğrencileri bu final notuna göre en yüksekten en düşüğe göre sıralayalım:

```
SELECT isim, sinav1, sinav2,
(40 / 100 * sinav1) + (60 / 100 * sinav2) as final_notu
FROM kullanicilar ORDER BY final_notu DESC;
```

```
+-------+--------+--------+------------+
| isim  | sinav1 | sinav2 | final_notu |
+-------+--------+--------+------------+
| Murat | 80     | 90     | 86         |
| Oğuz  | 70     | 80     | 76         |
| Mert  | 60     | 70     | 66         |
| Dilek | 50     | 70     | 62         |
| Ceren | 40     | 60     | 52         |
| Sevgi | 30     | 40     | 36         |
+-------+--------+--------+------------+
```

Görüldüğü üzere "final_notu" adlı sütun belirttiğimiz gibi ortaya çıktı, tabloya ait bir öğe olmamasına karşın adlandırarak sonuç üzerinden işlem yapabildik.

## Veri/Kayıt Güncellemek

Tablolardaki kayıtları güncellerken **UPDATE tablo_adı SET sütun_adı = deger WHERE koşul** komutundan yararlanırız.

Örneğin aşağıdaki tablodaki "Ali" adlı üyenin yaşını 40 yapalım:

```
SELECT * FROM uyeler;
```

```
+----+-------+-----+----------------+
| id | isim  | yas | email          |
+----+-------+-----+----------------|
| 1  | Ali   | 39  | ali@mail.com   |
+----+-------+-----+----------------+
```

```
UPDATE uyeler SET yas = 40 WHERE isim = "Ali";
SELECT yas FROM uyeler WHERE isim = "Ali";
```

```
+----+-------+-----+----------------+
| id | isim  | yas | email          |
+----+-------+-----+----------------|
| 1  | Ali   | 40  | ali@mail.com   |
+----+-------+-----+----------------+
```

Komutun **SET sütun_adı = deger** kısmında birden fazla veri değişikliği yapılabilir. Bunun için aralara virgül konularak sütun_adı = deger kısmı tekrarlanır. Örneğin:

```
UPDATE uyeler SET yas = 40, email = "ali@yenimail.com"
WHERE id = "1";
```

**NOT:** Dikkat edilirse sayısal ifadeler tırnak içine alınmadan yazılabilmekte. Fakat harflerden oluşan değerler boşluk içerebilir. Bu da komutun çalışmamasını sağlar. Bu nedenle her zaman için tırnak işareti kullanmak yararınıza olacaktır. Ayrıca bknz. Komut Kullanımı.

Sayısal değer içeren sütunlarda toplama, çıkarma gibi dört işlem yapılabilir. Örneğin Ali'nin yaşını 1 arttırmak isteseydik aşağıdaki komutu da kullanabilirdik:

```
UPDATE uyeler SET yas = yas + 1 WHERE isim = "Ali";
```

Böylelikle yas değeri yas değerinin 1 fazlası olarak kaydedilecek ve 39, 40 olacaktır.

## Tarihlerle İşlem

MySQL'deki veri türlerinden biri de **date** türüdür. Tablo yaratırken kullanabileceğiniz bu veri türü üzerinden işlemler yapmanız gerekebilir.

Aşağıdakine benzer bir tablo yaratalım:

```
+----+-------+------------+
| id | isim  | dogum_tar  |
+----+-------+------------+
| 1  | Murat | 22-02-1984 |
+----+-------+------------+

# Tabloyu oluşturuyoruz...

CREATE TABLE uyeler (
   id int(5) unsigned primary key not null auto_increment,
   isim varchar(45),
   dogum_tar date
);

# Veriyi giriyoruz...

INSERT INTO uyeler (isim, dogum_tar) VALUES("Murat", "1984-02-22");
```

Tarih formatı her zaman için YYYY-AA-GG (Y: Yıl, A: Ay, G: Gün) şeklinde yazılır. Bu kurala uymayan sorgular geçersiz kabul edilir.

### Tarih Türüne Özel Koşullar

Üyeler arasından sadece 2. ayda doğmuş olanları almak isteyebiliriz. Bunun için MONTH() kullanılır:

```
SELECT * FROM uyeler WHERE MONTH(dogum_tar) = 2;
```

Belli bir yılda doğmuş olanlar için YEAR() kullanılır:

```
SELECT * FROM uyeler WHERE YEAR(dogum_tar) = 1984;
```

Ayın 22'sinde doğanları bulmak isteseydik:

```
SELECT * FROM uyeler WHERE DAYOFMONTH(dogum_tar) = 22;
```

MySQL'de CURRENT_DATE o andaki zamanı verir. Bunun üzerinden de işlem yapılabilir:

```
SELECT * FROM uyeler WHERE MONTH(dogum_tar) = MONTH(CURRENT_DATE);
```

## Veri / Tablo / Veritabanı Silmek

### Kayıt Silmek

Tablolarda yer alan kayıtların tamamını ya da **WHERE koşul** ile belirttiğimiz koşullara uyanları silmek için **DELETE FROM tablo_adı** komutunu kullanırız:

```
DELETE FROM uyeler WHERE yas < 18;
```

Yukarıdaki sorgu ile tablomuzda "yas" sütunundaki değer 18'in altında olan tüm kayıtlar silinecektir.

Eğer **WHERE** belirtmemiş olsaydık. İlgili tablodaki tüm veriler silinecekti.

```
DELETE FROM uyeler;
```



### Tablo Silmek

Bunun için **DROP TABLE tablo_adı** kullanılır. Tablo tamamen ortadan kalkar ve içerisindeki tüm veriler de silinir:

```
DROP TABLE uyeler;
```

### Veritabanı Silmek

Bir veritabanını kaldırmak içerisinde barındırdığı tüm tabloları ve kayıtları da silmek anlamına gelmektedir. Bunun için de **DROP DATABASE veritabanı_adı** kullanılır:

```
DROP DATABASE veritabanim;
```

## Birden Fazla Tablo Sorgulama

**NOT:** Eğer bir önceki bölümdeki dersler yararlı geldiyse ve sorun yaşamadan komut kullanımını öğrendiyseniz bu konular MySQL'de çok daha geniş bilgiler edinmenizi sağlayacaktır.

Tablolarımız birbiriyle ortaklaşa sonuç üretmek zorunda kalabilir. Örneğin bir forumda üyeler ve mesajlar adlı iki tablomuz olsaydı mesaj hangi üye tarafından atılmış bilgisini direkt mysql'den almamız gerekebilirdi. Eğer bu iki tablo arasında bir bağlantıdan yararlanabiliyorsak (benzerlik içeren bir sütun) bunu yapmamız mümkün.

İşte böyle durumlar için sütunları adlandırdığımız gibi tabloları adlandırıp birden fazla tablo üzerinde işlem yapabilme şansına sahibiz. Bunu ayrıca bir sonraki derste göreceğiniz `INNER/LEFT/RIGHT JOIN` komutu ile de yapabilirsiniz.

Şimdi tabloları adlandırarak aşağıdaki iki tabloyu tek bir sorguda kullanalım:

```
SELECT * FROM uyeler;

+----+-------+
| id | isim  |
+----+-------+
| 1  | Seda  |
| 2  | Merve |
| 3  | Cemil |
+----+-------+

SELECT * FROM mesajlar;

+----+--------+---------------+
| id | uye_id | mesaj         |
+----+--------+---------------+
| 1  | 1      | Selamlar      |
| 2  | 3      | Nabersiniz?   |
| 3  | 3      | Kimse var mı? |
| 4  | 2      | Merhaba       |
+----+--------+---------------+
```

Görüldüğü üzere "mesajlar" tablosunda mesajın hangi üye tarafından gönderildiği "uye_id" sütununda sayısal olarak belirtilmiş. Şimdi sorguların üye adı - mesaj eşleştirmesiyle gelmesini sağlayalım:

```
SELECT t1.isim, t2.mesaj
FROM uyeler t1, mesajlar t2
WHERE t1.id = t2.uye_id;
```

```
+-------+---------------+
| isim  | mesaj         |
+-------+---------------+
| Seda  | Selamlar      |
| Cemil | Nabersiniz?   |
| Cemil | Kimse var mı? |
| Merve | Merhaba       |
+-------+---------------+
```

Sonucun yukarıdaki şekilde geldiğini göreceğiz. Burada **FROM tablolar** kısmında uyeler adlı tabloyu t1 adıyla, mesajlar adlı tabloyu t2 adıyla nitelendirdiğimizi sunucuya bildirmiş olduk. Böylelikle ilgili tablonun sütunlarını sanki tek tabloymuş gibi işleme aldık. FROM tablolar kısmında kullanacağımız tabloların yanına onu temsilen bir kelime yazarız ve aralara virgül koyarak bunlara yer veririz.

**WHERE koşul** kısmında benzerliğe yer verdik. Çünkü bu iki tablo temelde birbirinden ilgisiz tablolar olabilir. Bu durumda bir özelliğe göre eşleştirme yapmamız gerekli. İşte mesajlar kısmında uye_id'e yer vermemizin sebebi de bu.

Bu tarz sorgularda sütun adlarından önce mutlaka tablonun adını getirin, çünkü bir benzerlik durumunda istenilen sonucu alamayabilirsiniz. Örneğin: t1.isim (uyeler tablosunun isim sütunu).

**NOT:** Adlandırma yaparken **FROM uyeler as t1, mesajlar as t2** yapısını da kullanabilirsiniz.

## Sonuca Başka Bir Tabloyu Katmak

Birden fazla tablo ile sorgulama yapmanın diğer bir yolu JOIN kullanmaktır. Bu komutla birlikte ayrıca INNER JOIN, OUTER JOIN, LEFT JOIN ve RIGHT JOIN komutları da bu derste anlatılacaktır.

Bu sayfadaki tüm örneklerde aşağıdaki tablo dikkate alınacaktır:

```
SELECT * FROM demo_people;
```

```
+------------+--------------+------+
| name       | phone        | pid  |
+------------+--------------+------+
| Mr Brown   | 01225 708225 |    1 |
| Miss Smith | 01225 899360 |    2 |
| Mr Pullen  | 01380 724040 |    3 |
+------------+--------------+------+
```

```
SELECT * FROM demo_property;
```

```
+------+------+----------------------+
| pid  | spid | selling              |
+------+------+----------------------+
|    1 |    1 | Old House Farm       |
|    3 |    2 | The Willows          |
|    3 |    3 | Tall Trees           |
|    3 |    4 | The Melksham Florist |
|    4 |    5 | Dun Roamin           |
+------+------+----------------------+
```

Yukarıda yer alan ilk tablo "demo_people" adındadır ve müşterileri içerir. Alttaki tablo ise "demo_property" adında olup satış ilanlarını içerir.

İlk olarak sadece JOIN kullanarak müşteri - telefon numarası - sattığı emlak sütunlarını bir araya getirelim:

```
SELECT name, phone, selling
FROM demo_people join demo_property
on demo_people.pid = demo_property.pid;
```

```
+-----------+-------------+----------------------+
| name      | phone       | selling              |
+-----------+-------------+----------------------+
| Mr Brown  | 01225 708225 | Old House Farm       |
| Mr Pullen | 01380 724040 | The Willows          |
| Mr Pullen | 01380 724040 | Tall Trees           |
| Mr Pullen | 01380 724040 | The Melksham Florist |
+-----------+-------------+----------------------+
```



**JOIN tablo_adı ON koşul** şeklinde bir kullanım dikkatinizi çekmiştir. Burada bir önceki derste olduğu gibi müşteri numarası (pid) benzerliğinden yola çıkarak iki tabloyu birleştirdik.

**LEFT JOIN** kullanarak bu örneği yapsaydık fazladan 1 sonuç çıktığını görecektik:

```
SELECT name, phone, selling
FROM demo_people left join demo_property
on demo_people.pid = demo_property.pid;
```

```
+------------+--------------+----------------------+
| name       | phone        | selling              |
+------------+--------------+----------------------+
| Mr Brown   | 01225 708225 | Old House Farm       |
| Miss Smith | 01225 899360 | NULL                 |
| Mr Pullen  | 01380 724040 | The Willows          |
| Mr Pullen  | 01380 724040 | Tall Trees           |
| Mr Pullen  | 01380 724040 | The Melksham Florist |
+------------+--------------+----------------------+
```

Bu örnekte eşleşmeyen kayıtlarda en soldaki tabloda görülmektedir (Miss Smith).

**RIGHT JOIN** kullandığımızda bu kez sağ taraftaki tabloda eşleşmeyen kayıtları göreceğiz:

```
select name, phone, selling
from demo_people right join demo_property
on demo_people.pid = demo_property.pid;
```

```
+-----------+--------------+----------------------+
| name      | phone        | selling              |
+-----------+--------------+----------------------+
| Mr Brown  | 01225 708225 | Old House Farm       |
| Mr Pullen | 01380 724040 | The Willows          |
| Mr Pullen | 01380 724040 | Tall Trees           |
| Mr Pullen | 01380 724040 | The Melksham Florist |
| NULL      | NULL         | Dun Roamin           |
+-----------+--------------+----------------------+
```

**INNER JOIN**, LEFT JOIN örneğinde yaptığımız gibi bir sonuç verecekti. **OUTER JOIN** ise LEFT ve RIGHT kullanarak oluşturduğumuz sorgudan sonra istemediğimiz verileri ayıklamak için kullandığımız komuttur.

## Matematiksel İşlemler

Bir matematiksel işlem yapmak için **SELECT islem** yapılabilir. Örnekte geçen % mod anlamına gelmektedir ve bölümden kalanı gösterir:

```
SELECT 87 % 9; /* Sonuç 6 çıkacaktır */
```

Yukarıdaki mod işlemini **MOD(x, y)** fonksiyonunu kullanarak da yapabiliriz.

```
SELECT MOD(37, 13); /* Sonuç 11 çıkacaktır */
```

Mutlak değer sorgusu için ABS(x) fonksiyonunu kullanırız:

```
SELECT ABS(-4.5); /* Sonuç 4.5 çıkacaktır */
```

Bir sayının pozitif, negatif ya da sıfır olduğunu SIGN(x) fonksiyonu ile bulabiliriz:

```
SELECT SIGN(-34); /* Sonuç -1 çıkacaktır */
```

Bir sayının üslü değerini bulmak için POWER(x, y) kullanılabilir:

```
SELECT POWER(4, 3); /* 4 üssü 3 = sonuç 64 çıkacaktır */
```

Bir sayının kare kökü için SQRT(x) kullanılabilir:

```
SELECT SQRT(9); /* Sonuç 3 çıkacaktır */
```

Ondalıklı bir sayıyı yuvarlamak için ROUND(x) kullanılır:

```
SELECT ROUND(12.4);
/* Sonuç 12 çıkacaktır, 12.51 olsaydı 13 olurdu. */
```

Ondalıklı bir sayının alt tam sayısı için FLOOR(x), üst tam sayısı için CEILING(x) kullanılabilir.

```
SELECT FLOOR(36.6); /* Sonuç 36 çıkacaktır */

SELECT CEILING(36.6); /* Sonuç 37 çıkacaktır */
```

## Veri Türleri

Sütunlar çeşitli veri türlerine bağlı olarak veri saklarlar. Örneğin sayısal bir değer olabileceği gibi harflerden oluşan bir değere de sahip olabilirler. İşte MySQL'in veri türleri de bir tablo oluştururken ya da daha sonradan tablonun sütun türünü değiştirirken kullandığımız kodlardır.

Genel olarak 3 veri türünden bahsedebiliriz:

- Integer (Sayısal)
- Text (Alfasayısal)
- Date (Tarih)

### Sayısal Değer İçerenler (INTEGER)

Bunlar sadece rakamdan oluşan sayısal değerler barındırabilirler. Hepsi UNSIGNED ve AUTO_INCREMENT özelliği alabilirler. Bunları tanıyalım:

**TINYINT**, 0 ile 255 arasında sayı barındırabilirler. Eğer UNSIGNED özelliği belirtilmişse -128 ile 127 arasında değerleri depolarlar.
**SMALLINT**, 0 ile 65535 arasında sayı değerlerini barındırabilirler. Eğer UNSIGNED belirtilmişse -32768 ile 32767 arasındaki değerleri alırlar.
**MEDIUMINT**, 0 ile 16777215 arasındaki sayı değerlerini barındırabilirler. Eğer UNSIGNED belirtilmişse -8388608 ile 8388607 arasındaki değerleri alırlar.
**INT**, 0 ile 4294967295 arasındaki sayıları depolarlar. UNSIGNED kullanılmışsa -2147683648 ile 2147683647 arasındaki değerler alt ve üst limitleri olur.
**BIGINT**, -9223372036854775808 ile 9223372036854775807 arasındaki sayıları depolayabilirler.
**FLOAT**, pozitif ondalık sayıları depolar.
**DOUBLE**, negatif ve pozitif ondalık sayıları depolar.
**DECIMAL**, 10'luk değerleri depolar.

İhtiyaca göre sayısal değer içerek sütunlarda bu veri türleri kullanılabilir.

### Alfasayısal (Sözel) Veriler (VARCHAR, TEXT)

Barındıracağı harf/veri uzunluğuna göre belirlenirler. Bunlar karakterleri depolarlar. Aşağıda parantez ile belirtilen yere bir sayı gelir. Örneğin VARCHAR(10) dersek bu 10 karakter barındırabileceğini anlatır, (abcdefghij) gibi..

**CHAR(x)**, x kadar karakter barındırır (x en fazla 255 olabilir).
**VARCHAR(x)**, x kadar karakter barındırır (x en fazla 255 olabilir).
**TINYTEXT**, kısa yazılar.
**MEDIUMTEXT**, orta uzunlukta yazılar.
**TEXT**, uzun yazılar.
**LONGTEXT**, çok uzun yazılar.

### Tarih İçeren Veriler (DATE)

Format hiçbir zaman değişmemekle birlikte veriler tıpkı alfasayısal veriymiş gibi işlenir ve ancak belli fonksiyonlarla sorgulanabilir.

**DATE**, YYYY-AA-GG formatıyla tarihi depolar.
**TIME**, hh:mm:ss formatıysa saati depolar.
**DATETIME** gün önce başlamak üzere tarih ve saati depolar.
**TIMESTAMP** sayısal bir değer olarak saati işler.
**YEAR** yılı depolar.
Bu konu ayrıca Tarihlerle İşlem sayfasında anlatılmıştır.

## Tüm Komutlar

Bu sitede yer verdiğimiz tüm komutlar aşağıda listelenmiştir:
SQL programlama dilinde kullanılan tüm kodlar aşağıda listelenmiştir:



| Komut | Açıklama |
|---|---|
| # açıklama | Açıklama eklemek. |
| -- açıklama | Açıklama eklemek. |
| /* açıklama */ | Açıklama eklemek. |
| CREATE DATABASE | Veritabanı oluşturmak. |
| CREATE TABLE | Tablo oluşturmak. |
| DROP DATABASE | Veritabanı silmek. |
| DROP TABLE | Tablo silmek. |
| SELECT | Bir sorgu gerçekleştirmek. |
| DELETE FROM | Veri silmek. |
| INSERT INTO | Veri eklemek. |
| UPDATE | Veri güncellemek. |
| DESCRIBE | Tablo detaylarını öğrenmek. |