

Join Zoom Meeting: <https://zoom.us/j/99977279082?pwd=ZW1LenZhYWxLVlkxWlBlem1YR3ZjQT09>

Meeting ID: 999 7727 9082

Passcode: 851093

SQL-1

05.10.2020

Veritabanı genellikle elektronik olarak bir bilgisayar sisteminde depolanan yapılandırılmış bilgi veya veriden oluşan düzenli bir koleksiyondur.

Veri tabanı genellikle bir Veri Tabanı Yönetim Sistemi DBMS (**D**ata**B**ase**M**anagement**S**ystem) ile kontrol edilir.

Çoğu veri tabanında veri yazma ve sorgulama için yapılandırılmış sorgu dili **SQL** (**S**tructured **Q**uery **L**anguage) kullanılır.

SQL, verileri yönetmek ve tasarlamak için kullanılan bir dildir. SQL, kendisi bir programlama dili olmamasına rağmen birçok kişi tarafından programlama dili olarak bilinir. SQL herhangi bir veri tabanı ortamında kullanılan bir alt dildir. SQL ile yalnızca veri tabanı üzerinde işlem yapılabilir; veritabanlarında bulunan sistemlere bilgi ekleme, bilgi değiştirme, bilgi çıkarma ve bilgi sorgulama için kullanılmaktadır. Özellikle de ilişkisel veritabanı sistemleri üzerinde yoğun olarak kullanılmaktadır. SQL'e özgü cümleler kullanarak veri tabanına kayıt eklenebilir, olan kayıtlar değiştirilebilir, silinebilir ve bu kayıtlardan listeler oluşturulabilir.

Structured Query Language (SQL) - Yapılandırılmış Sorgu Dili - Strukturierte Abfragesprache.

Database' in Faydaları;

1. Yüksek miktarda bilgi depolanabilir.
2. Oluşturmak, okumak, değiştirme ve silme kolaylığı. **Create, Read, Update, Delete (CRUD)**
3. Girişin kolay ve kontrollü olması.
4. Dataya ulaşım kolaylığı.
5. Güvenlik

Dersler başlamadan; ilk olarak <https://livesql.oracle.com/> sitesinden kullanıcı hesabi oluşturulmalı.

Start Coding Now linkine tiklanır. Kullanici hesabi yoksa olusturulur. Varsa Sign in yapılır.

SQL Worksheet penceresine asagidaki script yapistirilir;

```
CREATE TABLE department
(
  id                number(5) constraint pk_department primary key,
  name              varchar2(50),
  monthly_budget    number(8,2),
  last_employee_id  number(5)
);

insert into department values (1,'ACCOUNTING',20000,8);
insert into department values (2,'MARKETING',15000,9);
insert into department values (3,'INFORMATION TECHNOLOGY',30000,10);
insert into department values (4,'HUMAN RESOURCES',25000,13);
insert into department values (5,'REGULATORY AFFAIRS',5000,null);
insert into department values (6,'CUSTOMER SERVICE',2000,null);

Ardından Run edilir. Tablo olarak görebilmek için "select * from department;" yazılır.
```

| ID | NAME | MONTHLY_BUDGET | LAST_EMPLOYEE_ID |
|----|------------------------|----------------|------------------|
| 1 | ACCOUNTING | 20000 | 8 |
| 2 | MARKETING | 15000 | 9 |
| 3 | INFORMATION TECHNOLOGY | 30000 | 10 |
| 4 | HUMAN RESOURCES | 25000 | 13 |
| 5 | REGULATORY AFFAIRS | 5000 | - |
| 6 | CUSTOMER SERVICE | 2000 | - |

Oluşan tabloyu silmek için **Schema**'ya gidilir, tablo tıklanır, Actions menüsünden **Drop** seçilir. (delete yada remove kullanılmaz) Gelen code run edilirse tablo silinmiş olur.

```
select E.first_name, E.salary from employees E
join departments D on E.department_id = D.department_id
join locations L on D.location_id = L.location_id
where E.salary > (select max(salary) from employees E
                  join departments D on E.department_id = D.department_id
                  join locations L on D.location_id = L.location_id
                  where L.city = 'Toronto')
and L.city <> 'Toronto';
```

```
select max(salary) from employees E
                  join departments D on E.department_id = D.department_id
                  join locations L on D.location_id = L.location_id
                  where L.city = 'Toronto';
```

```
select E.first_name, E.salary from employees E
join departments D on E.depatment_id = D.department_id
join locations L on D.location_id = L.location_id
where E.salary > (select max(salary) from employees E
                  join departments D on E.depatment_id = D.department_id
                  join locations L on D.location_id = L.location_id
                  where L.city = 'Toronto')
and L.city <> 'Toronto';
```

```
select max(salary) from employees E
                  join departments D on E.depatment_id = D.department_id
                  join locations L on D.location_id = L.location_id
                  where L.city = 'Toronto';
```

Database Validation (Doğrulama) Testi

Tester olarak bizler; **Database Validation (Doğrulama) Testi** yaparız, database oluşturmayız, kullanıcılara yetki vermeyiz, raporlama yapmayız, tablo silmeyiz. Buna **End To End (E2E) Testing** de denir.

Datayı User Interface (UI) kullanarak, SQL kodlarını kullanarak yada API kodlarını kullanarak da yollasak; Üç çeşit Database Validation Testi vardır;

1. Datayı UI dan arama fonksiyonunu kullanarak doğrulama (Selenium)
2. Datayı SQL kodlarını kullanarak doğrulama (SQL + Selenium)
3. Datayı API kodlarını kullanarak doğrulama (API + Selenium)

Application Programming Interface (API) bir uygulamaya ait yeteneklerin, başka bir uygulamada da kullanılabilmesi için yeteneklerini paylasan uygulamanın sağladığı arayüzdür.

Data Base Management System (DBMS) Veri tabanlarını yönetmek, kullanmak, geliştirmek ve bakımını yapmak için kullanılan yazılımlara denir.

- Database'e erişimi düzenler
- **Create, Read, Update ve Delete (CRUD)** işlemlerini düzenler
- Data güvenliğini sağlar
- Formlar oluşturur ve işler
- Sorgular oluşturur ve iletir
- Raporlar oluşturur ve işletir
- Uygulamayı kontrol eder
- Diğer uygulamalarla (Application) iletişimi sağlar

SQL'de datalar **Table**'larda oluşturulur;

- Başlığa **Headers**,
- Satırlara **Record**,
- Sütunlara **Field** denilir.

Relational Database (İlişkili Tablolar)

- SQL tablolar, dataları ilişkili tablolarda depolar
- Tablolar arasında ilişkiler net olmalıdır
- Tablolar arasında geçiş kolay olmalıdır
- Tablolar ve ilişkilerin bütününe "**Schema**" denir.
- Relational Databases, SQL Databases (Structured Query Language) olarak da adlandırılır.

En çok kullanılan SQL'ler (Relational Databases);

Microsoft SQL Server, MySQL Server, PostgreSQL Server ve Oracle PL/SQL

Non Relational Database – NoSQL Database

- SQL veritabanı verilerle çalışırken yapısal sorgu dili kullanır. Veri yapısını belirlemek için önceden tanımlanmış şemalar gerektirir.
- Tablolar ile çalışmaz, onun yerine doküman dosyalarının içinde depolanır.
- **NoSQL veritabanı** ise verilerle çalışırken Yapılandırılmamış Sorgu Dili kullanır.

SQL komutları şunlardır:

1. Veri Tanımlama Dili (Data Definition Language- DDL)

DDL komutları ile veritabanı ve tabloları oluşturma, değiştirme ve silme işlemleri yapılır. (Bunu Tester'lar yapmaz)

CREATE TABLE tablo_adi

Yeni bir tablo oluşturmak için kullanılır. Alan isimleri yazılırken sona virgöl konulur ve son satır olan işlemimizde virgöl konmadan parantez kapatılır. Ör;

```
CREATE TABLE tabloilceler (  
  ilceNo      mediumint(8) unsigned DEFAULT '0' NOT NULL,  
  ilce        varchar(30) NOT NULL,
```

```

postakodu    varchar(5),
ilceTel      char(3),
plakaKodu    char(2) NOT NULL
)

```

ALTER TABLE tablo_adi

Yeni bir sütun eklemek, sütunun tipini veya uzunluğunu değiştirmek/güncellemek vb. yapısal değişiklikler yapılması için kullanılır.

DROP TABLE tablo_adi

Tabloyu içerisindeki verilerle birlikte siler.

// TRUNCATE TABLE tablo_adi

Tablodaki tüm verileri siler, tablo yapısını korur.:

//CREATE VIEW görüş_adi

Görüntü oluşturmak için kullanılır

//DROP VIEW görüş_adi

Görüntüyü siler

//CREATE INDEX indeks_adi

Tablonun (en azından bir) sütun adı üzerinde indeks oluşturmak için kullanılır.

//DROP INDEX indeks_adi

Oluşturulan indeksleri veri tabanından kaldırmak için kullanılır.

2. Veri Sorgulama Dili (Data Query Language- DQL)

DQL içindeki SELECT komutu ile veritabanında yer alan mevcut verilerin bir kısmını veya tamamını, tanımlanan koşullara bağlı olarak alır.

SELECT deyim

```

SELECT ilçe, postakodu FROM tablolliceler WHERE plakaKodu = '34'
İstanbul'un ilçeleri ile posta kodlarını gösterir

```

3. Veri Kullanma Dili (Data Manipulation Language- DML)

DML komutları ile veritabanlarında bulunan verilere işlem yapılır. DML ile veritabanına yeni kayıt ekleme, mevcut kayıtları güncelleme ve silme işlemleri yapılır.

UPDATE deyim

```

UPDATE tablolliceler SET postakodu = '06720' WHERE ilçe = 'Bala'
Bala'nın posta kodunu değiştirir/günceller

```

INSERT deyim

```

INSERT INTO tablolliceler VALUES (, 'Yenişehir', , , '53')
Yeni veriler ekler

```

DELETE deyim

```

DELETE FROM tablolliceler WHERE plakaKodu = '53'
plakaKodu 53 olan bütün verileri siler

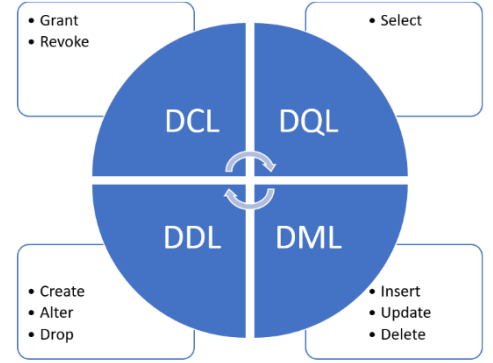
```

4. Veri Kontrol Dili (Data Control Language- DCL)

DCL komutları ile kullanıcılara veritabanı ve tablolar için yetki verilir veya geri alınır: (Bunu Tester'lar yapmaz)

GRANT: Bir kullanıcıya yetki vermek için kullanılır.

REVOKE: Bir kullanıcıya verilen yetkiyi geri almak için kullanılır.



Primary Key

- Her bir satır için eşsiz bir veridir. Primary Key tabloyu oluşturan kişi tarafından belirlenir.
- Unique 'dir ama her unique data **Primary Key** değildir.
- Duplication kabul etmez
- Null kabul etmez
- Bir tabloda yalnızca bir tane **Primary Key** olabilir.
- Her tabloda Primary Key olması zorunlu değildir.
- Primary Key her türlü datayı içerebilir (Sayı, String...).
- T.C. kimlik numarası, ISBN, email hesabi gibi gerçek verilere "**Natural Primary Key**" denir.
- Genel olarak kayıt eklenmeden önce üretilen sıra numarası gibi sayısal değerlere "**Surrogate Primary Key**" denir.

- Relational veri tabanlarında (relational database management system) mutlaka **Primary Key** olmalıdır.

Foreign Key

- İki tablo arasında relation (ilişki) oluşturmak için kullanılır.
- Başka bir tablodaki Primary Key ile ilişkilendirilmiş (bağlı) olmalıdır.
- Bir tabloda birden fazla Foreign Key olabilir.
- Foreign Key Null değeri kabul eder.
- Foreign Key olarak tanımlanan field'da tekrarlar (duplication) olabilir.
- Referenced table (bağlanılan tablo, Primary Key'in olduğu Tablo) "parent table" olarak adlandırılır. Foreign Key olan tabloya "child table" denir.
- *** "Parent Table" olmayan bir id'ye sahip datayı "Child Table"a ekleyemezsiniz.
- *** "Child Table" i silmeden "Parent Table" i silemezsiniz. Önce "Child Table" silinir, sonra "Parent Table" silinir.

SQL-2

06.10.2020

Composite Key birden fazla field (kolon)'in kombinasyonu ile oluşturulur. Tek başına bir kolon Primary Key olma özelliklerini taşııyorsa, bu özellikleri elde etmek için birden fazla kolon birleştirilerek **Primary Key** oluşturulur.

Primary Key'in Unique Key'den farkları;

1. Bir Tabloda sadece 1 tane olur
2. NULL değeri kabul etmez

Unique Key'in Primary Key'den farkları;

1. Bir tabloda birden fazla olabilir.
2. Sadece 1 tane NULL değeri kabul eder

Primary Key ve Unique Key'in ortak özelliği; Duplication (Çift Kullanım)'a izin vermezler.

Tablolarla arası Related üç şekilde olur;

- 1- One to One Relation
- 2- One to Many Relation
- 3- Many to Many Relation

String, Numeric, Date ve BLOB olmak üzere dört tane SQL Data Types vardır.

A. En çok kullanılan **String Data Types 4** çeşittir; **char(size)**, **nchar(size)**, **varchar2(size)** ve **nvarchar2(size)**

char(size); TC-Kimlik, tel no gibi uzunluğu sabit String datalarda kullanılır.

nchar(size); genellikle farklı dillerdeki karakterler gibi Unicode datalarda kullanılır.

varchar2(size); isim gibi uzunluğu sabit olmayan String datalarda kullanılır.

nvarchar2(size); değişken uzunluktaki Stringlerin Unicode değerleri için kullanılır.

1. nchar'ın dezavantajı iki kati byte kullanır.
2. Uzunluğu biliniyorsa Char kullanacağız (T.C. kimlik No gibi)
3. Uzunluğu sabit değilse varchar2 kullanacağız (isim, şehir vb)

B. Sayılar için Numeric Data Types kullanılır.

number(**p,s**) şeklinde kullanılır.

“**p**recision” (**p**) sayıdaki rakam sayısıdır.

“**s**cale” (**s**) virgülden sonra kaç rakam olduğunu belirler.

Örneğin: 1234,56 ==> Precision: 6, Scale: 2 => number(6,2)

C. Tarihler ve zamanı depolamak için Date Data Types kullanılır.

Saniyenin virgüllü kısmını da alır. Standart “Date Format”, “dd - MMM - yy”. Örneğin ‘13 - Apr – 20’
Tarih formatını “ALTER SESSION SET NLS_DATE_FORMAT = “YYYY-MM-DD” kodu kullanılarak
değiştirilebilir. Koddan sonra tarih 2020 – 04 – 13 olur.

D. Resim, video, ses gibi dataları binary formatına çevirerek depolamak için BLOB Data Types kullanılır.

“**B**inary **L**arge **O**bjects” demektir.

Practices;

```
-- P1 id, name, grade, adres ve last_update oluşan student_table oluşturunuz
```

```
CREATE TABLE student_table
(
  id char(11),
  name varchar2(50) NOT NULL,
  grade number(5,2),
  adres varchar2(100),
  last_update date,
  CONSTRAINT id_pk PRIMARY KEY (id)
);
```

```
--student_table'dan sadece name ve grade fieldlari alarak yeni bir tablo olusturun
```

```
CREATE TABLE student_grade
AS
SELECT name, grade
FROM student_table;
```

```
-- P2 tedarikci_id, tedarikci_ismi, tedarikci_adres, ve ulasim_tarihi olan
tedarikciler
```

```
CREATE TABLE tedarikciler
(
  tedarikci_id char(11),
  tedarikci_ismi varchar2(50),
  tedarikci_adres varchar2(100),
  ulasim_tarihi date
);
```

```
--
```

```
CREATE TABLE tedarikçi_ziyaret
AS
```

```
SELECT tedarikci_ismi, ulasim_tarihi
FROM tedarikçiler;
```

SQL-3

13.10.2020

- SQL’de Code yazarken bir field “boş bırakılmasın” diyorsa “NOT NULL”,
- SQL’de Code yazarken bir field “tekrarlı değer kabul etmesin/tekrarsız” diyorsa “UNIQUE”,
- SQL’de Code yazarken yorum yazılacaksa başına “--” işareti,
- Bir field Primary Key atanacaksa; Data Type’den sonra “PRIMARY KEY” yazılır. (Ör; id number(9) PRIMARY KEY,)
- Birden fazla değer Primary Key atanacaksa;
CONSTRAINT tablo_ismi_pk **PRIMARY KEY** (field1, field2, id, ...) yazılır.
- Bir “Tabloya Data Ekleme” için;
INSERT INTO tablo_ismi **VALUES** (değer1, değer2,...); yazılır.
- Bir “Tabloda bazı field’lara Data Ekleme” için;
INSERT INTO tablo_ismi (column1, column2) **VALUES** (değer1, değer2); yazılır.
- **INSERT INTO** kodunu kullanarak bir tabloya data eklemek istediğinizde, **CONSTRAINT**’lere (kısıtlama) uymak zorundayız. Örneğin; **NOT NULL** yazan field’a bir değer atamak zorundayız aksi takdirde hata alınır.
- Oluşturulan bir tabloyu ekrana yazdırmak için; **SELECT * FROM** tablo_ismi; yazılır.

--P3 “şehirler” isimli bir Table oluşturun. Tabloda “alan_kodu”, “isim”, “nufus” field’lari olsun. Isim field’i bos birakilmasin.

--1.Yontemi kullanarak “alan_kodu” field’ini “Primary Key” yapin

```
CREATE TABLE sehirler
(
  alan_kodu char(3) PRIMARY KEY,
  isim varchar2(50) NOT NULL,
  nüfus number(7)
);
```

--P4 “ogretmenler” isimli bir Table oluşturun. Tabloda “id”, “isim”, “brans”, “cinsiyet” field’lari olsun.

--Id field’i tekrarli deger Kabul etmesin.

--2.Yontemi kullanarak “id ve isim” field’lerinin birlesimini “primary key” yapin

```
CREATE TABLE ogretmenler
(
  id char(10) UNIQUE,
  isim varchar2(50) NOT NULL,
  brans varchar2(20),
  cinsiyet varchar2(10),
  CONSTRAINT ogretmenler_pk PRIMARY KEY (id, isim)
);
```

- Bir tabloya Foreign Key atanacaksa;

CONSTRAINT tablo_ismi_fk **FOREIGN KEY** (field3) **REFERENCES** diğer_ tablo_ismi (field3) yazılır.

```
--P5 "tedarikciler" isimli bir tablo olusturun. Tabloda "tedarikci_id",  
"tedarikci_ismi", "iletisim_isim" field'leri olsun ve  
--"tedarikci_id" yi Primary Key yapin.  
--"urunler" isminde baska bir tablo olusturun "tedarikci_id" ve "urun_id"  
field'leri olsun ve  
--"tedarikci_id" yi Foreign Key yapin.
```

```
CREATE TABLE tedarikciler  
(  
tedarikci_id char(10) PRIMARY KEY,  
tedarikci_ismi varchar2(50),  
iletisim_isim varchar2(50),  
);
```

```
CREATE TABLE urunler  
(  
tedarikci_id char(10),  
urun_id char (10),  
CONSTRAINT urunler_fk FOREIGN KEY (tedarikci_id) REFERENCES tedarikciler  
(tedarikci_id)  
);
```

```
-- P6 "tedarikciler" isimli bir Tablo olusturun. Icinde "tedarikci_id",  
"tedarikci_isim", "iletisim_isim" field'leri olsun.  
--"tedarikci_id" ve "tedarikci_isim" fieldlerini birlestirerek Primary Key  
olusturun.  
--"urunler" isminde baska bir tablo olusturun. Icinde "tedarikci_id" ve "urun_id"  
fieldleri olsun.  
--"tedarikci_id" ve "urun_id" fieldlerini birlestirerek Foreign Key olusturun
```

```
CREATE TABLE tedarikciler01  
(  
tedarikci_id char(10),  
tedarikci_isim varchar2(50),  
iletisim_isim varchar2(50),  
CONSTRAINT tedarikciler01_pk PRIMARY KEY (tedarikci_id, tedarikci_isim)  
);
```

```
CREATE TABLE urunler01  
(  
tedarikci_id char(10),  
urun_id varchar2(10),  
CONSTRAINT urunler01_fk FOREIGN KEY (tedarikci_id, urun_id) REFERENCES  
tedarikciler01 (tedarikci_id, tedarikci_isim)  
);
```

```
-- P  
CREATE TABLE students  
(  
id number(9),  
isim varchar2(50),  
derece number(3),  
adres varchar2(100),  
last_modification date,  
CONSTRAINT id_pk PRIMARY KEY(id)  
);
```

(1- Tüm field lere data eklemek için;)

```
INSERT INTO students VALUES (123456789, 'Ali Can', 85, 'Paris, Louvre Museum', '13-  
Oct-2020');
```

(2- Bazı field lere data eklemek için;)

```
INSERT INTO students (id, isim) VALUES (456123789, 'Veli Han');
```



```
SELECT * FROM students;
```

SQL-4

14.10.2020

- Tablodaki Data Nasıl Update Edilir (UPDATE SET)?

-- P Bir "tedarikçiler" tablosu oluşturun. içinde id, isim ve iletisim_isim field'leri olsun. Id ve isim'i beraber Primary Key yapın.

```
CREATE TABLE tedarikciler
(
id number(10),
isim varchar2(50),
iletisim_isim varchar2(50),
CONSTRAINT tedarikci_pk PRIMARY KEY (id, isim)
);
-- Icine 3 kayıt ekleyin (1, 'ACB', 'Ali Can'), (2, 'RDB', 'Veli Gul'), (3, 'KMN', 'Ayse Gulmez').
INSERT INTO tedarikciler VALUES (1, 'ACB', 'Ali Can');
INSERT INTO tedarikciler VALUES (2, 'RDB', 'Veli Gul');
INSERT INTO tedarikciler VALUES (3, 'KMN', 'Ayse Gulmez');
-- id'si 1 olan tedarikcinin ismini 'KRM' ve iletisim_isim'ini 'Hasan Han' yapın
UPDATE tedarikciler
SET isim = 'KRM', iletisim_isim = 'Hasan Han'
WHERE id = 1;
-- Ismi RDB olan tedarikcinin iletisim isim'ini Kemal Yasa yapın
UPDATE tedarikciler
SET iletisim_isim = 'Kemal Yasa'
WHERE isim = 'RDB';
```

-- P11 a) Urunler tablosundan Ali Can'in aldığı urunun ismini, tedarikci tablosunda irtibat_isim Merve Temiz olan sirketin ismi ile değiştirin

--b) TV satın alan müşterinin ismini, Apple'in irtibat_isim'i ile değiştirin

```
CREATE TABLE tedarikci
(
id number(5) PRIMARY KEY,
isim varchar2(50),
irtibat_isim varchar2(50)
);
INSERT INTO tedarikci VALUES (100, 'IBM', 'Ali Can');
INSERT INTO tedarikci VALUES (101, 'APPLE', 'Merve Temiz');
INSERT INTO tedarikci VALUES (102, 'SAMSUNG', 'Kemal Can');
INSERT INTO tedarikci VALUES (103, 'LG', 'Ali Can');

CREATE TABLE urunler
(
tedarikci_id number(5),
urun_id number(11),
urun_isim varchar2(50),
musteri_isim varchar2(50),
CONSTRAINT urunler_fk FOREIGN KEY (tedarikci_id) REFERENCES tedarikci (id)
);
INSERT INTO urunler VALUES (100, 1001, 'Laptop', 'Suleyman');
INSERT INTO urunler VALUES (101, 1002, 'iPad', 'Fatma');
INSERT INTO urunler VALUES (102, 1003, 'TV', 'Ramazan');
INSERT INTO urunler VALUES (103, 1004, 'Phone', 'Ali Can');
```

```
a) UPDATE urunler
SET urun_isim = (SELECT isim
FROM tedarikci
WHERE irtibat_isim = 'Merve Temiz')
```

```

WHERE musteri_isim = 'Ali Can'
b) UPDATE urunler
SET musteri_isim = (SELECT irtibat_isim
                     FROM tedarikci
                     WHERE isim = 'APPLE')
WHERE urun_ismi 'TV';

-- P Ogrenciler isminde bir tablo olusturun, icinde id, isim, not_ortalamasi, adres
ve son_degistirme_tarihi fieldleri olsun

CREATE TABLE ogrenciler
(
id char(11),
isim varchar2(50),
not_ortalamasi number(3),
adres varchar2(50),
son_degistirme_tarihi date
);

--123456789, Ali Can', 80, 'Istanbul,bakirkoy', '14-Oct-2020'

INSERT INTO ogrenciler VALUES('123456789', 'Ali Can', 80, 'Istanbul, bakirkoy',
'14-Oct-2020');
INSERT INTO ogrenciler VALUES('123456788', 'Veli Han', 83, 'Ankara, Cankaya', '12-
Oct-2020');

SELECT * FROM ogrenciler;

```

• SQL-4 Slayt Sayfa-8 önemli bir soru

| | |
|--|---|
| <p>1) Ogrenciler tablosu olusturun. Icinde id,isim,veli_isim ve grade field'lari olsun. Id ve isim fieldlari birlikte Primary Key olsun.</p> <pre> CREATE TABLE ogrenciler (id char(3), isim varchar2(50), veli_isim varchar2(50), yazili_notu number(3), CONSTRAINT ogrenciler_pk PRIMARY KEY (id)); </pre> | <p>2) 3 kisiyi tabloya ekleyin. (123, 'Ali Can', 'Hasan',75), (124, 'Merve Gul', 'Ayse',85), (125, 'Kemal Yasa', 'Hasan',85).</p> <pre> INSERT INTO ogrenciler VALUES(123, 'Ali Can', 'Hasan',75); INSERT INTO ogrenciler VALUES(124, 'Merve Gul', 'Ayse',85); INSERT INTO ogrenciler VALUES(125, 'Kemal Yasa', 'Hasan',85); </pre> |
| <p>3) notlar tablosu olusturun. ogrenci_id,ders_adi,yazili_notu field'lari olsun, ogrenci_id field'i Foreign Key olsun</p> <pre> CREATE TABLE notlar (ogrenci_id char(3), ders_adi varchar2(30), yazili_notu number(3), CONSTRAINT notlar_fk FOREIGN KEY (ogrenci_id) REFERENCES ogrenciler (id)); </pre> | <p>4) notlar tablosuna 3 kayıt ekleyin ('123','kimya',75), ('124','fizik',65),('125','tarih',90)</p> <pre> INSERT INTO notlar VALUES ('123','kimya',75); INSERT INTO notlar VALUES ('124','fizik',65); INSERT INTO notlar VALUES ('125','tarih',90); </pre> |
| <p>5) Tum ogrencilerin yazili notlarini notlar tablosundaki ile update edin</p> <pre> UPDATE ogrenciler SET yazili_notu= (SELECT yazili_notu FROM notlar WHERE ogrenciler.id=notlar.ogrenci_id WHERE id>100; </pre> | |

SQL-5

15.10.2020

```

-- mart_satislar isminde bir tablo olusturun. Icinde urun_id, musteri_isim,
urun_isim, urun_fiyat fieldlari olsun
CREATE TABLE mart_satislar
(
urun_id char(5),
musteri_isim varchar2(30),
urun_isim varchar2(50),
urun_fiyat number (9)
);
INSERT INTO mart_satislar VALUES (10, 'Ali', 'Honda', 75000);
INSERT INTO mart_satislar VALUES (10, 'Ayse', 'Honda',80000);

```

```

INSERT INTO mart_satislar VALUES (20, 'Hasan', 'Toyota',90000);
INSERT INTO mart_satislar VALUES (30, 'Veli', 'Ford',100000);
INSERT INTO mart_satislar VALUES (20, 'Ali', 'Toyota',110000);
INSERT INTO mart_satislar VALUES (10, 'Veli', 'Honda',120000);
INSERT INTO mart_satislar VALUES (40, 'Ayse', 'Hyundai',130000);
INSERT INTO mart_satislar VALUES (20, 'Ali', 'Toyota',140000);

SELECT * FROM mart_satislar;

--musteri ismi Hasan olan satisinin urun_isim 'ini Honda yapiniz
UPDATE mart_satislar
SET urun_isim = 'Honda'
WHEREmusteri_isim = 'Hasan';

--urun_isim = 'Toyota' olanlari urun_id = '50' yapin
UPDATE mart_satislar
SET urun_id = '50'
WHERE urun_isim = 'Toyota';

-- urun_isim 'i 'Honda' olanlari urun_fiyat larini %10 artirin
UPDATE mart_satislar
SET urun_fiyat = urun_fiyat * 1.1
WHERE urun_isim = 'Honda';

--musteri_isim 'leri Ayse olanlara %10 indirim yapin
UPDATE mart_satislar
SET urun_fiyat = urun_fiyat * 0.9
WHEREmusteri_isim = 'Ayse';

```

Tablodan Data Nasıl Silinir (DELETE)?

```

CREATE TABLE ogrenciler
(
id char(3),
isim varchar2(50),
veli_isim varchar2(50),
yazili_notu number(3),
CONSTRAINT ogrenciler_pk PRIMARY KEY (id)
);

INSERT INTO ogrenciler VALUES (123, 'Ali Can', 'Hasan', 75);
INSERT INTO ogrenciler VALUES (124, 'Merve Gul', 'Ayse', 85);
INSERT INTO ogrenciler VALUES (125, 'Kemal Yasa', 'Hasan', 85);

SELECT * FROM ogrenciler;

DELETE ogrenciler;

```

- **DELETE** tablo_ismi yazarsak tablodaki tüm rekord'ları (dataları) siler.
- Tüm kayıtlar silindikten sonra bos bir tablo kalır. Eğer tabloyu görmek isterseniz **“no data found”** yazar, tabloyu göstermez.
- **DELETE** komutu tabloyu silmez, sadece kayıtları siler.

```

--isim 'i 'Ali Can' olan kaydı silin
DELETE ogrenciler
WHERE isim = 'Ali Can';

-- yazili_notu 85 olanlari siliniz
DELETE ogrenciler
WHERE yazili_notu = 85;

--veli_isim Hasan veya Ayse olan kayitlari sil

```

```
DELETE ogrenciler
WHERE veli_isim = 'Hasan' OR veli_isim = 'Ayse';

--veli_isim Ayse veya yazili_notu = 75 olan kayitlari sil
DELETE ogrenciler
WHERE veli_isim = 'Ayse' OR yazili_notu = 75;

--veli_isim Hasan ve yazili_notu = 75 olan kayitlari sil
DELETE ogrenciler
WHERE veli_isim = 'Hasan' AND yazili_notu = 75;

--yazili_notu = 85 olmayan kayitlari sil (!= veya <>)
DELETE ogrenciler
WHERE yazili_notu <> 85;
```

Tablodan Data Nasıl Silinir (TRUNCATE)?

```
TRUNCATE TABLE ogrenciler;
```

- Truncate ve Delete komutlarının ikisi de bir tabloda bulunan kayıtları silmek için kullanılır.
- İki komutta sadece belirtilen tablodaki kayıtları siler.
- En belirgin farkı ise DELETE komutu ile belli bir aralığı silebilirken TRUNCATE komutu ile tablonun tamamı silinmektedir.
- “Truncate” kodu kullanılarak bir tablo silinirse dataların geri getirilme ihtimali olmaz.
- “Truncate” kodu geri getirilmesini (rolling back) istemeyeceğiniz tabloları silmek için kullanılır.
- DELETE FROM ile sildiğimiz kayıtları geri getirebiliriz ama TRUNCATE ile silinen kayıtlar geri getirilemez.

```
TRUNCATE TABLE ogrenciler;
DELETE ogrenciler; yada
DELETE ogrenciler
WHERE veli_isim;

SELECT * FROM ogrenciler;
```

Tablodan Data Nasıl Silinir (DROP)?

```
DROP TABLE ogrenciler;
```

- DROP TABLE tüm tabloyu siler ve RECYCLEBIN 'e gönderir.
- DROP TABLE ile silinen tablolar FLASHBACK TABLE ile geri getirilebilir.

```
FLASHBACK TABLE ogrenciler TO BEFORE DROP;
```

- Bir tabloyu geri getirilmemek üzere silmek istiyorsak iki yöntemle yapabiliriz; **(sf 8)**

```
-- 1) Önce DROP TABLE ile silip PURGE TABLE ile RECYCLEBIN 'den de silinebilir
```

```
DROP TABLE ogrenciler;
PURGE TABLE ogrenciler;
```

```
-- 2) DROP TABLE ve PURGE TABLE komutlarını beraber kullanıp geri getirilmeyecek şekilde silebiliriz
```

```
DROP TABLE mart_satislar PURGE;
FLASHBACK TABLE ogrenciler TO BEFORE DROP;
```

- DROP TABLE ile silinmeyen bir tablo direkt PURGE TABLE yapılamaz.

```
-- direkt PURGE TABLE ile silmeye çalışırsanız hata alırsınız.
PURGE TABLE ogrenciler;
```

UYARI: Purge kullandığımızda Tabloyu ve dataları geri getirmek mümkün değildir.

Purge Kullanmanın Amacı: Hassas bilgileri silmek istediğinizde başka insanların o bilgiye ulaşamayacağından emin olursunuz.

```
SELECT * FROM mart_satislar;
```

```
INSERT INTO ogrenciler VALUES (123, 'Ali Can', 'Hasan', 75);  
INSERT INTO ogrenciler VALUES (124, 'Merve Gul', 'Ayse', 85);  
INSERT INTO ogrenciler VALUES (125, 'Kemal Yasa', 'Hasan', 85);
```

SELECT KOMUTU

- **SELECT** komutu bize var olan bir dosyadaki istediğimiz dataları getirir

-- **1) Tüm Dataları çağırma;**

```
SELECT *  
FROM ogrenciler;
```

-- **2) Sadece bir field**

-- ogrenciler tablosundaki tüm isim'leri yazdırın

```
SELECT isim  
FROM ogrenciler;
```

- Kayıtlar arasında filtreleme yapmak için **WHERE** komutu kullanılır.

--ogrenciler tablosundan yazili notu 85 olanların veli isimlerini getirin

```
SELECT veli_isim  
FROM ogrenciler  
WHERE yazili_notu = 85;
```

--**3) Tablodan birden fazla field'i çağırma**

--veli_isim 'i 'Hasan' olanların isim ve yazili_notu 'larını listeleyen bir sorgu yapınız


```
SELECT isim, yazili_notu  
FROM ogrenciler  
WHERE veli_isim = 'Hasan';
```

--**4) Bir kayıta ait birden fazla sütunu listeleme**

--id'si 124 olan öğrencinin isim ve yazili_notu

```
SELECT isim, yazili_notu  
FROM ogrenciler  
WHERE id = 124;
```

--**WHERE** komutu **AND**, **OR**, **<**, **=** ve **>** gibi mantıksal operatörler ile kullanılabilir.



| | |
|-----|-----------------------------------|
| = | ==> Equal to sign |
| > | ==> Greater than sign |
| < | ==> Less than sign |
| >= | ==> Greater than or equal to sign |
| <= | ==> Less than or equal to sign |
| < > | ==> Not Equal to sign |
| AND | ==> And operator |
| OR | ==> Or operator |

IN CONDITION

IN Condition birden fazla mantıksal ifade ile tanımlayabileceğimiz durumları (**Condition**) tek komutla yazabilme imkânı verir.

```
CREATE TABLE musteriler
(
urun_id number(10),
musteri_isim varchar2(50),
urun_isim varchar2(50)
);
```

```
INSERT INTO musteriler VALUES (10, 'Mark', 'Orange');
INSERT INTO musteriler VALUES (10, 'Mark', 'Orange');
INSERT INTO musteriler VALUES (20, 'John', 'Apple');
INSERT INTO musteriler VALUES (30, 'Amy', 'Palm');
INSERT INTO musteriler VALUES (20, 'Mark', 'Apple');
INSERT INTO musteriler VALUES (10, 'Adem', 'Orange');
INSERT INTO musteriler VALUES (40, 'John', 'Apricot');
INSERT INTO musteriler VALUES (20, 'Eddie', 'Apple');
```

```
SELECT * FROM musteriler;
```

--Orange veya Apple alan musteriler isimlerini yazdırın

```
SELECT musteriler_isim
FROM musteriler
WHERE urun_isim = 'Orange' OR urun_isim = 'Apple';
```

--Orange veya Apple veya Apricot alan musteriler isimlerini yazdırın

```
SELECT musteriler_isim
FROM musteriler
WHERE urun_isim = 'Orange' OR urun_isim = 'Apple' OR urun_isim = 'Apricot';
```

```
SELECT musteriler_isim
FROM musteriler
WHERE urun_isim IN ('Orange', 'Apple', 'Apricot');
```

--musteriler_isim 'Mark' 'John' olanların aldıkları urun_isim 'lerini yazdırın

```
SELECT urun_isim
FROM musteriler
WHERE musteriler_isim IN ('Mark', 'John');
```

-- urun_id si 20'den büyük ve urun_id si 40'dan küçük olan ürünleri alan

```
musteriler_isim 'leri
SELECT musteriler_isim
FROM musteriler
WHERE urun_id >20 AND urun_id<40;
```

-- urun_id si 20'ye eşit ve büyük veya urun_id si 40'a eşit ve 40'dan küçük olan ürünleri alan müşteri isimleri

```
SELECT musteriler_isim
FROM musteriler
WHERE urun_id >= 20 AND urun_id <= 40;
```

BETWEEN CONDITION

BETWEEN Condition iki mantıksal ifade ile tanımlayabileceğimiz durumları tek komutla yazabilme imkânı verir. **BETWEEN** komutunda yazdığımız iki sınır da aralığa dahildir (**INCLUSIVE**)

```
SELECT musteriler_isim
FROM musteriler
WHERE urun_id BETWEEN 20 AND 40;
```

NOT BETWEEN CONDITION

NOT BETWEEN Condition iki mantıksal ifade ile tanımlayabileceğimiz durumları tek komutla yazabilme imkânı verir. **NOT BETWEEN** komutunda yazdığımız 2 sınır da aralığa hariştir (**EXCLUSIVE**)

```
--urun_id si 20'den küçük veya 30'dan büyük olan urun_isim 'lerini
SELECT urun_isim
FROM musteriler
WHERE urun_id < 20 OR urun_id > 30;

SELECT urun_isim
FROM musteriler
WHERE urun_id NOT BETWEEN 20 AND 30;
```

"Truncate ve Delete komutlarının ikisi de bir tabloda bulunan kayıtları silmek için kullanılır. İki komutta sadece belirtilen tablodaki kayıtları siler. En belirgin farkı ise DELETE komutu ile belli bir aralığı silebilirken TRUNCATE komutu ile tablonun tamamı silinmektedir."

1. DELETE ile TRUNCATE arasındaki fark nedir?

- A) TRUNCATE tüm kayıtları siler, DELETE istersek tüm kayıtları, istersek belirli kayıtları siler
- B) DELETE ile sildiğimiz dataları ROLLBACK yapabiliriz, TRUNCATE ile silinenler geri getirilemez
- C) DELETE ile WHERE komutunu kullanabiliriz ama TRUNCATE ile kullanamayız

2. DELETE ile DROP arasındaki fark nedir?

DELETE kayıtları siler, DROP ise tabloları.

3. DROP ile DROP PURGE arasındaki fark nedir?

DROP ile sildiğimiz dosyalar RECYCLEBIN 'e gider. PURGE RECYCLEBIN 'deki dosyaları geri getirilmeyecek şekilde siler. DROP PURGE beraber kullanılırsa geri getirilmeyecek şekilde silinir.

SQL-6

20.10.2020

EXISTS Condition subquery'ler ile kullanılır. IN ifadesinin kullanımına benzer olarak, EXISTS ve NOT EXISTS ifadeleri de alt sorgudan getirilen değerlerin içerisinde bir değerın olması veya olmaması durumunda işlem yapılmasını sağlar.

Nazmi Mert GitHub ta butun kodları: <https://github.com/nzmrt/batch5Api.git>

```
--Personel isminde bir tablo olusturun. Icinde id, isim, sehir, maas ve sirket
field'lari olsun.
--Id'yi 2.yontemle PK yapın

CREATE TABLE personel
(
id number(9),
isim varchar2(50),
sehir varchar2(50),
maas number(20),
sirket varchar2(20),
CONSTRAINT personel_pk PRIMARY KEY (id)
);

INSERT INTO personel VALUES(123456789, 'Ali Yilmaz', 'Istanbul', 5500, 'Honda');
```

```

INSERT INTO personel VALUES(234567890, 'Veli Sahin', 'Istanbul', 4500, 'Toyota');
INSERT INTO personel VALUES(345678901, 'Mehmet Ozturk', 'Ankara', 3500, 'Honda');
INSERT INTO personel VALUES(456789012, 'Mehmet Ozturk', 'Izmir', 6000, 'Ford');
INSERT INTO personel VALUES(567890123, 'Mehmet Ozturk', 'Ankara', 7000, 'Tofas');
INSERT INTO personel VALUES(456715012, 'Veli Sahin', 'Ankara', 4500, 'Ford');
INSERT INTO personel VALUES(123456710, 'Hatice Sahin', 'Bursa', 4500, 'Honda');

SELECT * FROM personel;

--Personel_bilgi isminde bir tablo olusturun. Icinde id, tel ve cocuk_sayisi
field'leri olsun.
--Id'yi FK yapin ve personel tablosu ile relation kurun

CREATE TABLE personel_bilgi
(
id number(9),
tel char(10) UNIQUE ,
cocuk_sayisi number(2),
CONSTRAINT personel_bilgi_fk FOREIGN KEY (id) REFERENCES personel (id)
);

INSERT INTO personel_bilgi VALUES(123456789, '5302345678', 5);
INSERT INTO personel_bilgi VALUES(234567890, '5422345678', 4);
INSERT INTO personel_bilgi VALUES(345678901, '5354561245', 3);
INSERT INTO personel_bilgi VALUES(456789012, '5411452659', 3);
INSERT INTO personel_bilgi VALUES(567890123, '5551253698', 2);
INSERT INTO personel_bilgi VALUES(456789012, '5524578574', 2);
INSERT INTO personel_bilgi VALUES(123456710, '5537488585', 1);

SELECT * FROM personel_bilgi;

--SORU 1) Personel_bilgi tablosundan 5 cocugu olan kisinin cocuk sayisini 2 yapin
UPDATE personel_bilgi
SET cocuk_sayisi = 2
WHERE cocuk_sayisi = 5;

--SORU 2) Personel tablosundan ucreti 4500 veya 5000 olanlari maaslarini %10
artirin
UPDATE personel
SET maas = maas *1.1
WHERE maas IN (4500, 5000);

--SORU 3) Personel tablosundan maasi 4950 olanlari silin
DELETE personel
WHERE maas = 4950;

Parent-Child relation'i oluřturulan tablolarda child datalar silinmeden parent datalar
silinemez, SQL ORA-02292 hatası verir.

--SORU 4) cocuk sayisi 3 veya 4 olanlari siliniz.
DELETE personel_bilgi
WHERE cocuk_sayisi IN (3,4);

--SORU 5) Honda da calisip maasi 4500 ve üzeri olanlari silin.
DELETE personel
WHERE sirket = 'HONDA' AND maas = 3500;

--SORU 6) personel_bilgi den datalari geri gelmeyecek sekilde silin.
TRUNCATE TABLE personel_bilgi;

--SORU 7) personel tablosundan maasi 4000 ile 5000 arasinda olanlari silin.
DELETE personel
WHERE maas BETWEEN 4000 AND 5000;

```



```
--SORU 8) Personel tablosundan maasi 5000 ile 6000 arasında olmayanları silin.  
DELETE personel  
WHERE maas NOT BETWEEN 5000 AND 6000;
```

```
--SORU 9) Personel tablosunu geri getirilemeyecek şekilde silin.
```

- Her ne kadar personel_bilgi tablosundan silmiş olsak da tablo duruyor
- Child tablo boş olarak dururken parent tablodan kayıtları rahatlıkla silebilirsiniz
- Ancak child tablo silinmeden parent tablo silinemez.

```
DROP TABLE personel PURGE; hata verir.
```

- Önce personel_bilgi tablosu silinmeli
DROP TABLE personel_bilgi;
- Sonra personel tablosu silinmeli
DROP TABLE personel PURGE;

SUBQUERIES

SUBQUERY başka bir SORGU (**query**)'nün içinde çalışan SORGU 'dur.

1. **WHERE** den sonra kullanılabilir.

```
--personel tablosu  
CREATE TABLE personel  
(  
  id number(9),  
  isim varchar2(50),  
  sehir varchar2(50),  
  maas number(20),  
  sirket varchar2(20)  
);  
  
INSERT INTO personel VALUES(123456789, 'Ali Seker', 'Istanbul', 2500, 'Honda');  
INSERT INTO personel VALUES(234567890, 'Ayse Gul', 'Istanbul', 1500, 'Toyota');  
INSERT INTO personel VALUES(345678901, 'Veli Yilmaz', 'Ankara', 3000, 'Honda');  
INSERT INTO personel VALUES(456789012, 'Veli Yilmaz', 'Izmir', 1000, 'Ford');  
INSERT INTO personel VALUES(567890123, 'Veli Yilmaz', 'Ankara', 7000, 'Hyundai');  
INSERT INTO personel VALUES(456789012, 'Ayse Gul', 'Ankara', 1500, 'Ford');  
INSERT INTO personel VALUES(123456710, 'Fatma Yasa', 'Bursa', 2500, 'Honda');  
  
SELECT * FROM personel;  
  
--sirketler tablosu  
CREATE TABLE sirketler  
(  
  sirket_id number(9),  
  sirket varchar2(20),  
  personel_sayisi number(20)  
);  
  
INSERT INTO sirketler VALUES(100, 'Honda', 12000);  
INSERT INTO sirketler VALUES(101, 'Ford', 18000);  
INSERT INTO sirketler VALUES(102, 'Hyundai', 10000);  
INSERT INTO sirketler VALUES(103, 'Toyota', 21000);  
  
SELECT * FROM sirketler;
```

- SUBQUERY SELECT ve WHERE ile kullanılabilir

--WHERE soruları

--**Soru 1)** Personel sayısı 15.000'den çok olan şirketlerin isimlerini ve bu şirkette çalışan personelin isimlerini listeleyin

```
SELECT şirket, isim
FROM personel
WHERE şirket IN ('Ford', 'Toyota');
```

--Personel sayısı 15.000'den çok olan şirketler

```
SELECT şirket
FROM şirketler
WHERE personel_sayısı > 15000;
```

--iki QUERY'i iç içe yazıyoruz

```
SELECT şirket, isim
FROM personel
WHERE şirket IN (SELECT şirket
                  FROM şirketler
                  WHERE personel_sayısı > 15000);
```

--**Soru 2)** Şirket_id'si 101'den büyük olan şirketlerin maaşlarını ve şehirlerini listeleyiniz

```
SELECT maaş, şehir
FROM personel
WHERE şirket IN (SELECT şirket
                  FROM şirketler
                  WHERE şirket_id > 101);
```

--**Soru 3)** 'Ankara' daki şirketlerin (Honda, Hyundai, Ford) şirket_id ve personel_sayısı'nı listeleyiniz

```
SELECT şirket_id, personel_sayısı
FROM şirketler
WHERE şirket (ortak) IN (SELECT şirket
                          FROM personel
                          WHERE şehir='Ankara');
```

SELECT ile SUBQUERY kullanımı

- Yazdığımız QUERY'lerde SELECT satırında field isimleri kullanıyoruz.
- Dolayısıyla eğer SELECT satırında bir SUBQUERY yazarsak sonucunun tek bir field ismi olması gerekir.
- Ancak SELECT CLAUSE da kullanılan SUBQUERY sadece 1 değer dönmelidir.
- Dolayısıyla SELECT satırında SUBQUERY yazarsak SUM, COUNT, MIN, MAX ve AVG gibi fonksiyonlar kullanılır.
- Bu fonksiyonlara AGGREGATE FUNCTION (hesaplama fonksiyonları) denir.

| Fonksiyon | Kullanımı |
|-----------|---------------------|
| SUM | Toplam |
| AVG | Ortalama |
| MAX | En büyük değer |
| MIN | En küçük değer |
| COUNT | Toplam Kayıt Sayısı |

Aggregate Function (Hesaplama Fonksiyonları)

2. **SELECT'** den sonra kullanılabilir.

--SORU 1- Her sirketin ismini, personel_sayisi ni ve personelin ortalama maasini listeleyen bir QUERY yazin.

```
SELECT sirket, personel_sayisi, (  
    SELECT AVG (maas)  
    FROM personel  
    WHERE sirketler.sirket = personel.sirket  
    ) ortalama_maas  
FROM sirketler;
```

--SORU 2- Her sirketin ismini ve personelin aldigi max. maasi listeleyen bir QUERY yazin.

```
SELECT sirket, (  
    SELECT MAX (maas)  
    FROM personel  
    WHERE sirketler.sirket = personel.sirket  
    ) max_maas  
FROM sirketler;
```

--SORU 3- Her sirketin id'sini, ismini ve toplam kac sehirde bulunduğunu listeleyen bir QUERY yaziniz.

```
SELECT sirket_id, sirket, (  
    SELECT COUNT (sehir)  
    FROM personel  
    WHERE sirketler.sirket = personel.sirket  
    ) bulunduгу_sehir_sayisi  
FROM sirketler;
```

--SORU 3,5- id si 101'den buyuk olan sirketlerin id'sini, ismini ve toplam kac sehirde bulunduğunu listeleyen bir QUERY yaziniz.

```
SELECT sirket_id, sirket, (  
    SELECT COUNT (sehir)  
    FROM personel  
    WHERE sirketler.sirket = personel.sirket  
    ) bulunduгу_sehir_sayisi  
FROM sirketler  
WHERE sirket_id>101;
```

SQL-7

21.10.2020

--SORU 4-