SELECT

Sorgu (Query)

SQL komutlarını içeren sorgu cümleleridir.

Üzerine ilk konuşacağımız SQL komutu SELECT komutudur. SELECT en çok kullanılan SQL komutudur ve veritabanından belirtilen sütunlardaki verileri çekmemizi sağlar. Ayrıca SELECT komutunu çoğunlukla diğer SQL komutlarıyla birlikte kullanırız.

SELECT Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>;

Eğer tablodaki tüm sütunlardaki verileri çekmek istersek asteriks \* karakterinden faydalanırız.

SELECT \*

FROM <tablo\_adı>;

SQL komutlarının büyük harf - küçük harf duyarlılıkları yoktur. (Case Insensitive)

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.

WHERE Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>

WHERE <koşul>;

Eğer tablodaki tüm sütunlardaki verileri çekmek istersek asteriks \* karakterinden faydalanırız.

SELECT \*

FROM <tablo\_adı>;

WHERE Örnek Kullanım

SELECT title, replacement\_cost

FROM film

WHERE replacement\_cost = 14.99;

BETWEEN ve IN

BETWEEN

Aşağıdaki sorgumuzda **AND** mantıksal operatörü yardımıyla **film** tablosunda bulunan verilerimizi uzunluğu 140 tan küçük eşit **VE** 100 den büyük eşit olmak üzere sıralıyoruz.

SELECT \*

FROM film

WHERE length >= 100 AND length <= 140;

Burada temel olarak yaptığımız belirli aralıkta bulunan verileri sıralamak. Bunun **BETWEEN ... AND** yapısını kullanarak da yapabiliriz.

BETWEEN AND Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>

WHERE <koşul>;

BETWEEN Örnek Kullanım

SELECT \*

FROM film

WHERE length BETWEEN 100 AND 140; -- WHERE length >= 100 AND length <= 140 ifadesi ile aynı sonucu verir.

Burada dikkat edilmesi gereken nokta 100 ve 140 sınır değerleri aralığa dahildir.

IN

Şöyle bir senaryo düşünelim, yine film tablosundan uzunluğu 30, 60, 90 veya 120 dakikaya eşit olan verileri sıralayalım.

SELECT \*

FROM film

WHERE length = 30 OR length = 60 OR length = 90 OR length = 120;

sorgusuyla verileri aldık ancak burada şöyle bir sorunumuz var peki 4 farklı değer için değil 14 farklı değer için bu sorgumuzu gerçekleştirmek için 14 ayrı OR mantıksal operatörü kullanmamız gerekirdi. Bunun yerine istenilen değerleri liste haline getirip **IN** anahtar kelimesiyle kullanabiliriz.

IN Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>

WHERE <sütun\_adı> IN (değer1, değer2, ...);

IN Örnek Kullanım

SELECT \*

FROM film

WHERE length IN (30,60,90,120);

LIKE ve ILIKE

Aşağıdaki sorgumuzda **actor** tablomuzda bulunan tüm sütunlardaki verileri first\_name sütununda ki değeri 'Penelope' olmak üzere getiriyoruz.

SELECT \*

FROM actor

WHERE first\_name = 'Penelope';

Ancak bizler bazı durumlarda bu şekilde tam eşleşme değil belirli şablonlara uyan koşulların sağlanmasını isteriz. Örneğin aşağıdaki sorgumuzda first\_name sütunun 'Penelope' değerine eşit olmasını değil, ilk harfin 'P' olması koşulunu sağlar. Bunun için **LIKE** operatörünü kullanırız.

SELECT \*

FROM actor

WHERE first\_name LIKE 'P%';

Burada kullanılan % karakteri sıfır, bir veya daha fazla karakteri temsil eder ve Wildcard olarak isimlendirilir. Bir diğer wildcard karakteri \_ karakteridir ve bir karakteri temsil eder.

LIKE Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>

WHERE <sütun\_adı> LIKE <şablon>;

ILIKE operatörü LIKE operatörünün case - insensitive versiyonudur.

DISTINCT ve COUNT

DISTINCT

Şimdiye kadar yaptığımız SQL sorgularında genellikle verileri belirli koşullar altında sıraladık. Dikkat ettiyseniz bir çok durumda aynı sütün içerisinde birbirinin aynı olan veriler ile karşılaştık. Örneğin **dvdrental** veritabanı içerisinde bulunan **film** tablosundaki replacement\_cost, rental\_rate gibi sütunlar birbirini tekrar eden verilerden oluşmaktadır. Bazı durumlarda bir sütun içerisinde bulunan farklı değerleri görmek isteriz.

SELECT DISTINCT rental\_rate

FROM film;

sorgusu bize rental\_rate sütununda bulunan birinden farklı 2.99, 0.99, 4.99 verilerini gösterir.

SELECT DISTINCT Söz Dizimi

SELECT DISTINCT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>;

COUNT

COUNT **aggregate** fonksiyonu ilgili sorgu sonucunda oluşan veri sayısını bildirir. Aşağıdaki sorguda

ismi 'Penelope' olan aktörleri sıralıyoruz.

SELECT \*

FROM actor

WHERE first\_name = 'Penelope';

ancak veri sayısını bulmak istersek **COUNT** fonksiyonunu kullanırız.

SELECT COUNT(\*)

FROM actor

WHERE first\_name = 'Penelope';

Yukarıda da belirttiğimiz gibi COUNT fonksiyonu ile sorgu sonucunda ortaya verileri sayıyoruz. Bu nedenle COUNT(\*) veya COUNT(sütun\_adı) aynı sonucu verir.

ORDER BY

ORDER BY anahtar kelimesi sayesinde bizler verilerimizi herhangi bir sütunda bulunan değerlere göre azalan veya artan bir şekilde sıralayabiliriz.

ORDER BY Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ...

FROM <tablo\_adı>

ORDER BY <sütun\_adı>, <sütun\_adı>, ... ASC|DESC;

ORDER BY Örnek Kullanım

SELECT \*

FROM film

ORDER BY title (ASC);

Bu sorgumuzda **dvdrental** veritabanında bulunan **film** tablosundaki tüm sütunları **title** sütununda bulunan verilere göre artan (ASC) şeklinde sıralıyoruz.

ASC sıralama varsayılan olduğu için ayrı bir şekilde yazılması zorunluluğu yoktur ancak sorguyu belirginleştirmesi açısından genelde yazılır.

SELECT \*

FROM film

ORDER BY title ASC length DESC;

Sıralama birden fazla sütuna göre de yapılabilir. Yukarıdaki örneğimizde sıralama **title** sütununa göre artan **length** sütununa göre azalan şeklinde yapılıyor.

SELECT \*

FROM film

WHERE title LIKE 'A%'

ORDER BY title ASC length DESC;

Yukarıdaki örneğimizde de gördüğümüz gibi sıralama işlemi, koşuldan sonra yazılır.

LIMIT ve OFFSET

LIMIT

Şimdiye kadar yaptığımız SQL sorgularında genellikle verilerin tamamını belirli koşullar altında sıraladık. Bazı durumlarda ise koşullarımızı sağlayan verilerin tamamını değil belirli sayıda olanlarını sıralamak isteriz, bunun için **LIMIT** anahtar kelimesini kullanırız.

Şöyle bir senaryo üzerine düşünelim. **dvdrental** veritabanında bulunan **film** tablosundan B ile başlayan filmleri uzunluklarına göre en uzun olan 10 filmi sıralayalım.

SELECT \*

FROM film

WHERE title LIKE 'B%'

ORDER BY length DESC

LIMIT 10;

Yukarıdaki sorgumuzda da görmüş olduğunuz gibi önce koşullamayı, sonra gruplamayı en son ise **LIMIT** kullanarak istediğimiz veri sayısını belirttik.

OFFSET

Bazı durumlarda sonuç olarak gördüğümüz veri grubu içerisinden bazılarını "pass" geçmek isteriz. Yukarıdaki senaryomuzu tekrar düşünelim, **dvdrental**veritabanında bulunan **film** tablosundan B ile başlayan filmleri uzunluklarına göre sıralayalım ancak en uzun 6 filmi "pass" geçelim ve sonrasındaki 4 filmi sıralayalım. Bu durumda **LIMIT** 4 ve **OFFSET** 6 olacak.

SELECT \*

FROM film

WHERE title LIKE 'B%'

ORDER BY length DESC

OFFSET 6

LIMIT 4;

Aggregate Fonksiyonlar - MIN, MAX, SUM, AVG

Aggregate fonksiyonları yardımıyla bizler veri kümelerimizden sonuçlar çıkarabiliriz. Ne demek istiyorum? Şu senaryoları düşünelim.

* Toplam kaç adet müşterimiz var?
* Elimizde bulunan filmlerin ortalama uzunluğu nedir?

Bu şekilde belirli veri kümelerinden tek bir sonuç çıkarmak için aggregate fonksiyonları kullanırız.

Örnek Kullanımlar

AVG fonksiyonunu kullandığımız sayısal değerlerden oluşan sütunun ortalama değerini alırız.

SELECT AVG(length)

FROM film;

sorgusu sayesinde **film** tablosunda bulunan **length** sütunundaki değerlerin ortalamasını alırız. SUM fonksiyonunu kullandığımız sayısal değerlerden oluşan sütunun toplam değerini alırız.

SELECT SUM(length)

FROM film;

sorgusu sayesinde **film** tablosunda bulunan **length** sütunundaki değerlerin toplamını alırız. MAX fonksiyonunu kullandığımız sayısal değerlerden oluşan sütunun en yüksek değerini alırız.

SELECT MAX(length)

FROM film;

sorgusu sayesinde **film** tablosunda bulunan **length** sütunundaki değerlerin en yüksek değerini alırız. MIN fonksiyonunu kullandığımız sayısal değerlerden oluşan sütunun en düşük değerini alırız.

SELECT MIN(length)

FROM film;

sorgusu sayesinde **film** tablosunda bulunan **length** sütunundaki değerlerin en düşük değerini alırız.

GROUP BY

Bizler şimdiye kadar olan sorgularımızın tamamında sorguları yaparken genel veri kümesinin tamamı üzerine düşündük, ancak bazı durumlarda aynı sonuçları veri kümesinin içerisinde bulunan farklı gruplarda da bulmak isteyebiliriz. Senaryomuzu şu şekilde düşünelim, **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 sorunumuz var 3 farklı değer yerine 30 farklı değer olsaydı? İşte bu şekilde senaryolar için yani verileri gruplama için **GROUP BY** anahtar kelimesi kullanılır.

GROUP BY Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı>, ... (veya aggregate func)

FROM <tablo\_adı>

GROUP BY <sütun\_adı>, <sütun\_adı>, ...

Burada şuna dikkat etmemiz gerekir, SELECT anahtar kelimesinde bulunan sütunların GROUP BY anahtar kelimesi içerisinde bulunması gerekir.

GROUP BY Örnek Kullanım

Yukarıdaki senaryomuzu GROUP BY anahtar kelimesini kullanarak gerçekleştirelim. Dikkat ettiğiniz üzere SELECT ile kullanılan rental\_rate sütunu GROUP BY satırında da kullanılmıştır.

SELECT rental\_rate, MAX(length)

FROM film

GROUP BY rental\_rate;

HAVING

**HAVING** anahtar kelimesi sayesinde gruplandırılmış verilere koşullar ekleyebiliriz. Hemen aklımıza **WHERE** anahtar kelimesi geldi değil mi? Ancak WHERE anahtar kelimesi ile biz satır bazlı koşullar verebiliyoruz.

Şöyle bir senaryomuz olsun. Her bir rental\_rate oranına karşılık gelen film sayısını bulalım. Bunu GROUP BY ile gerçekleştirebiliriz. Ancak bu kez 1 adım öteye gidip şöyle bir koşul ekleyelim toplam film sayısı 325 ten fazla olan rental\_rate oranlarını görelim. Bu durumda GROUP BY ile elde ettiğimiz toplam film sayılarına koşul eklememiz gerekir.

SELECT rental\_rate, COUNT(\*)

FROM film

GROUP BY rental\_rate

HAVING COUNT(\*) > 325;

ALIAS (AS)

**AS** anahtar kelimesi sayesinde sorgular sonucu oluşturduğumuz sanal tablo ve sütunlara geçici isimler verebiliriz.

**ALIAS SÜTUN KULLANIMI**

SELECT <sütun\_adı> AS <geçici\_ad>

FROM <tablo\_adı>;

**ALIAS TABLO KULLANIMI**

SELECT <sütun\_adı>, <sütun\_adı>...

FROM <tablo\_adı> AS <geçici\_ad>;

-------------------------------------------------------------------------------------------------------------------------------------------

Tablo Oluşturmak ve Silmek (CREATE - DROP)

Tablo Oluşturmak - CREATE

SQL ile yeni bir tablo oluşturmak için **CREATE** anahtar kelimesi kullanılır. Tablo oluştururken sonrasında daha detaylı konuşacağımız 3 önemli başlık daha vardır.

Sütunlara verilecek isim, sütunların veri tipi ve varsa sütunlarda bulunan kısıtlama yapıları.

Tablo Oluşturmak - CREATE Söz Dizimi

CREATE TABLE <tablo\_adı> (

<sütun\_adı> <veri\_tip> (kısıtlama\_adı>,

<sütun\_adı> <veri\_tip> (kısıtlama\_adı>,

....

);

Tablo Oluşturmak - CREATE Örnek Kullanım

**author** isminde bir tablo oluşturalım, id, first\_name, last\_name, email, birthday sütunları olsun. Veri tipleri ve kısıtlama yapılarıyla ilgili sonrasında detaylı olarak konuşacağız.

CREATE TABLE author (

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100)

birthday DATE

);

Tablo Silmek - DROP

Oluşturduğumuz tabloları silmek için **DROP** anahtar kelimesi kullanılır.

Tablo Silmek - DROP Söz Dizimi

DROP TABLE (IF EXISTS) <tablo\_adı>;

Burada **IF EXISTS** yapısını kullanarak yanlış tablo ismi yazımı durumunda hata mesajı almayı önleriz.

Tablo Silmek - DROP Örnek Kullanım

"test" isimli tablomuzu silmek istersek;

DROP TABLE IF EXISTS test;

**IF EXISTS =>Çook önemlidir !!!**

Tablo Verilerini Güncellemek (UPDATE - DELETE)

Bir tabloda bulunan verileri güncellemek veya silmek için öncelikle örnek bir tablo oluşturup içine 5 date veri yerleştireceğim.

Bunun için [Mockaroo](https://www.mockaroo.com/" \t "_blank) benzeri servisleri kullanabiliriz. Aşağıdaki örnek tablo oluşturma ve veri girme komutlarını bulabilirsiniz.

CREATE TABLE my\_apps (

id INT,

name VARCHAR(50),

price VARCHAR(50)

);

INSERT INTO my\_apps (id, name, price) values (1, 'Ronstring', '$0.96');

INSERT INTO my\_apps (id, name, price) values (2, 'Duobam', '$3.44');

INSERT INTO my\_apps (id, name, price) values (3, 'Tresom', '$2.21');

INSERT INTO my\_apps (id, name, price) values (4, 'Redhold', '$2.52');

INSERT INTO my\_apps (id, name, price) values (5, 'Y-find', '$9.14');

UPDATE

**UPDATE** anahtar kelimesi sayesinde tablomuzda bulunan verileri güncelleyebiliriz.

UPDATE Söz Dizimi

UPDATE <tablo\_adı>

SET <sütun\_adı> = değer,

<sütun\_adı> = değer,

----

WHERE <koşul\_adı>;

UPDATE Örnek Kullanım

**my\_apps** tablosunda bulunan ve id 2' ye eşit olan verimizin name sütunundaki degerini 'Mayak' price sütunundaki değerini '$5.22' ile değiştirelim.

UPDATE my\_apps

SET name = 'Mayak',

price = '$5.22'

WHERE id = 2;

DELETE

**DELETE** anahtar kelimesi sayesinde tablomuzda bulunan verileri silebiliriz.

DELETE Söz Dizimi

DELETE FROM <tablo\_adı>

WHERE <koşul\_adı>;

DELETE Örnek Kullanım

**my\_apps** tablosunda bulunan name sütunundaki verisi 'Tresom' olan satırı silelim.

DELETE FROM my\_apps

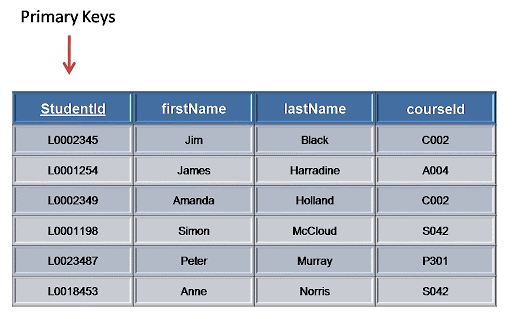
WHERE name = 'Tresom';

Birincil Anahtar ve Yabancı Anahtar (PRIMARY KEY ve FOREIGN KEY)

PRIMARY KEY

**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.

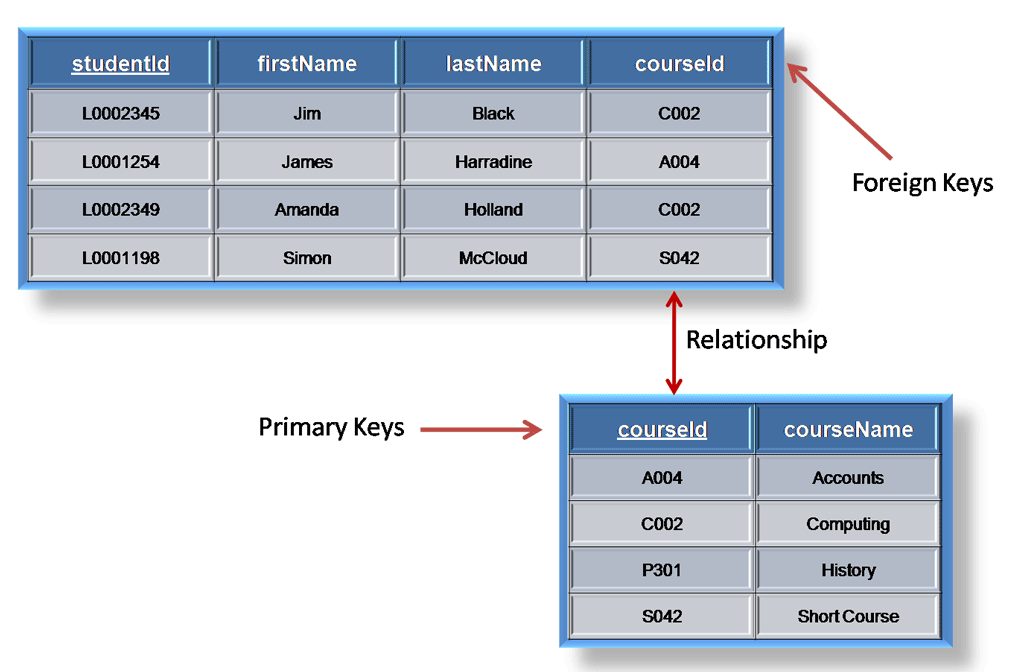
[](https://github.com/Kodluyoruz/taskforce/blob/main/sql101/PrimaryKeyAndForeignKey/figures/PrimaryKey.gif)

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.

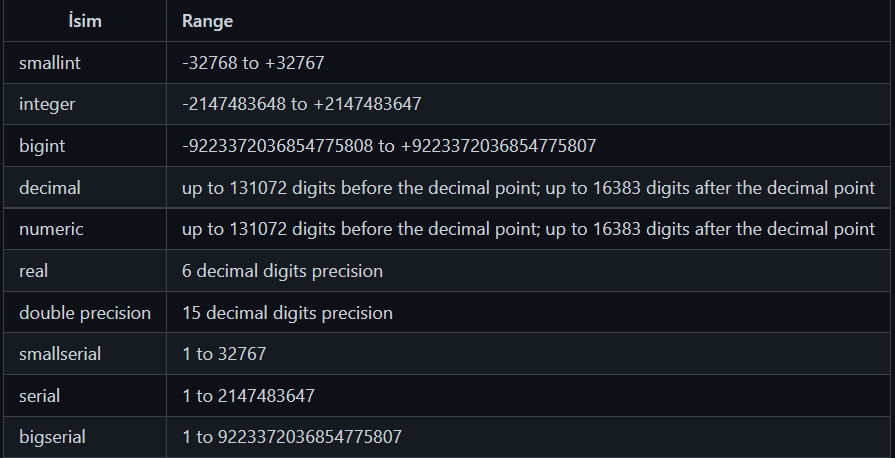
[](https://github.com/Kodluyoruz/taskforce/blob/main/sql101/PrimaryKeyAndForeignKey/figures/ForeignKey.gif)

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.

Temel Veri Tipleri

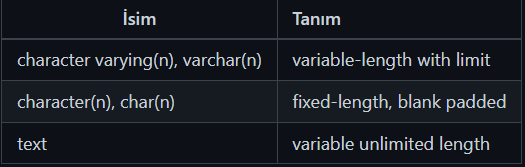
* Sayısal Veri Tipleri
* Karakter Veri Tipleri
* Boolean Veri Tipleri
* Date / Time Veri Tipleri

Sayısal Veri Tipleri



Veri Tipleri II

Karakter Veri Tipleri



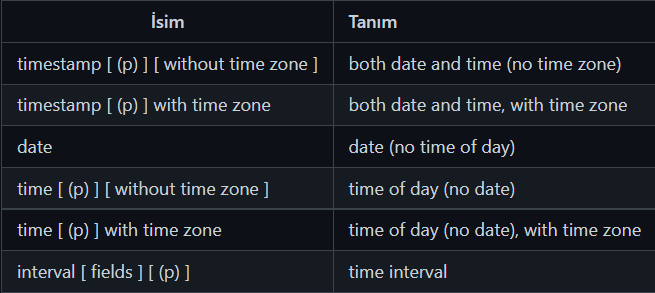
Sınırlı sayıda karekter kullanımı için VARCHAR veya CHAR veri tipleri kullanılır. VARCHAR veri tipi doldurulmayan karakterleri yok sayar, CHAR veri tipi ise doldurulmayan karakterler için boşluk bırakır. Sınırsız karekter kullanımı için ise TEXT veri tipi kullanılır.

Boolean Veri Tipleri

TRUE, FALSE veya NULL (Bilinmeyen) değerlerini alabilirler.

* TRUE: true, yes, on, 1
* FALSE: false, no, off, 0

Zaman / Tarih Veri Tipleri



ALTER ve NOT NULL

NOT NULL

Birçok durumda bizler herhangi bir sütuna yazılacak olan verilere belirli kısıtlamalar getirmek isteriz. Örneğin yaş sütünunda sadece sayısal verilerin olmasını isteriz ya da kullanıcı adı sütununda bilinmeyen (NULL) değerlerin olasını 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\_adı>

ALTER COLUMN <sütun\_adı>

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)

-------------------------------------------------------------------------------------------------------------------------------------------

JOIN Kavramı (Birleştirme)

Veritabanları çoğunlukla birbiri ile ilişkili olan tablolardan oluşur. Bu birbiri ile ilişkili olan tablolardaki verileri farklı JOIN yapıları kullanarak sanal olarak birleştirip daha anlamlı veriler haline getirebiliriz.

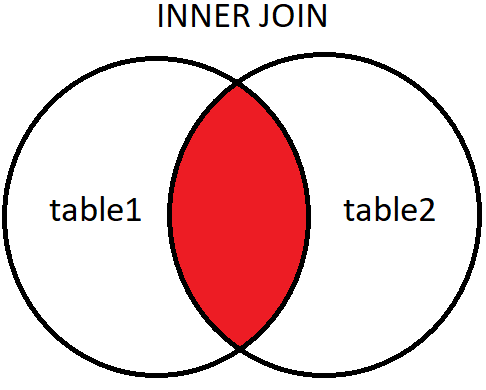
INNER JOIN

INNER JOIN yapısı sayesinde birbiriyle ilişkili olan tabloların birbiriyle eşleşen (kesişen) verilerini sıralayabiliriz. Senaryomuzda kitapları gösterdiğimiz **book**tablosu ve yazarları gösterdiğimiz **author** tablosu var, author tablosunun id sütunuyla book tablosunun author\_id sütunlarında bulunan veriler sayesinde her iki tabloya ait bilgilerden daha anlamlı sonuçları elde edebiliriz.

SELECT book.title, author.first\_name, author.last\_name

FROM book

JOIN author ON author.id = book.author\_id;



Yukarıdaki görselimizde de gördüğümüz üzere INNER JOIN tablolar arasındaki eşleşen (kesişen) verileri sıralar. Bundan dolayı INNER JOIN yapısı simetriktir, author - book tablolarının yerlerinin değiştirilmesi sonucu etkilemez.

INNER JOIN Söz Dizimi

SELECT <sütun\_adı>, <sütun\_adı> ...

FROM <tablo1\_adı>

INNER JOIN <tablo2\_adı>

ON <tablo1\_adı>.<sütun\_adı> = <tablo2\_adı>.<sütun\_adı>;

Buradaki tablo1 "left table", tablo2 "right table" olarak da adlandırılır.