

SQL

Structured Query Language (SQL)

- SQL, Yapılandırılmış Sorgu Dili anlamına gelir ve ilişkisel veritabanları ile (reational databases) iletişim kurmak için kullanılır.
- SQL bildirimsel bir dildir, prosedürel değil. Bildirimsel programlama, programların gerçekleştirilmesi gereken komutları veya adımları açıkça listelemekten istenen sonuçları tanımladığı, zorunlu olmayan bir programlama stilidir. Tek bir SQL bildirimi yazıp DBMS (database management system)'e teslim ediyoruz, DBMS daha sonra bizden gizlenen bir dahili kodu yürütüyor.
- Bildirimsel paradigma, nasıl yapacağınızı söylemek zorunda kalmadan istediğinizi söylediğiniz yerdir. Prosedürel paradigma ile (JAVA, C'de kullanılır), sonucu almak için kesin adımları belirtmeniz gerekir. SQL, yordamdan çok bildirimseldir, çünkü sorgular sonucu üretmek için adımlar belirtmez.
- SQL, en saf haliyle bir programlama dili değil, bir sorgu dilidir. Çünkü döngüleri ve kontrol yapılarını gerçekleştirebilmesi gerekiyor.
- Prolog gibi bazı mantıksal programlama dilleri ve SQL gibi veritabanı sorgulama dilleri, prensipte bildirimsel olmakla birlikte, prosedürel bir programlama stilini de destekler.
- SQL ile veritabanında depolanan verilere erişebilir ve bu verileri değiştirebilirsiniz. Farklı erişim türleri vardır; Veritabanından **veri alınması (retrieval)**, **Yeni verilerin eklenmesi (insert)**, **verilerin güncellenmesi (update)**, **verilerin silinmesi (delete)**, ayrıca SQL ile veritabanında tablolar oluşturulabilir. (Bunlara dikkat çektim çünkü SQL bu dört ana durum üzerinden yayılacak, şimdi kafa karışmasın sadece dursun)

<https://www.youtube.com/watch?v=XklfyJpxFdU>

QUERY

- Veritabanından bilgi almak için sorgu adı verilen belirli bir kelime vardır. SQL bir dil olduğu için grameri vardır. Sorgu veri tabanından bilgi anımasını isteyen bir ifadedir.

TABLO

-TABLO ANATOMİSİ-

- **Tablo** sütun ve satırlardan oluşur. Tabloya ilişki (relation) de denir. Tablo, sütunlar ve satırlar biçiminde depolanan düzenli bir veri topluluğudur.
- **Sütun**, tablo tarafından depolanan bir veri parçasıdır. Bir sütuna alan(field) veya nitelik (attribute) de denir.
- **Satır**, tek bir şeyin niteliklerini tanımlayan tek bir sütun kümesidir. Satıra ayrıca kayıt (record) veya demet (tuple) adı da verilir.
- Kabaca ve daha net bir şekilde; herhangi bir sayfa üzerinde yanlamasına sıralanmış sözcüklerin oluşturduğu her bir diziye satır, herhangi bir sayfanın yukarıdan aşağıya doğru ayrılmış olduğu bölümlerden her birine sütun adı verilir. Kısacası yatay olan kısımlara satır. Dikey olan kısımlara ise sütun adı verilir.
- Bir veritabanı bir veya daha fazla tablodan oluşabilir. Çoğu durumda, birden fazla tablo. Her tablonun çalışanlar, departmanlar veya müşteriler gibi benzersiz bir adı vardır.

SQL DİL ÖĞELERİ

SQL SYNTAX

SQL Büyük / Küçük Harf Duyarlı DEĞİLDİR. Ancak sözdiziminde PEP8 gibi adabı muaşeret denebilecek bazı hususlar vardır.

- SQL deyimleri SELECT, INSERT, UPDATE, DELETE vb. bir anahtar sözcükle başlar ve tüm deyimler noktalı virgül (;) ile biter.
- Sondaki noktalı virgül, ifadenin tamamlandığını ve yürütülmeye hazır olduğunu gösterir.

- Komutlarını büyük harfle yazmak en yaygın ve tercih edilen stildir. Yani anahtar kelimeleri.
- SQL ayrıca büyük/küçük harfe duyarlı değildir; bu, sorgunuzda hem SELECT hem de select kullanabileceğiniz anlamına gelir. SQL için aynı şeyi ifade ederler.
- SQL'de beyaz boşluklar (white spaces) ve boş satırlar yoksayılr. Fakat sorgunuzu temiz ve daha okunaklı tutmak için gereksiz boş satırlar ve boşluklar kullanmanız önerilmez.

SELECT

SELECT, bir veritabanından veri seçmek için kullanılır. Bu işleme sorgulama denir. SELECT komutunun döndürülen veriler sonuç kümesi adı verilen bir tabloda saklanır.

- SQL'deki kullanılan ifadelerin çoğu 'SELECT' komutu ile başlar. SELECT veritabanından verileri alır.

tablo

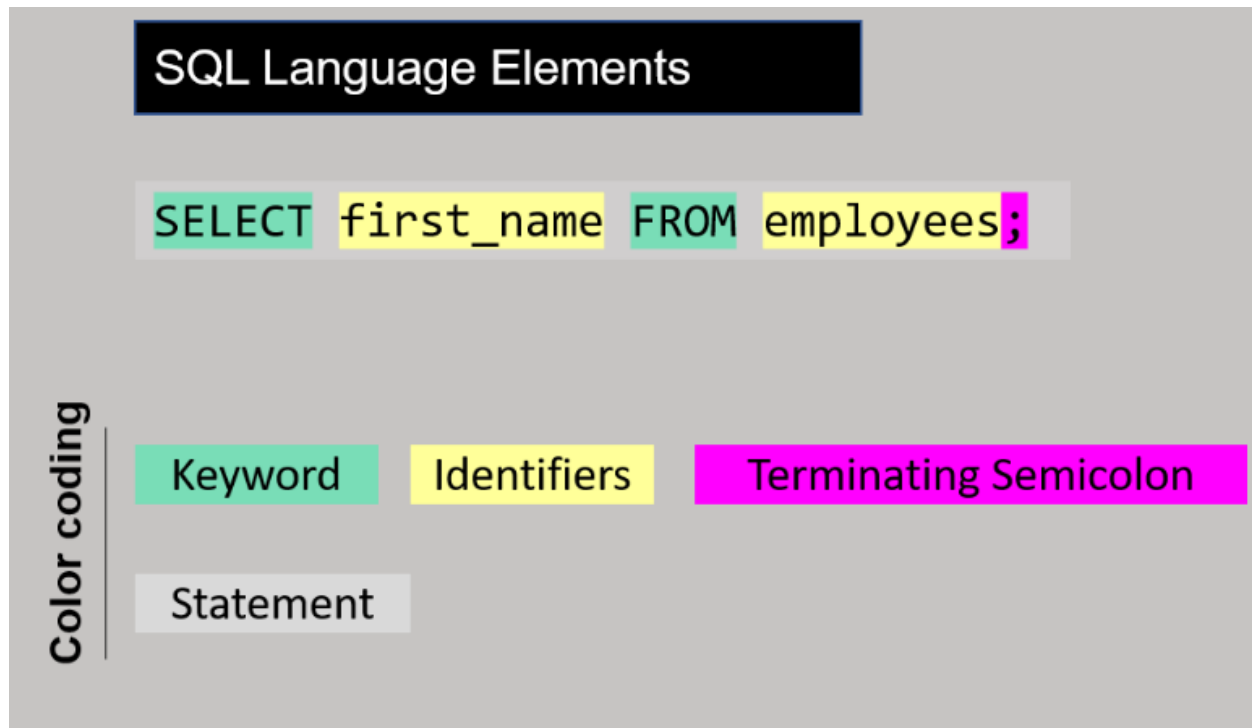
Aa ID	::: ad	≡ Soyad	≡ Adres	≡ Şehir
<u>KullaniciID1</u>	1 AD	Soyad1	Adres1	Şehir1
<u>KullaniciID2</u>	2 AD	Soyad2	Adres2	Şehir2
<u>KullaniciID3</u>	3 AD	Soyad3	Adres3	Şehir3

(Kod yazacak bir şeyler ararken tablo yazmış bulundum, sağlık olsun)

```
SELECT ad, Soyad FROM tablo;
SELECT column_name(s) FROM table_name;
```

- burada 'ad, Soyad' içinden veri seçmek istediğimiz tablonun alan adlarıdır.
- Tabloda bulunan tüm alanları seçmek istiyorsak '*' yani 'asteriks' komutu kullanılır.
- Kolon isimlerini SELECT ifadesinden sonra aralarında virgül ile belirtmemiz gerekmektedir.

- **FROM** ile birlikte kullanılır. FROM nereden olacağı, * tüm veriyi, SELECT ise elde edilen veri üzerinde ne yapılacağını belirtir.



: Burada gördüğümüz **Statement**, **Keyword** ve az sonra değineceğim **Clause**, SQL terminolojisinde bilmemiz gereken tabirlerdir. Bu dilin öğrenilmesi bu üç terim üzerinden ilerleyecek:

- **Keyword** yukarıda da görüldüğü üzere SELECT, FROM gibi önceden tanımlanmış ve anlamını değiştiremeyeğimiz öğelerdir.
- **Clause** SQL'de kullanabileceğimiz yerleşik işlevlerdir. Cümlelerin yardımıyla, tabloda kolayca saklanan verilerle başa çıkabiliriz.

Yan tümceler, verileri hızlı bir şekilde filtrelememize ve analiz etmemize yardımcı olur. Veritabanında depolanan büyük miktarda veriye sahip olduğumuzda, kullanıcı tarafından istenen verileri sorgulamak ve almak için Yan Tümceleri kullanırız.

where, and, or, like, top gibi

- **Statement** Tam bir sorgu "statement" olarak adlandırılır. Yani statement bir tam sorgu ifadesidir.

Sorgunun sonucu, sonuç kümesi adı verilen bir sonuç tablosunda saklanır.

- Birden fazla sütunun verilerini görmek ve çağırmak istediğimizde ',' ile ayırarak çoklu (multiple) veri döndürebiliriz:

```
SELECT column1, column2 FROM table1;
```

DISTINCT Clause

SELECT DISTINCT komutu / deyimi yalnızca farklı değerleri döndürmek için kullanılır. Genelde bir tablonun içinde bir çok aynı değer yinelenir. DISTINCT cümlesi kullanıldığında aynı satırlar tekrar listelenmez. Yalnızca eşsiz olanları getirir.

WHERE & LIMIT Clauses

- WHERE cümlesi tablodaki kayıtları filtrelemek için kullanılır. Filtreleme yapmak için spesifik kriterleri belirtmemiz gerekir.
- Where cümlesini kullanarak, yapmış olduğumuz sorguya koşul ekleyebiliriz.
- WHERE yan tümcesi, yalnızca belirtilen bir koşulu yerine getiren kayıtları ayıklamak için kullanılır.
- LIMIT komutunu kullanılarak veritabanından dönecek sonuçlarda sayı sınırı getirebilmekteyiz

```
SELECT column_name(s) FROM table_name WHERE condition(s);
```

---ya da

```
SELECT * FROM student_table WHERE grade > 70
```

----LIMIT için

```
SELECT column_name(s) FROM table_name LIMIT number_rows;
```

Operators in the WHERE Clause

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. This operator may be written as != in some versions of SQL
BETWEEN	Test if a value is between a certain range of values
LIKE	Determine if a character string matches a predefined pattern
IN	Test whether or a value matches any value in a list

- LIMIT yan cümlesi WHERE yan cümlesinden sonra gelir.

ORDER BY Clause

- SELECT ifadesi, kayıtları belirtilmemiş bir sırada döndürür. Alfabetik veya sayısal sırayla veri almak istemeniz durumunda ORDER BY anahtar sözcüğünü kullanırız.
- SQL sorgularında, SELECT ile çektiğimiz sonuçları sıralamak için ORDER BY kelimesini kullanırız. Bu sıralamayı yapabilmek için bir kolon ismi vermek gerekir.
- ORDER BY anahtar sözcüğü, sonuç kümesini azalan veya artan düzende sıralar.
- Varsayılan olarak ORDER BY anahtar sözcüğü, kayıtları artan düzende sıralar (otomatik tanımlı olarak). **ASC** anahtar sözcüğü ile de bu komutu verebiliriz.

- Kayıtları azalan düzende sıralamak için **DESC** keyword'ünü kullanmamız gerekmektedir.

```
SELECT * FROM employees ORDER BY first_name;
```

ORDER BY komutu ile **çoklu sıralama** da yapabiliriz:

```
SELECT column_name(s)
FROM table_name
ORDER BY column1
ASC|DESC, column2 ASC|DESC, columnN ASC|DESC;
```

not: daha önce syntax'tan bahsederken farklı satırlara da yazabileceğimizi söylemiştik. sql büyük küçük harfe ya da white space durumuna takılmıyor.

WHERE sorgusuyla ORDER BY'nin birlikte kullanımı;

- ORDER BY deyimini WHERE deyiminden sonra gelir.

```
SELECT column_name(s)
FROM table_name
WHERE condition
ORDER BY column_name(s) ASC|DESC;
```

Aşağıya 'employees' adında bir tablo bırakıyorum. Bundan sonra bir kaç örneği bunun üzerinde yapacağız.

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

Maaşı 80000 dolardan fazla olan çalışanları azalan düzende sıralamak istediğimizde;

```
SELECT *
FROM employees
WHERE salary > 80000
ORDER BY first_name DESC;
```

sorgumuz yukarıdaki gibi olurken çıktı da şu şekilde olacaktır;

emp_id	first_name	last_name	salary	job_title	gender	hire_date
70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
17679	Robert	Gilmore	110000	Operations Dire	Male	2018-09-04
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02

Önce maaşı 80.000 doların üzerinde olan çalışanları iade ettik. Ardından, bunu azalan düzende ilk adlara göre sıraladık.


```

SELECT
    sutun1,
    sutun2,
    sutunN
FROM
    tablo_adi
ORDER BY
    sutunA ASC,
    sutunB DESC

```

WHERE sorgusuyla AND, OR ve NOT Kullanımı

- Where koşulunu kullanırken birden fazla koşul belirtmek mümkündür. Bunu yapabilmek için AND ya da Or kullanılır.
- AND kullanıldığında belirtilen koşulların tamamının sağlanması beklenirken OR kullanıldığında koşullardan herhangi birinin sağlanması yeterlidir.

```

SELECT sütun1,sütun2, ...
FROM tablo_adi
WHERE koşul1 AND koşul2 AND koşul3 .... ;

--- ya da

SELECT sütun1,sütun2, ...
FROM tablo_adi
WHERE koşul1 OR koşul2 OR koşul3 .... ;

```

- NOT parametresi AND ve OR gibi WHERE içinde şart olarak kullanılır. Belirtilen şartları karşılamayanları gösterir.

‘employees’ tablosunu göz önünde aldığımızda;

```

WHERE left_conditon AND right_condition

```

formülüne göre;

```
SELECT *
FROM employees
WHERE job_title = 'Data Scientist' AND gender = 'Male';
```

hem job_title = 'data scientist' hem de gender = 'Male' şartlarına göre çıktı verecektir;

emp_id	first_name	last_name	salary	job_title	gender	hire_date
30840	David	Barrow	85000	Data Scientist	Male	2019

```
WHERE first_condition OR second_condition
```

formülüne göre;

```
SELECT *
FROM employees
WHERE job_title = 'Data Scientist' OR gender = 'Male';
```

sorgusunu yaptığımızda çıktımız;

emp_id	first_name	last_name	salary	job_title	gender	hire_date
17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
76589	Jason	Christian	99000	Project Manager	Male	2019-01-21

şeklinde olacaktır.

Yani AND kullanımında koşulların tamamını sağlanması gerekirken OR kullanımında koşullardan herhangi birinin sağlanması yeterli olacaktır.

- **AND** işlecinin istenilen kaydı getirebilmesi için **her iki ifadenin de doğru** olması gerekmektedir.
- **OR** işlecinin istenilen kaydı getirebilmesi için **iki ifadeden en az birinin doğru** olması gerekmektedir.

bir başka deyişle;

- TÜM koşullarınızın doğru olmasını istiyorsanız, AND kullanın.
- Koşullarınızdan HERHANGİ BİRİNİN doğru olmasını istiyorsanız, OR ögesini kullanın.
- AND ve OR işleçlerini bir arada kullanmayı düşünüyorsanız parantez kullanmayı sakın unutmayın!
- **NOT** operatörü, WHERE yan tümcesindeki bir koşulu reddetmek için kullanılır. NOT, WHERE anahtar sözcüğünden hemen sonra yer alır. AND & OR operatörleri ile kullanabilirsiniz. Sözdizimi;

```
WHERE NOT first_condition
```

```
SELECT *  
FROM employees  
WHERE gender = 'Male';  
---  
SELECT *  
FROM employees  
WHERE NOT gender = 'Female';
```

BETWEEN Operatörü

BETWEEN operatörü, WHERE yan tümcelerinde karşılaştırma için kullanılır. Bu bir karşılaştırma operatörüdür. Bir değer bir değer aralığında olup olmadığını test etmek için

kullanabilirsiniz. Değer belirtilen aralıktaysa, sorgu bu aralıkta kalan tüm kayıtları döndürür.

```
WHERE test_expression BETWEEN low_expression AND high_expression
```

Önemli: BETWEEN operatörü kapsayıcıdır. Özel bir aralık belirtmek için büyüktür (>) ve küçüktür işleçlerini (<) kullanın.

```
WHERE test_expression >= low_expression AND test_expression <= high_expression
```

Bu tabloyu hatırlıyoruz;

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	11/24/2017
70950	Rodney	Weaver	87000	Project Manager	Male	12/20/2018
97927	Billie	Lanning	67000	Web Developer	Female	6/25/2018
67323	Lisa	Wiener	75000	Business Analyst	Female	8/9/2018
17679	Robert	Gilmore	110000	Operations Director	Male	9/4/2018
76589	Jason	Christian	99000	Project Manager	Male	1/21/2019
51821	Linda	Foster	95000	Data Scientist	Female	4/29/2019
71329	Gayle	Meyer	77000	HR Manager	Female	6/28/2019
49714	Hugo	Forester	55000	IT Support Specialist	Male	11/22/2019
30840	David	Barrow	85000	Data Scientist	Male	12/2/2019

Bunun üzerinden bir örnekle açıklayacak olursak;

```
SELECT * FROM employees WHERE salary BETWEEN 80000 AND 90000;
```

Sonuç şu şekilde olacaktır:

emp_id	first_name	last_name	salary	job_title	gender	hire_date
26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20

Yukarıdaki kodu şu şekilde de yazabiliriz:

```
SELECT * FROM employees WHERE salary >= 80000 AND salary <= 90000;
```

NOT BETWEEN

BETWEEN operatörünün sonucunun tersini alabiliriz NOT BETWEEN operatörüyle. Yani o aralığı kapsamayanlar çağrılacaktır sorgu sonunda.

```
WHERE test_expression NOT BETWEEN low_expression AND high_expression;
```

```
SELECT * FROM employees WHERE salary NOT BETWEEN 80000 AND 90000;
```

Alacağımız sonuç bir önceki BETWEEN sonucunun tam zıttı yani şöyle olacaktır;

emp_id	first_name	last_name	salary	job_title	gender	hire_date
17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

BETWEEN operatörünü tarih aralıkları için de kullanabiliriz:

employees table							
	emp_id	first_name	last_name	salary	job_title	gender	hire_date
1	17679	Robert	Gilmore	110000	Operations Director	Male	2018-09-04
2	26650	Elvis	Ritter	86000	Sales Manager	Male	2017-11-24
3	30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
4	49714	Hugo	Forester	55000	IT Support Specialist	Male	2019-11-22
5	51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
6	67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
7	70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
8	71329	Gayle	Meyer	77000	HR Manager	Female	2019-06-28
9	76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
10	97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

Bu tablo üzerinden bir çağırma yapacak olursak:

1 Haziran 2018'den 31 Mart 2019'a kadar şirkete katılan çalışanları bulmaya çalıştığımızı varsayalım. Ayrıca artan düzende işe alma tarihine göre sıralamak istiyoruz.

```
SELECT * FROM employees WHERE hire_date BETWEEN 01-06-2018 AND 31-03-2019
ORDER BY hire_date;
```

Çıktımız şu şekilde olacaktır:

emp_id	first_name	last_name	salary	job_title	gender	hire_date
97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25
67323	Lisa	Wiener	75000	Business Anal	Female	2018-08-09
17679	Robert	Gilmore	110000	Operations Di	Male	2018-09-04
70950	Rodney	Weaver	87000	Project Manag	Male	2018-12-20
76589	Jason	Christian	99000	Project Manag	Male	2019-01-21

Bir de 'capcap reis'den dinleyelim:

https://www.youtube.com/watch?time_continue=44&v=58-_A_RotfQ&feature=emb_title

IN OPERATÖRÜ

Bu operatör ile bir kolon içinde yer alan verilerin istediğimiz veriler olup olmadığını karşılaştırma yaparak kontrol etmesini sağlarız.

```
HERE column_name IN (value_list)

--yani:

SELECT *
FROM employees
WHERE job_title = 'Data Scientist'
OR
job_title = 'Business Analyst';

--gibi
```

'yani' 'gibi' sorgusu bize aşağıdaki gibi bir output basacaktır:

emp_id	first_name	last_name	salary	job_title	gender	hire_date
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09

Ancak bu tarz eşleştirmelerde kullanabileceğimiz daha iyi bir operatör vardır; **IN OPERATÖRÜ**

kodu şu şekilde yazabiliriz:

```
SELECT *
FROM employees
WHERE job_title IN ('Data Scientist', 'Business Analyst');
```

IN operatörünü kullanmak kodunuzu kısaltıyor.

Başka bir örneklendirme yapalım:

```
SELECT *
FROM employees
WHERE job_title = 'Data Scientist'
OR
job_title = 'Business Analyst'
OR
job_title = 'Project Manager'
OR
job_title = 'Web Developer';
```

Sonuç şöyle olacaktır:

emp_id	first_name	last_name	salary	job_title	gender	hire_date
30840	David	Barrow	85000	Data Scientist	Male	2019-12-02
51821	Linda	Foster	95000	Data Scientist	Female	2019-04-29
67323	Lisa	Wiener	75000	Business Analyst	Female	2018-08-09
70950	Rodney	Weaver	87000	Project Manager	Male	2018-12-20
76589	Jason	Christian	99000	Project Manager	Male	2019-01-21
97927	Billie	Lanning	67000	Web Developer	Female	2018-06-25

Aynı sonucu elde etmek için ve tabii ki **daha okunabilir bir kodlama için** şöyle de yazabiliriz:

```
SELECT *
FROM employees
WHERE job_title IN ('Data Scientist', 'Business Analyst', 'Project Manager', 'Web Developer');
```

NOT IN

Hangi değerleri bir listeye dahil etmek istemediğimizi biliyorsak, IN ile NOT keyword'ü kullanabiliriz.

IN Operatöründen hemen önce kullanılır.

```
SELECT *
FROM employees
WHERE job_title
NOT IN ('Operations Director', 'HR Manager', 'Sales Manager');
```


dediğimizde bu sefer 'Operations Director', 'Hr Manager', 'Sales Manager'i dahil etmeyecektir. Harici kalanları çağıracaktır.

LIKE OPERATÖRÜ

SQL **LIKE** komutu, joker karakter operatörleri kullanarak bir değeri benzer değerlerle karşılaştırmak için kullanılır. LIKE komutu ile birlikte kullanılan iki joker karakter vardır.



- **Yüzde işareti (%)**
- **Alt çizgi (_)**

Yüzde işareti sıfır, bir veya birden çok karakteri temsil eder. Alt çizgi, tek bir sayıyı veya karakteri temsil eder. Bu joker karakterler birlikte de kullanılabilirler.

Bazen veritabanımızda yer alan verilerin içeriğinin bir kısmını biliriz. Ya da bir kısmı benzer olan verilerin listelenmesini isteyebiliriz. Örneğin açıklamalardan oluşan bir veri içinde aynı kelimelere sahip olan verilerin listelenmesini isteyebiliriz. Çeşitli joker karakter yardımları ile istediğimiz kriterdeki verilerin listelenmesini sağlayabiliriz. Tüm bunlar için LIKE deyimini kullanırız.

- SQL LIKE komutu, bir sütundaki belirli bir veriyi aramak için bir WHERE koşuluyla kullanılır.

Örneklendirelim;

 LIKE komutları	 Açıklama
<u>WHERE Soyad LIKE 'a%'</u>	"a" ile başlayan tüm verileri bulur.
<u>WHERE Soyad LIKE '%a'</u>	"a" ile biten tüm verileri bulur.
<u>WHERE Soyad LIKE '%or%'</u>	Herhangi bir konumda "or" bulunan verileri bulur.
<u>WHERE Soyad LIKE '_r%'</u>	İkinci konumda "r" bulunan tüm verileri bulur.
<u>WHERE Soyad LIKE 'a_ %'</u>	"a" ile başlayan ve en az 2 karakter uzunluğunda olan tüm verileri bulur.

Aa LIKE komutları	Açıklama
<u>WHERE Soyad LIKE 'a_ %'</u>	"a" ile başlayan ve en az 3 karakter uzunluğunda olan tüm verileri bulur.
<u>WHERE Soyad LIKE 'a%o'</u>	"a" ile başlayan ve "o" ile biten tüm verileri bulur.

Bu arada syntax'ımızı da yazmayı unutmayalım;

```
SELECT column_name(s)
FROM table_name
WHERE column_1 LIKE pattern;
```

—

****COUNT'a geldiğinde düzenlersin şimdilik böylece dursun önemli video;

<https://www.youtube.com/watch?v=yY0Go4a4IZU&t=158s>

—

MIN Fonksiyonu

MIN fonksiyonu sütunlardaki minimum değeri çevirir.

```
SELECT MIN(Column_name)
FROM table_name;
```

- MIN fonksiyonu NULL olan satırı yok sayar. Böylece NULL haricinde kalan minimum satırı çevirir.
- ORDER BY ve LIMIT keyword'ları ile de aynı işlemi yapabilirsiniz. Aynı sonucu verir.
- WHERE yan cümlesiyle birleştirilebilir.

MAX Fonksiyonu

MAX fonksiyonu seçili kolondaki maksimum değeri döndürür.

```
SELECT MAX(column_name)
FROM table_name;
```

- MIN işleminde olduğu gibi MAX'da da WHERE yan cümlesiyle birleştirilebilir.

Örneğin;

```
SELECT MAX(salary) AS highest_salary
FROM employees
WHERE gender = 'Male';
```

- ORDER BY ve LIMIT keyword'ları ile de aynı işlemi yapabilirsiniz. Aynı sonucu verir.

SUM Function

SUM işlevi sayısal bir sütunun **toplamını** döndürür.

```
SELECT SUM(column_name)
FROM table_name;
```

AVG Function

AVG fonksiyonu numeric bir sütunun (sayısal) **ortalamasını** hesaplar.

Syntax;

```
SELECT AVG(column_name)
FROM table_name;
```

GROUP BY Yan Cümlesi

- GROUP BY yan tümcesi, satırları özet satırlar halinde gruplandırır. Her grup için bir değer döndürür ve genellikle toplama işlevleriyle (COUNT, MAX, MIN, SUM, AVG) kullanılır.
- WHERE yan tümcesinin kullanılması durumunda, GROUP BY yan tümcesi WHERE yan tümcesinden sonra gelmelidir.
- GROUP BY'den hemen sonra sütunu veya virgülle ayrılmış sütunların bir listesini belirtiriz.
- SELECT içindeki tüm toplu olmayan ifadeler GROUP BY yan tümcesine dahil edilmelidir.
- GROUP BY, veri grubu başına yalnızca bir sonuç döndürür.
GROUP BY Cümlesi her zaman WHERE Cümlesini takip eder.
GROUP BY Cümlesi her zaman ORDER BY'den önce gelir.

GROUP BY with COUNT Function

```
SELECT gender, COUNT(job_title)
FROM employees
WHERE job_title = 'Data Scientist'
GROUP BY gender;
```

GROUP BY with MIN&MAX Functions

```
SELECT gender,
MAX(salary) AS max_salary
FROM employees
GROUP BY gender
ORDER BY max_salary DESC;
```

azalan düzende her grup için maksimum maaşları sıraladık.

GROUP BY with SUM&AVG Functions

```
SELECT gender, SUM(salary) AS total_salary
FROM employees
GROUP BY gender;
```

```
SELECT gender,
SUM(salary) AS total_salary
FROM employees
GROUP BY gender
ORDER BY total_salary DESC;
```

JOIN

JOINleri anlamaya başlamadan önce mantığın oturması için **key** meselesine göz atmakta fayda var. Çünkü JOIN meselesi key kavramıyla ilintili. Derinlemesine irdelemeyi sonraya bırakarak kabaca değinecek olursak;

KEY, SQL içerisinde iki veya daha fazla tabloyu ilişkilendirmek için kullanılan kısıtlamalardır. Bunlar **Primary Key**, **Unique** ve **Foreign Key**'dir.

Keyler karışıkların önüne geçmek ve ufak yazım yanlışlarından doğabilecek hataların önüne geçmek için kullanılır diyebiliriz. Bunlar çok kaba tabirler tabii ki ama anlamayı kolaylaştırır diye umuyorum. Az sonra joinlerde yapacağımız da bu olacak, işimizi kolaylaştırmak(normalizasyon) için ana tabloda her şeyi tek tek yazmak yerine harici tablolar oluşturup veriyi oradan çağıracağız. Meselemiz bu.

Böylece hem sistemli ve steril bir şekilde ilerlemiş olacağız hem de sürekli sürekli aynı şeyleri yapıp kendimizi tekrar etmemiş olacağız. Bu altın kuralı yazmanın tam zamanı sanırım:

DRY : DON'T REPEAT YOURSELF!

Foreign Key:

Foreign key kısıtlamasının kullanım amacı veri bütünlüğünü sağlamaktır.

Veri tabanı tasarımı sırasında (neyi nerden geldiğine kafa yormadan önce neyin nasıl yapıldığını öğrenmekte fayda var diye düşünüyorum. Bu yüzden ilerlemeye devam etmeden evvel bir boşluktan **veri tabanı nasıl tasarlanır**a bakmakta fayda var diye düşünüyorum) tablolar çeşitli parçalara ayrılarak veri tekrarının önüne geçilir.

Parçalara ayrılan tabloları anlamlı bir şekilde birleştirmek için ortak bir sütun belirlenebilir.

Ortak sütun parçaların birleştirilmesinde faydalı olsa da bir tablodaki satırın silinmesi veya güncellenmesi sırasında veri bütünlüğü bozulabilir.

Bu durumda tablolar arası Foreign Key veya Uzak Anahtar denilen ilişki kurularak veri bütünlüğü sağlanmış olur.

UNIQUE:

Unique key kısıtlaması her bir verinin birbirinden farklı olmasını sağlar. Örnek olarak günlük hayatta çokça kullanılan kimlik numaraları ve sıra numaraları verilebilir.

Tablodaki herhangi bir sütuna Unique kısıtlaması eklenerek, benzer değerlerin eklenmesi engellenir.

Primary Key:

Primary key kullanım amacı her bir satırın farklı olmasını garantilemektir.

Unique kısıtlamasından farkı değer olarak NULL içermemesi ve her tabloya sadece bir tane eklenebilmesidir.

ve tekrar **JOINLER**

SQL JOIN ifadesi, tabloları birleştirmek için kullanılır. JOIN iki veya daha fazla tabloyu birleştirmek için kullanılır.

```
SELECT tablo_adi.sutun_adi, ...
FROM tablo_A
{INNER JOIN | LEFT JOIN | RIGHT JOIN} tablo_B ON tablo_A.sutun_adi = tablo_B.sutun_adi;
```

Örnek tablo;

kat_id	kat_adi
1	Bilgisayar
2	Telefon
3	Elektronik

urunler tablosunda bulunan veriler:

urun_id	urun_adi	urun_fiyat	kat_id
1	Masaüstü Bilgisayar	1799	1
2	Akıllı Telefon	799	2
3	Dizüstü Bilgisayar	2199	1
4	SQL Kitabı	59	99

Örneklerde bulunan **kategoriler** tablosu **kat_id** ile **urunler** tablosundaki **kat_id** adlarına ve değerlerine dikkat edin.

İki parçaya ayrılan tabloları birleştirerek kategorilerde bulunan ürünleri listeleyelim.

Bunun için join kullanabileceğimiz gibi iki tabloyu aşağıdaki gibi sırayla yazarak birleştirebiliriz.

```
SELECT * FROM kategoriler, urunler;
```

Bu tarz birleştirme türüne **CROSS JOIN** adı verilir.

Birleştirme için bir koşul belirtilmediğinden **kategoriler** tablosundaki her satır, **urunler** tablosundaki tüm satırlarla eşleşir. Koşul belirterek sadece ilişkili olan sütunları birleştirmek için **WHERE** koşulu kullanılabilir. Bu birleştirme türü **INNER JOIN** olarak adlandırılır.

```
SELECT * FROM kategoriler, urunler WHERE urunler.kat_id = kategoriler.kat_id;
```

SQL Join Türleri

- **(INNER) JOIN:** İki tablodaki eşleşen kayıtlar için kullanılır. Ortak değere sahip tabloları birleştirmek için kullanılır.

SQL inner join iki veya daha fazla tablodaki ilişkili değerleri seçmek/birleştirmek için kullanılır.

```
SELECT tablo_adi.sutun_adi, ...  
FROM tablo_A  
INNER JOIN tablo_B ON tablo_A.sutun_adi = tablo_B.sutun_adi;
```

AS komutu ile takma isim atayarak kullanmak yaygın olarak yapılan bir işlemdir;

```
SELECT tablo_adi.sutun_adi, ...  
FROM tablo_A AS t1  
INNER JOIN tablo_B AS t2 ON t1.sutun_adi = t2.sutun_adi;
```

INNER JOIN ikinde fazla tabloyu eşleştirmek için de kullanılır. Bunun için tekrar INNER JOIN kullanmak yeterli olacaktır;

```
SELECT tablo_adi.sutun_adi, ...  
FROM tablo_A  
INNER JOIN tablo_B ON tablo_A.sutun_adi = tablo_B.sutun_adi  
INNER JOIN tablo_C ON tablo_A.sutun_adi = tablo_C.sutun_adi;
```

- **LEFT (OUTER) JOIN:** İki tablodaki eşleşen kayıtlar ve eşleşmeyen sol kayıtlar için kullanılır.

SQL left join ilk (sol) tablodaki tüm satırları ve koşul ile belirtilen ikinci (sağ) tablodaki satırları seçmek/birleştirmek için kullanılır.

İlk (sol) tablodaki değer ile ikinci (sağ) tablodaki eşleşmeyen değer olursa ikinci (sağ) tablodaki değerler NULL değerini alır.

Özetle; SQL LEFT JOIN ifadesi, ilk seçilen tablodaki tüm satırları ve ikinci seçilen tablodaki eşleşen satırları birleştirmek için kullanılır.


```
SELECT tablo_adi.sutun_adi, ...
FROM tablo_A
LEFT JOIN tablo_B ON tablo_A.sutun_adi = tablo_B.sutun_adi;
```

Bildiğiniz gibi **AS** kullanımı takma ad vermek için sıklıkla kullanılmaktadır. Daha doğrusu takma isim verme işlemi uygulanan bir işlemdir. AS kullanmadan da tablo ya da sütun isminden sonra direk vereceğimiz takma adı yazarak da bu işlemi yapabiliyoruz. Ama tekrar hatırlatmakta fayda var, karışıklıkların önünü almak için AS ifadesini orada görmek daha sağlıklı olacaktır.

```
SELECT tablo_adi.sutun_adi, ...
FROM tablo_A AS t1
LEFT JOIN tablo_B AS t2 ON t1.sutun_adi = t2.sutun_adi;
```

Son olarak şunu ekleyeyim **outer** ibaresi farklı veri tabanlarındaki kullanımına göre kullanılıyor ya da kullanılmıyor. Standart kullanımda pek kullanılmıyor. Ancak aynı şeyi temsil ediyor. Yani Left Join yazsak da Left Outer Join yazsak da aynı kapıya çıkıyor. Right ve Full için de aynı durum geçerli.

- **RIGHT (OUTER) JOIN:** İki tablodaki eşleşen kayıtlar ve eşleşmeyen sağ kayıtlar için kullanılır.

SQL RIGHT JOIN ifadesi, ikinci seçilen tablodaki tüm satırları ve ilk seçilen tablodaki eşleşen satırları birleştirmek için kullanılır.

İkinci (sağ) tablodaki tüm satırları ve koşul ile belirtilen ilk (sol) tablodaki satırları seçmek/birleştirmek için kullanılır.

İkinci (sağ) tablodaki değer ile ilk (sol) tablodaki eşleşmeyen değer olursa ilk (sol) tablodaki değerler NULL değerini alır.

```
SELECT tablo_adi.sutun_adi, ...
FROM tablo_A
RIGHT JOIN tablo_B ON tablo_A.sutun_adi = tablo_B.sutun_adi;

--AS ile takma ad vererek kullanımı:

SELECT tablo_adi.sutun_adi, ...
FROM tablo_A AS t1
RIGHT JOIN tablo_B AS t2 ON t1.sutun_adi = t2.sutun_adi;
```

-
- **FULL (OUTER) JOIN:** İki tablodaki eşleşen kayıtlar ve eşleşmeyen sol ve sağ kayıtlar için kullanılır. LEFT ve RIGHT JOIN birleşimidir. Bütün kayıtları çağırır.

SQL JOIN ayrıca ekleme, güncelleme ve silme işlemlerinde de kullanılabilir.