**<https://www.w3schools.com/sql/default.asp>**

[**https://www.postgresql.org/docs/15/**](https://www.postgresql.org/docs/15/)

**SELECT**

**-- Yorum SATIDIR**

**SELECT** \* **FROM** <tablo\_adı>; // Tüm tablodaki verileri getirir

**SELECT** <sütun\_adı>, <sütun\_adı> **FROM** <tablo\_adı>; // Belirtilen sütun bilgileri getirilir

**SELECT** first\_name, last\_name **FROM** actor;

**WHERE ve Karşılaştırma Operatörleri**

**SELECT** <sütun\_adı>, <sütun\_adı>

**FROM** <tablo\_adı>

**WHERE** <koşul>;

**SELECT** title, replacement\_cost

**FROM** film

**WHERE** replacement\_cost = 14.99;

| **Operator** | **Description** |
| --- | --- |
| **<** | **Küçüktür** |
| **>** | **Büyüktür** |
| **<=** | **Küçük Eşit** |
| **>=** | **Büyük Eşit** |
| **=** | **Eşit** |
| **<>** | **Eşit Değil** |
| **!=** | **Eşit Değil** |

**WHERE ve Mantıksal Operatörler**

**SELECT** \* **FROM** actor

**WHERE** first\_name = 'Penelope' **AND** last\_name = 'Monroe';

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

**WHERE** **NOT** (rental\_rate = 4.99 **OR** rental\_rate = 2.99);

| ***a*** | ***b*** | ***a* AND *b*** | ***a* OR *b*** |
| --- | --- | --- | --- |
| TRUE | TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE | TRUE |
| TRUE | NULL | NULL | TRUE |
| FALSE | FALSE | FALSE | FALSE |
| FALSE | NULL | FALSE | NULL |
| NULL | NULL | NULL | NULL |

| ***a*** | **NOT *a*** |
| --- | --- |
| TRUE | FALSE |
| FALSE | TRUE |
| NULL | NULL |

**BETWEEN**

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

**WHERE** length **BETWEEN** 100 **AND** 140;

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

**IN**

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

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

-- WHERE length = 30 OR length = 60 OR length = 90 OR length = 120; // ifadesi ile aynı sonucu verir.

**LIKE ve ILIKE**

**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.

**SELECT** \* **FROM** actor

**WHERE** first\_name **LIKE** 'P\_RDE';

Like ve ilike arasındaki fark, like büyük küçük harfe duyarlıdır, ilike ise duyarlı değildir

| **Operator** | **Equivalent** |
| --- | --- |
| **~~** | **LIKE** |
| **~~\*** | **ILIKE** |
| **!~~** | **NOT LIKE** |
| **!~~\*** | **NOT ILIKE** |

**DISTINCT**

**SELECT** **DISTINCT** rental\_rate **FROM** film;

rental\_rate sütununda bulunan birbirinden farklı 2.99, 0.99, 4.99 gibi verilerini gösterir.

**COUNT**

**SELECT** **COUNT**(\*) **FROM** actor

**WHERE** first\_name = 'Penelope';

veri sayısını bulmak istersek COUNT fonksiyonunu kullanırız.

**ORDER BY**

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

**ORDER** **BY** title **ASC**;

ASC sıralama varsayılan olduğu için ayrı bir şekilde yazılması zorunluluğu yoktur

DESC ise ters sıralama yapar

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

**WHERE** title **LIKE** 'A%'

**ORDER** **BY** title **ASC** length **DESC**;

**--ORDER BY RIGHT**(sutün, 3)**—** ismin son 3 harfini alıp sıralar, EMRE de MRE diye sıraya sokar

**LIMIT**

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

**WHERE** title **LIKE** 'B%'

**ORDER** **BY** length **DESC**

**LIMIT** 10;

Verilerin tamamını değil belirli sayıda olanlarını sıralamak isteriz, bunun için LIMIT anahtar kelimesini kullanırız.

**OFFSET**

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

**WHERE** title **LIKE** 'B%'

**ORDER** **BY** length **DESC**

**OFFSET** 6

**LIMIT** 4;

Bazı durumlarda sonuç olarak gördüğümüz veri grubu içerisinden bazılarını "pass" geçmek isteriz. Yukarıdaki örnekte ilk 6 veri pas geçer ve sonraki 4 listelenir yani 7,8,9,10 listelenir

**Aggregate Fonksiyonlar**

**MIN**

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

Sütundaki minimum değeri alır

**MAX**

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

Sütundaki maximum değeri alır

**SUM**

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

Sütundaki toplam değeri verir

**AVG**

**SELECT** **AVG**(length) **FROM** film;

Sütundaki ortalama değeri verir

**GROUP BY**

**SELECT** rental\_rate, **MAX**(length) **FROM** film

**GROUP** **BY** rental\_rate;

SELECT anahtar kelimesinde bulunan sütunların GROUP BY anahtar kelimesi içerisinde bulunması gerekir.

**HAVING**

**SELECT** rental\_rate, **COUNT**(\*) **FROM** film

**GROUP** **BY** rental\_rate

**HAVING** **COUNT**(\*) > 325;

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.

**ALIAS (AS)**

**SELECT** rental\_rate **AS** yeni\_ismi **FROM** film

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

**Tablo Oluşturmak – CREATE**

**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**

**DROP** **TABLE** **IF** **EXISTS** test;

**Tablo Verilerini Güncellemek**

**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** my\_apps

**SET** name = 'Mayak',

price = '$5.22'

**WHERE** id = 2;

**DELETE**

**DELETE** **FROM** my\_apps

**WHERE** name = 'Tresom';

**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.

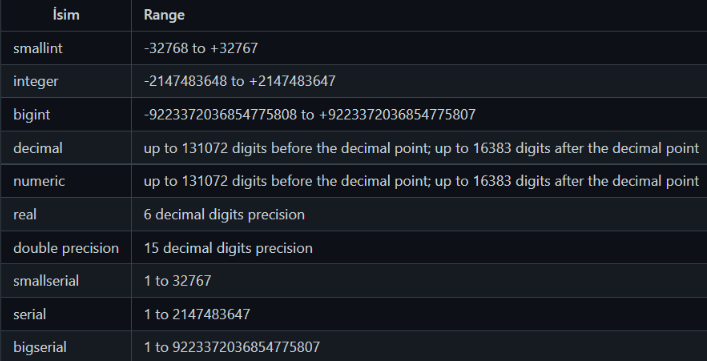
**FOREIGN KEY**

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

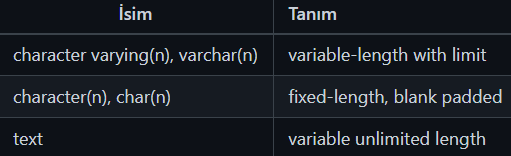
* Bir tabloda birden fazla sütun FK olarak tanımlanabilir.
* Aynı sütunun içerisinde aynı değerler bulunabilir.

**VERİ TİPLERİ**

**Sayısal Veri Tipleri**



**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.

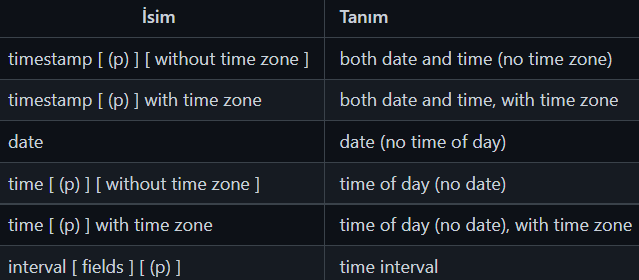
**Boolean Veri Tipleri**

TRUE, FALSE veya NULL (Bilinmeyen) değerlerini alabilirler.

TRUE: true, yes, on, 1

FALSE: false, no, off, 0

**Zaman / Tarih Veri Tipleri**



**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.

**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 kıstlamalar da vermek isteyebiliriz.

**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 tablardaki 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 sorgumuzda tablolar arasındaki eşleşmeyi author.id ve book.author\_id sütunları yardımıyla yapıyoruz.

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>;

**LEFT JOIN**

LEFT JOIN yapısındaki tablo birleştirmesinde, birleştirme işlemi tablo 1 (soldaki tablo) üzerinden gerçekleştirilir.

Aşağıdaki SQL sorgusunda kitap isimlerinin tamamını alıyoruz, sonrasında bu kitap isimleriyle eşleşebilen yazar isimlerini alıyoruz. Kitap isimlerine karşılık olmayan yazarlar için NULL değeri alıyoruz.

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

**FROM** book

**LEFT** **JOIN** author

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

**LEFT JOIN Söz Dizimi**

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

**FROM** <tablo1\_adı>

**LEFT** **JOIN** <tablo2\_adı>

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

**RIGHT JOIN**

RIGHT JOIN yapısındaki tablo birleştirmesinde, birleştirme işlemi tablo 2 (sağdaki tablo) üzerinden gerçekleştirilir

Aşağıdaki SQL sorgusunda yazar isim ve soyisim bilgilerinin tamamını alıyoruz, sonrasında eşleşebilen kitap isimlerini alıyoruz. Yazar bilgilerine karşılık olmayan kitap isimleri için NULL değeri alıyoruz.

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

**FROM** book

**RIGHT** **JOIN** author

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

**RIGHT JOIN Söz Dizimi**

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

**FROM** <tablo1\_adı>

**RIGHT** **JOIN** <tablo2\_adı>

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

**FULL JOIN**

Full JOIN yapısındaki tablo birleştirmesinde, birleştirme işlemi her iki tablo üzerinden gerçekleştirilir.

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

**FROM** book

**FULL** **JOIN** author

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

,Yukarıdaki sorgumuz sonucunda göreceğimiz gibi kitapların yazar bilgisine sahip değilse NULL değerlerini alırız, yazarlar kitap bilgisine sahip değilse orada da NULL değerlerini alırız.

**FULL JOIN Söz Dizimi**

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

**FROM** <tablo1\_adı>

**FULL** **JOIN** <tablo2\_adı>

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

**UNION**

UNION operatörü sayesinde farklı SELECT sorgularıyla oluşan sonuçları tek bir sonuç kümesi haline getiririz.

**UNION Kullanımı**

bookstore veritabanında iki adet sorgu yapıyoruz. İlk sorgumuzda sayfa sayısı en fazla olan 5 kitabı, ikinci sorgumuzda ise isme göre 5 kitabı sıralıyoruz. UNION anahtar kelimesi sayesinde bu iki sorguyu da birleştirebiliriz.

(

**SELECT** \*

**FROM** book

**ORDER** **BY** title

**LIMIT** 5

)

**UNION**

(

**SELECT** \*

**FROM** book

**ORDER** **BY** page\_number **DESC**

**LIMIT** 5

);

UNION operatörü kullanılacağı sorguların, sütun sayıları eşit olmalıdır ve sütunlardaki veri tipleri eşleşmelidir.

**UNION Söz Dizimi**

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

**FROM** <table1>

**UNION**

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

**FROM** <table2>

**UNION ALL**

UNION operatörü bize birleşik veriler içerisindeki tekrar edenleri göstermez. Tekrar edenleri görmek için UNION ALL kullanırız.

**INTERSECT ve EXCEPT**

INTERSECT operatörü sayesinde farklı SELECT sorgularıyla oluşan sonuçların kesişen verilerini tek bir sonuç kümesi haline getiririz.

**INTERSECT Kullanımı**

bookstore veritabanında iki adet sorgu yapıyoruz. İlk sorgumuzda sayfa sayısı en fazla olan 5 kitabı, ikinci sorgumuzda ise isme göre 5 kitabı sıralıyoruz. INTERSECT anahtar kelimesi sayesinde bu iki sorgu sonucunda oluşan veri kümelerinden kesişen verileri tek bir sonuçta birleştiririz.

(

**SELECT** \*

**FROM** book

**ORDER** **BY** title

**LIMIT** 5

)

**INTERSECT**

(

**SELECT** \*

**FROM** book

**ORDER** **BY** page\_number **DESC**

**LIMIT** 5

);

INTERSECT operatörü kullanılacağı sorguların, sütun sayıları eşit olmalıdır ve sütunlardaki veri tipleri eşleşmelidir.

**INTERSECT Söz Dizimi**

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

**FROM** <table1>

**INTERSECT**

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

**FROM** <table2>

**INTERSECT ALL**

INTERSECT operatörü bize kesişen veriler içerisindeki tekrar edenleri göstermez. Tekrar edenleri görmek için INTERSECT ALL kullanırız.

**EXCEPT Kullanımı**

bookstore veritabanında iki adet sorgu yapıyoruz. İlk sorgumuzda sayfa sayısı en fazla olan 5 kitabı, ikinci sorgumuzda ise isme göre 5 kitabı sıralıyoruz. EXCEPT anahtar kelimesi sayesinde ilk sorguda olup ancak ikinci sorguda olmayan verileri gösterir.

(

**SELECT** \*

**FROM** book

**ORDER** **BY** title

**LIMIT** 5

)

**EXCEPT**

(

**SELECT** \*

**FROM** book

**ORDER** **BY** page\_number **DESC**

**LIMIT** 5

);

EXCEPT operatörü kullanılacağı sorguların, sütun sayıları eşit olmalıdır ve sütunlardaki veri tipleri eşleşmelidir.

**EXCEPT Söz Dizimi**

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

**FROM** <table1>

**EXCEPT**

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

**FROM** <table2>

**EXCEPT ALL**

EXCEPT operatörü bize ilk sorguda olan ancak ikinci sorguda olmayan veriler içerisindeki tekrar edenleri göstermez. Tekrar edenleri görmek için EXCEPT ALL kullanırız.

**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 ve All Operatörleri**

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

**ANY Operatörü**

Alt sorgudan gelen herhangi bir değer koşulu sağlaması durumunda TRUE olarak ilgili değerin koşu sağlamasını sağlar.

**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.

**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 eşitlik olduğu için aynı anda 4 ve 5 in bu koşulu sağlaması olanaksız olduğu için herhangi bir değer dönülmeyecektir.

**Alt Sorgular ve JOIN Kullanımı**

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.