

## SQL İÇİNDEKİLER TABLOSU

<b>DAY 1</b>	<b>3</b>
PRIMARY KEY	4
FOREIGN KEY	4
<b>DAY 2</b>	<b>4</b>
VARIABLE'LAR : STRING DATA TYPES	4
NUMERIC DATA TYPES	5
DATE DATA TYPES	5
BLOB DATA TYPES	5
TABLO OLUŞTURMA	5
VAROLAN BİR TABLODAN YENİ TABLO ÜRETMEK	6
<b>DAY 3</b>	<b>6</b>
PRIMARY KEY OLUŞTURMANIN BİRİNCİ YOLU	6
PRIMARY KEY OLUŞTURMANIN İKİNCİ YOLU	6
BİRDEN FAZLA SUTUN KULLANARAK PRIMARY KEY OLUŞTURMA	7
FOREIGN KEY OLUŞTURMAK	7
TABLOYA VERİ GİRİŞİ YAPMA	8
INSERT INTO: İÇİNE YERLEŞTİR	8
BİR TABLONUN BELLİ FIELD'LARINA DATA EKLEMEK	8
TÜM TABLOYU KONSOLA YAZDIRMA	8
INSERT INTO KULLANILIRKEN DIKKAT EDİLMESİ GEREKENLER	8
BİR VERİYİ DEĞİŞTİRME UPDATE ETME	9
<b>DAY 4</b>	<b>9</b>
TABLE'DAN İSTENEN BİR SATIRI SİLMEK	9
TABLODAKİ TÜM SATIRLARI SİLME	9
BİRDEN FAZLA SATIR SİLME	9
AND KULLANARAK SİLME	9
OR KULLANARAK SİLME	9
TRUNCATE: TABLODAKİ TÜM SATIRLARI SİLER	9
DROP İLE TABLE SİLME	9
PURGE KULLANARAK TABLE SİLME	10
SELECT KULLANIMI	10
IN	10
SUM, COUNT, AVG, MIN, MAX	12
<b>DAY 5</b>	<b>12</b>
EXISTS	12
BETWEEN	13
NOT BETWEEN, BETWEEN'İN TERSİ OLARAK ÇALIŞIR.	13
IS NULL □ BOŞ ALANLARI SEÇ	14
ORDER BY □ SIRALA	14
DESC □ BÜYÜKTEN KÜÇÜĞE SIRALAMA	14
ASC □ KÜÇÜKTEN BÜYÜĞE SIRALAMA	15
AS □ RAPOR İÇİN SÜTUN İSİMLERİNİ DEĞİŞTİRME	15
GROUP BY	15
HAVING	16
UNION	17
<b>DAY 6</b>	<b>18</b>
UNION ALL	19
INTERSECT	19

MINUS	21
JOIN	22
LEFT JOIN	23
RIGHT JOIN	24
FULL JOIN	24

## DAY 7 24

WILDCARDS	24
% : BİR VEYA BİRDEN FAZLA KARAKTER GÖSTERİR.	25
LIKE GIBİ DEMEK J% : J İLE BAŞLASIN DEVAMI NE OLURSA OLSUN DEMEKTİR.	25
_ : SADECE BİR KARAKTERİ TEMSİL EDER, NE OLDUĞU ÖNEMLİ DEĞİL.	25
[ ] : SEMBOL İÇİNE KOYULAN HARFLERDEN HERHANGİ BİRİSİ OLABİLİR MANASINDA	26
^ : İLE BAŞLAYAN MANASINDA	27
(*) : HERŞEY MANASINDA	27

## DAY 8 27

PIVOT: SATIRLARI SÜTUNA DÖNÜŞTÜRMEK İÇİN KULLANILIR. ÖZET TABLO(EXCEL)	28
DISTINCT: SÜTÜN İÇERİSİNDE GEÇEN DATALARI TEKRARSIZ OLARAK GÖSTEREN KOD.	29
ALTER TABLE: TABLOLAR ÜZERİNDE DEĞİŞİKLİK YAPMA	30
ADD: TABLOYA YENİ SÜTÜN EKLEME	30
DROP COLUMN: TABLODAN SÜTÜN SİLME	32
RENAME COLUMN: SÜTÜN İSMİNİ DEĞİŞTİRME	32
RENAME TO: TABLO İSMİNİ DEĞİŞTİRME	33
MODIFY: TABLO SÜTUNLARININ YAPISINI DEĞİŞTİRME	33
SQL INTERVIEW QUESTIONS	34
MOD METODU: ÇİFT VEYA TEK SAYILARI FİLTRELEME.	36
ROWNUM: TABLODA İSTENİLEN SAYIDA SATIR SAYISINI GÖSTERME	37

## DAY 9 39

SELF JOIN	41
UPPER: BİR SÜTUNDAKİ DATALARIN BÜYÜK HARFLE YAZILMASINI SAĞLAR.	42
LOWER: BİR SÜTUNDAKİ DATALARIN KÜÇÜK HARFLE YAZILMASINI SAĞLAR.	42
INITCAP: BİR SÜTUNDAKİ DATALARIN İLK HARFINİ BÜYÜK YAPAR DİĞERLERİNİ KÜÇÜK.	42

## DAY 1

- API farklı applicationlar arasında haberleşmeyi sağlar.
- API'nın haber yollayıp cevap alma kabiliyeti var.
- DBMS data base management system
- Girişi düzenleyen, rapor almayı sağlayan, yeni tablolar oluşturmaya, silmeye, diğer applicationlar ile irtibat kurmamızı sağlayan bir sistemdir.
- Create: oluşturma
- Read: okuma
- Update: değiştirme, güncelleme
- Delete: silme
- Hepsi CRUD
- Database'den rapor alma: Her bölümden en çok satılan ürün hangisi gibi...
- SQL'de her şey bir tabloda saklanır. SQL içerisinde tablosuz data yoktur.
- Row: satır Column: sütun normalde
- Record: satır Field: Sütun SQL'de bu terimler kullanılıyor
- Header: başlık
- Relational Databases: birbiriyle ilişkili tablolar veya SQL Database aynı manaya gelir. Structured Query Language: yapılandırılmış sorgulama dili.
- Database hızlı çalışsın diye farklı tablolar oluşturulup bunlar birbiriyle ilişkilendirilir.
- Birbiriyle ilişkili tablolar yapısına Schema (şema) denir.
- Microsoft SQL Server : user interface güzel, büyük datalarda çok başarılı ama pahalı.
- MySQL : ücretsiz, kullanan çok, karşılaşılan problemlerle ilgili internette çok çözüm yolları mevcut, çok aşırı iş yüklemesinde başarısız.
- PostgreSQL: piyasaya yeni çıktı. Diğerlerinde olmayan farklı özellikler mevcut. Yüklemesini yapma, çalışır hale getirme vakit alıyor. Yeni başlayanları zorlayabilir. Bunun haricinde başarılı bir program.
- PL / SQL : Güvenlik ön planda, oracle ait bir program. Object – Oriented Programming ile uyumlu.
- Aralarında kullanım olarak çok küçük fark var. Biri öğrenildiğinde diğerleride öğrenilmiş olur.
- SQL olmayan Databases: Non Relational Databases : birbiriyle ilişkili olmayan tablolar. Tablo kullanılmadan yapılan veri oluşturma NoSQL denir.

## PRIMARY KEY

- **Primary Key'i** kullanarak uniQ(ikincisi olmayan) datalara ulaşırsınız. Tablolarda buna ihtiyaç duyulur. Tablo oluştururken bir data girerek bir şeylere ulaşmak istiyorsak Primary Key oluşturmamız. Primary Key tek sütunda olabilir, iki sütun veya üç sütunluda olabilir ama bunlar beraber tek bir Primary Key olur. Ad, soyad, baba adı gibi... Primary key Null olamaz. Bir tane Primary key birden fazla sütun içerebilir. Primary Key yapılacak sütun her tip datayı içerebilir. E-Mail adresi iyi bir Primary Key'dir. Gerçek bir Datayı email gibi Primary Key olarak kullanırsanız "Natural Key" denir. Gerçek olmayan Primary Key'lere "Surrogate Key" denir. İki sütun birleştirilerek Primary Key olarak tanımlanmışsa buna "Composite Key" denir.

## FOREIGN KEY

- **Foreign Key** => Primary Key Başka bir Tablonun Primary Key'i ile irtibat kurulan sütuna Foreign Key deniyor. Primary Key'e hangi sütun bağlanıyorsa ona Foreign Key deniyor. Foreign Key de tekrarlı değerler olabilir. Foreign Key Null değer alabilir. Bir tabloda bir tane Primary Key olabilir ama birden fazla Foreign Key olabilir. Foreign Key Tablolar arasında bağlantıyı sağlar. Foreign Key'in bulunduğu Tabloya Parent derler, bağlanan Tabloya Child Table denir.
- Database validation: veri doğrulama demektir. Tester bu işi yapar.

## DAY 2

- SQL kullanarak Database oluşturup yönetebiliriz, design'ini değiştirebilir, yeni tablolar oluşturabiliriz, silebiliriz, security ayarlarını yapabilir ve kullanıcı yetkilerini ayarlama.
- SQL içerisinde 4 farklı dil var. 1- Data kontrol language 2-Data Definition Language data oluşturma dizayn etme 3- Data Definition Language data değiştirme dili 4- Data Query Language veriyi sorgulama
- dili Oluşturmak ve yönetmek, oluşturmak ve dizayn etmek, data create yapma, security de kullanılıyor.

## VARIABLE'LAR : STRING DATA TYPES

- Char(size): çoklu karakteri kabul ediyor. Uzunluğu belli olan verilerde kullanılıyor. TC Kimlik 10 11 hanedir. Uzunluğu bellidir, sabittir, bunun için char kullanılır. Uzunluğu sabit olan verilerde kullanılır.
- Varchar2(size): çoklu karakteri kabul ediyor. Uzunluğunu bilmediğimiz, değişen stringlerde kullanılır. Bir max size belirlenir riske girilmeden bir rakam atanır. Yoksa geri kalan kısmı almaz.
- Nchar(size): n: number demek, ascii kodlarını depolamış oluruz. Uzunluğu belli olan kodların ascii kodlarını depolar.
- Nvarchar2(size): uzunluğu belli olmayan kodların ascii kodlarını depolar. Var: varie değişen demek.
- Veri uzunluğu sabit değişmiyorsa char(size), uzunluk sabit değil bilmiyorsak varchar2(size) kullanıyoruz.

## NUMERIC DATA TYPES

- `number(p, s)` p: sayıda kullanılan rakamların miktarını gösterir. S: ondalık kısımdaki sayıların miktarını gösterir.
- `number(5,2)` 123,45 demektir. `number(4)` 5678 olarak kabul eder ama 12,53 yazarsak ondalık kısmı belirtilmediği için onu 0 kabul eder bu sayıyı hafızaya 12 olarak alır. `number(4)` ile `number(4,0)` aynı anlama gelir.
- `Number(5,1)` 6789,12 böyle bir yazım mümkün değildir, bunu kabul etmez error verir.
- P sayısı 38 den büyük olamaz. S yerine negatif değerler kullanılabilir.
- `Number(5, -2)` = 63781,38 yuvarlayıp 63800 olur. 2 rakam geri git yuvarlama yap demek.
- `Number(4,-1)` = 6537,2 yuvarlayıp 6540

## DATE DATA TYPES

- 13-Apr-20 standart format budur istenirse format değiştirilebilir.

## BLOB DATA TYPES

- Resim Video gibi data typleri için BLOB denilen bir data type'i vardır.
- Binary Large objects BLOB
- SQL açıp SQL Worksheet sayfasına gittik
- CREATE TABLE students yazdık. Tablo oluştur ismin students olsun
- Birden fazla kelime varsa `update_date` şeklinde yazılıyor.
- Bir tablo aşağıdaki şekilde oluşturuluyor

## TABLO OLUŞTURMA

- CREATE TABLE students
- (  
• id char(11),  
• name varchar2(50),  
• grade number(3),  
• address varchar2(80),  
• update\_date  
• );
- Bunu seçip sağ üstte run yaptık, tablo oluşturdu.
- Schema'ya giderek oluşturduğumuz tablomuzun özelliklerine bakabiliriz.
- Tempory: geçici
- Nested: başka bir tablo içindeki bir tablo mu
- VALİD: geçerli
- Başlık isimlerini küçük yazıyoruz, kendisi otomatik olarak bunları büyük harf olarak yazar.
- Nullable: boş bırakılabilir mi. Hiçbir sütunu primary key olarak tanımlamadığımız için boş bırakılabilir mi sütun izahına yes yazmış.

### Varolan bir tablodan yeni tablo üretmek

- -- mesela sadece isim ve notlar olsun
- 
- CREATE TABLE student\_grade
- AS -- gibi demek
- SELECT id, grade
- FROM students;
- id char(11) NOT NULL, -- bu sütun boş kalmaması istiyorsak NOT NULL yazıyoruz.
- Constraints: sınırlama demektir.
- name varchar2(50) UNIQUE, bir sütunun tekrarlı olmamasını istiyorsak UNIQUE yazıyoruz.

## DAY 3

- Bir sütünü primary key yapabilmek için sonuna PRIMARY KEY yazıyoruz
- Constraint: sınırlandırma demektir. NOT NULL, UNIQUE, PRIMARY KEY (boş olmaz, tekrarlı olamaz) bunların hepsi Constraint'tir. Bir sütünü PRIMARY KEY yaptığımız zaman Constraint'te ona SYS\_C00055335 bir rakam atar.
- Status:ENABLED aktif demek
- Nullable: boş olabilir
- Uniqueness: UNIQUE benzeri yok

### Primary Key oluşturmanın birinci yolu

```
CREATE TABLE students3
```

```
(  
id char(11) PRIMARY KEY, -- bir sütünü primary key yapmak istiyorsak bu şekilde yazıyoruz.  
name varchar2(50),  
grade number(3),  
address varchar2(80),  
update_date date  
);
```

### Primary Key oluşturmanın ikinci yolu

```
CREATE TABLE students4
```

```
(  
id char(11),  
name varchar2(50),  
grade number(3),  
address varchar2(80),  
update_date date,  
--CONSTRAINTS İSİM PRIMARY KEY(SUTUN İSMİ) bazen  
CONSTRAINTS id_pk PRIMARY KEY(id)  
);
```

Bazen primary key ile başka tabloyu birleştirmemiz gerekecek bu durumda id\_pk ismi gerekli olacak.

### Birden fazla sütun kullanarak Primary Key oluşturma

```
CREATE TABLE students5
(
  id char(11),
  name varchar2(50),
  grade number(3),
  address varchar2(80),
  update_date date,
  --CONSTRAINTS İSİM PRIMARY KEY(SUTUN İSMİ) bazen
  CONSTRAINTS id_name_pk PRIMARY KEY(id, name)
);
```

- Composite: birleştirmek demek

### Foreign Key oluşturmak

```
CREATE TABLE students6
(
  id char(11),
  name varchar2(50),
  grade number(3),
  address varchar2(80),
  update_date date,
  CONSTRAINTS id_fk FOREIGN KEY(id) REFERENCES students4(id) -- students6 tablosundan
  students4 tablosundaki primary keye ulaşmak istiyorum.
  -- REFERENCES dan sonra tablo ismi ve primary key olan sütun ismini yazıyoruz.
  -- bu sütun primary key olmazsa incompatible(uygun değil) diye hata alınır.
  -- FOREIGN KEY de index oluşturmaz.
);
```

### TABLOYA VERİ GİRİŞİ YAPMA

```
CREATE TABLE students
(
students_id char(7),
students_name varchar2(50),
students_grades number(3),
students_cinsiyet char(5)
);
```

### INSERT INTO: içine yerleştir

```
INSERT INTO students VALUES('2020301', 'Ali Can', 87, 'Erkek');
```

### Bir tablonun belli field'larına data eklemek

```
INSERT INTO students(students_id,students_name,students_cinsiyet) VALUES('2020404',
'Emine saz', 'Kız');
INSERT INTO students(students_name,students_cinsiyet) VALUES('Asım Genç', 'Erkek');
```

### TÜM TABLOYU KONSOLA YAZDIRMA

-- altta konsolda yazdırmak için aşağıdaki kodları yazıyoruz.

```
SELECT * -- hepsini seç students tablosundan
FROM students;
```

```
CREATE TABLE students_family1
(
students_id char(7) UNIQUE,
mother_name varchar2(50) NOT NULL,
father_name varchar2(50),
address varchar2(80)
);
```

### INSERT INTO kullanılırken dikkat edilmesi gerekenler

- 1) Constraints'lerle çelişen data girişi yapmayın
- 2) Data Type'larına uygun veriler girin
- 3) Sütun sıraları ile data sıraları uyumlu olmalıdır.

-- aşağıda ilk ikisini çalıştırır , üçüncüsünde error verir. mother\_name boş bırakılamaz, boş bırakılarak giriş yapmak isteniyor hata verir.

```
INSERT INTO students_family1 VALUES('2020301', 'Halime', 'Recep', 'Istanbul Bakirkoy');
INSERT INTO students_family1 VALUES('2020302', 'Melahat', 'Kerem', 'Istanbul Basaksehir');
INSERT INTO students_family1(students_id,father_name,address) VALUES('2020304',
'Ramazan', 'Istanbul Bagcilar');
```

### bir veriyi değiştirme UPDATE etme



```
UPDATE students_family1
SET address = 'Ankara Cankaya'
WHERE students_id = '2020301';
```

## DAY 4

Table'dan istenen bir satırı silmek

```
DELETE FROM products -- DELETE bir satır siler
WHERE supplier_id = 103;
```

### TABLODAKİ TÜM SATIRLARI SİLME

--DELETE FROM products; kodu table'daki tüm dataları siler ama table'in yapısı durur.  
DELETE FROM products; -- bir satır belirtilmezse tüm satırları siler.

### BİRDEN FAZLA SATIR SİLME

```
DELETE FROM products
WHERE supplier_id < 102;
```

### AND KULLANARAK SİLME

--AND kullanırsanız SQL iki şartda sağlanıyorsa bu satırı bulur ve siler  
--iki şart birden sağlanmazsa SQL silme işlemini yapamaz, error da vermez.  
DELETE FROM products  
WHERE product\_name = 'Phone' AND costumer\_name = 'Ramazan';

### OR KULLANARAK SİLME

--altta iki satırda silecektir  
DELETE FROM products  
WHERE product\_name = 'Phone' OR costumer\_name = 'Ramazan';

### TRUNCATE: Tablodaki tüm satırları siler

```
TRUNCATE TABLE products;
```

--TRUNCATE ile DELETE FROM arasındaki fark DELETE kullanırsanız data tekrar kurtarabilirsiniz, TRUNCATE kullanırsanız tekrar kullanamazsınız.  
--çöp kutusuna atma veya del+enter gibi bir şey  
--özel bilgilerin geri getirilmemesi isteniyorsa TRUNCATE kullanılır.

### DROP İLE TABLE SİLME

--DROP: Bir Table'ı yapısı ve içindeki dataları ile birlikte silmek için  
--kullanılır. Artık tablo yok demektir  
--DROP: İngilizcede düşür demektir.  
--DROP yazarak imha ettiğimiz TABLE bazı kodlar kullanılarak geri çağırılabilir.

```
DROP TABLE products;
```

## PURGE KULLANARAK TABLE SİLME

--DROP kullanarak TABLE imha ettiğinizde geri çağırılmasını engellemek istiyorsanız

DROP TABLE products PURGE; --PURGE: Tenkil manasında

## SELECT KULLANIMI

-- 1) Table'daki tüm dataları(\*) görmek için kod yazınız.

```
SELECT *  
FROM products
```

-- 2) Table'daki belli bir satırı görmek için kod yazınız.

```
SELECT *  
FROM products  
WHERE product_id = 1001;
```

-- 3) TABLE'daki bazı satırları görmek için kod yazınız.

```
SELECT *  
FROM products  
WHERE supplier_id = 101 OR supplier_id = 103;
```

## IN

-- birden fazla satır görmek için OR dan daha kısa bir çözüm IN kullanmaktır.

```
SELECT *  
FROM products  
WHERE supplier_id IN (101,103) -- bu küme içinde olanları göster.
```

```
SELECT *  
FROM products  
WHERE supplier_id < 103; -- 103 ten küçük olanları göster.
```

-- 4) Bir TABLE'daki belli bir sütunu görmek için kod yazınız.

```
SELECT customer_name  
FROM products;
```

-- 5) Bir TABLE'da birden fazla sütunu görmek için kod yazınız.

```
SELECT customer_name, product_name  
FROM products;
```

-- 6) Bir TABLE'dan birden fazla sütun ve bir satır görmek için kod yazınız.

```
SELECT costumer_name, product_name  
FROM products  
WHERE costumer_name = 'Suleyman';
```

-- Tek sütun iki satır

```
SELECT costumer_name  
FROM products  
WHERE costumer_name IN ('Suleyman', 'Ramazan');
```

-- SELECT'ten sonra SELECT kullanırsanız içerdeki koda SUBQUERY denir.

-- işçi sayısı 15000 den fazla olan company'lerin company isimlerini ve işçi

-- isimlerini gösteren kodu yazınız.

-- SELECT'ten sonra SELECT kullanırsanız içerdeki SUBQUERY denir.

```
SELECT company, name  
FROM employees  
WHERE company IN ( SELECT company  
                    FROM companies  
                    WHERE number_of_employees >15000);
```

-- company id si 102 den küçük olan companylerin salary'lerini ve state lerini

-- gösteren kodu yazınız

```
SELECT salary, state  
FROM employees  
WHERE company IN ( SELECT company  
                    FROM companies  
                    WHERE company_id < 102);
```

-- Florida'daki company'lerin company id lerini ve number of employees i gösteren kodu yazınız

```
SELECT company_id, number_of_employees  
FROM companies  
WHERE company IN ( SELECT company  
                    FROM employees  
                    WHERE state = 'Florida');
```

-- NOT: SUBQUERY'ler WHERE den sonra kullanılabildiği gibi SELECT'den sonra da kullanılabirler.

-- Her şirketteki işçi sayısını ve ortalama işçi ücretlerini gösteren kodu yazınız.

```
SELECT company,number_of_employees, (SELECT AVG(salary) -- AVG ortalamasını alan bir  
method
```

```
FROM employees
```

```
WHERE companies.company=employees.company) average_salary -- .
```

içindeki manasında

```
FROM companies;
```

-- her şirketteki toplam state sayısını gösteren kodu yazınız( COUNT(state))

```
SELECT company, (SELECT COUNT(state)
```

```
FROM employees
```

```
WHERE companies.company=employees.company) count_state
```

```
FROM companies;
```

-- Her şirketin company id'sini ve işçilerine yaptığı toplam ödemeyi gösteren kodu yazınız

```
SELECT company,company_id, (SELECT SUM(salary)
```

```
FROM employees
```

```
WHERE companies.company=employees.company) sum_salary
```

```
FROM companies;
```

SUM, COUNT, AVG, MIN, MAX

gibi fonksiyonlar AGGREGATE fonksiyonlar olarak adlandırılır.

- SUM: sütun toplamını alır. SELECT SUM(salary)
- COUNT: sütunda geçen adet sayısını yazar. SELECT COUNT(state)
- AVG: sütunun ortalamasını alır. SELECT AVG(salary)
- MIN: sütun içerisindeki min değeri alır. SELECT MIN(salary)
- MAX: sütun içerisindeki max değeri alır. SELECT MAX(salary)

## DAY 5

### EXISTS

-- EXISTS komutu SUBQUERY'lerle beraber kullanılır.

-- EXISTS: sütunda var mı yokmu diye bakar

-- IN komutu OR komutunun kısa yazılmış halidir. IN komutu tek başına SUBQUERY'lerle

-- Beraber kullanılmıyor. SUBQUERY kullanacaksanız EXISTS kullanmamız gerekir.

-- Product\_id leri aynı olan müşterilerin isimlerini gösteriniz.

```
SELECT customer_name
FROM customers_products
WHERE EXISTS (SELECT product_id
              FROM customers_likes
              WHERE customers_products.product_id = customers_likes.product_id
              );
```

## BETWEEN

-- BETWEEN(arasında ==> 10 ile 20 arasında : BETWEEN 10 AND 20) komutunun daha anlaşılır halidir.

-- BETWEEN komutunda sınırlar dahildir. (10 ve 20 dahil)

-- BETWEEN kullanırken ilk datanın ikinciden küçük olması gerekiyor.

-- Product\_id'si 20 ile 40 arasında olan productların isimlerini ve product\_id lerini gösteren kodu yazınız.

```
SELECT product_name, product_id
FROM customers_products
WHERE product_id BETWEEN 20 AND 40;
```

-- Product\_name ile Liked\_product'ı aynı olan müşterilerin isimlerini gösteren

-- kodu EXISTS kullanarak yazınız.

```
SELECT customer_name
FROM customers_products
WHERE EXISTS ( SELECT liked_product
              FROM customers_likes
              WHERE customers_products.product_name = customers_likes.liked_product
              );
```

-- ismi J'den T'ye kadar harflerle başlayan müşterilerin tüm bilgilerini gösteren kodu yazınız

```
SELECT *
FROM customers_products
WHERE customer_name BETWEEN 'J' AND 'T';
```

NOT BETWEEN, BETWEEN'in tersi olarak çalışır.

-- product\_id'si 20 ile 40 arasında olmayan product'ların tüm bilgilerini gösteren kodu yazınız.

```
SELECT *
FROM customers_products
WHERE product_id NOT BETWEEN 20 AND 40;
```

## IS NULL ☐ BOŞ ALANLARI SEÇ

-- IS NULL data girilmemiş olan satırları seçmek için kullanılır.

-- customers\_products Tablosundan müşteri ismi girilmemiş dataların tamamını gösteren kodu yazınız.

```
SELECT *  
FROM customers_products  
WHERE customer_name IS NULL;
```

-- IS NULL, UPDATE komutu ile de kullanılır

```
UPDATE customers_products  
SET customer_name = 'İsim girilmemiş'  
WHERE customer_name IS NULL;
```

## ORDER BY ☐ SIRALA

-- ORDER BY dataların belli bir field'a göre natural orda şeklinde sıralanmasına yarar.

-- ORDER BY'dan sonra field(sütun) ismi kullanıldığı gibi field numarasıda kullanılabilir.

-- Yani; ORDER BY product\_id; ile ORDER BY 1; aynı şeydir

-- customers\_products'daki tüm dataları product\_name'e göre sıralayan kodu yazınız.

```
SELECT *  
FROM customers_products  
ORDER BY product_name; -- product_name' e göre sırala dedik.
```

-- customers\_products'daki customer\_name'i 'Mark' olan dataları product\_id'ye göre sıralayan kodu yazınız.

```
SELECT *  
FROM customers_products  
WHERE customer_name = 'Mark'  
ORDER BY product_id;
```

-- aşağıdaki kod ile yukarıdaki aynı sonucu verir.

```
SELECT *  
FROM customers_products  
WHERE customer_name = 'Mark'  
ORDER BY 1;
```

## DESC ☐ BÜYÜKTEN KÜÇÜĞE SIRALAMA

-- customers\_products table'indeki tüm dataları product\_id'lerine göre

-- büyükten küçüğe(reverse order / descending order) sıralayınız.

```
SELECT *  
FROM customers_products  
ORDER BY product_id DESC;
```

## ASC ↗ KÜÇÜKTEN BÜYÜĞE SIRALAMA

- customers\_products table'ındaki tüm dataları product\_name'lerine göre
- büyükten küçüğe(reverse order / descending order) sıralayınız.
- customer\_name'lerine göre de natural order'da sıralayınız.

```
SELECT *  
FROM customers_products  
ORDER BY product_name DESC, customer_name ASC;
```

## AS ↗ RAPOR İÇİN SÜTUN İSİMLERİNİ DEĞİŞTİRME

- ALIASESE Tablodaki field(sütun) isimlerini çıktıda farklı görmek için kullanılır (Data Base değişmiyor)
- AS kullandığınızda database deki tablonun sütun isimleri değişmez siz
- sadece yeni sütun isimlerine sahip rapor alırsınız.

- Field isimlerini Türkçeleştirin.

```
SELECT product_id AS urun_kodu, customer_name AS musteri_adı, product_name AS  
urun_ismi  
FROM customers_products;
```

- Field isimlerini Türkçeleştirin ve product\_id ile product\_name aynı sütunda olsun.

```
SELECT customer_name AS musteri_ismi, product_id || product_name AS urun_kodu_ismi  
FROM customers_products;
```

- product\_id || product\_name AS urun\_kodu\_ismi

-- ÜSTTEKİ KOD İLE DATABASE DEĞİŞMEDEN İKİ SATIRI BİRLEŞİRİP O SÜTUNA YENİ İSİM ATAMA YAPTIK

## GROUP BY

- GROUP BY dataları gruplandırarak görmemizi sağlar

- Herbir product'ı alan müşteri sayısını gösteren kodu yazınız.

```
SELECT product_name, COUNT(product_name) AS number_of_customers  
FROM customers_products  
GROUP BY product_name
```

- Herbir product\_id'nin kaç kere kullanıldığını gösteren kodu yazınız.

```
SELECT product_id, COUNT(product_id) AS number_of_productID  
FROM customers_products  
GROUP BY product_id
```

- employees tablosundan her işçinin aldığı toplam ücreti bulunuz.

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

--Herbir state'de kaç işçi olduğunu gösteren kodu yazınız

```
SELECT state, COUNT(name) AS total_employees  
FROM employees  
GROUP BY state;
```

--Herbir şirkette maaşı 2000'inin üzerinde olan kaç işçi olduğunu gösteren kodu yazınız.

```
SELECT company, COUNT(name) AS total_employees  
FROM employees  
WHERE salary>2000  
GROUP BY company;
```

--Herbir şirkette verilen max ve min ücretleri gösteren kodu yazınız

```
SELECT company, MAX(salary) AS max_salary, MIN(salary) AS min_salary  
FROM employees  
GROUP BY company
```

**HAVING**

--HAVING, GROUP BY'dan sonra AGGREGATE functionlarla filtrelemek için kullanılır.

--Herbir şirketin min salary'lerini 2000 üzerinde ise göster ve max salary'lerini göster.

```
SELECT company, MIN(salary) AS min_salary, MAX(salary) AS max_salary  
FROM employees  
GROUP BY company  
HAVING MIN(salary)>2000;
```

--Toplam geliri 2500 den fazla olan herbir işçiyi gösteren kodu yazınız

```
SELECT name, SUM(salary) AS sum_salary  
FROM employees  
GROUP BY name  
HAVING SUM(salary)>2500;
```

--Herbir State'de çalışan işçi sayısını ve state ismini gösteren kodu yazınız

-- ama işçi sayısı birden fazla olmalı

```
SELECT state, COUNT(name) AS number_of_employees  
FROM employees  
GROUP BY state  
HAVING COUNT(name)>1;
```

--Herbir State'te verilen max ücreti 3000'den az ise gösteren kodu yazınız

```
SELECT state, MAX(salary) AS max_salary  
FROM employees  
GROUP BY state  
HAVING MAX(salary)<3000; --HAVING=...olan
```



## UNION

--UNION Operation: iki farklı sorgulamanın sonuçlarını birleştirme işlemidir.  
--Aynı sütunda birleştirilecek sütunların data typleri aynı olması gerekli yoksa hata verir.  
--UNION kullanırken her iki sorgudaki sütun data typleri örtüşmelidir.

--Salary'si 3000'den fazla olan state ve işçi isimlerini gösteren kodu yazınız.

```
SELECT state AS name_and_state,salary  
FROM employees  
WHERE salary>3000
```

## UNION

```
SELECT name AS name_and_state,salary  
FROM employees  
WHERE salary>3000;
```

--Eddie Murphy'inin aldığı toplam ücreti ve Florida'da ödenen ücretleri

--bir tabloda gösteren kodu yazınız

```
SELECT name AS name_state, salary  
FROM employees  
WHERE name = 'Eddie Murphy'
```

## UNION

```
SELECT state AS name_state, salary  
FROM employees  
WHERE state= 'Florida';
```

--state'lerde ödenen ücreti 3000'den fazla olan ve işçilere ödenen ücreti 2000

-- den az olanları bir tabloda gösteren kodu yazınız

```
SELECT state AS name_and_state, salary  
FROM employees  
WHERE salary>3000
```

## UNION

```
SELECT name AS name_and_state, salary  
FROM employees  
WHERE salary<2000;
```

## DAY 6

```
CREATE TABLE students
(
students_id char(9),
students_name varchar2(50),
students_address varchar2(80),
students_grade number(3),
last_modification_date date,
CONSTRAINTS id_pk PRIMARY KEY(students_id)
);
```

```
INSERT INTO students VALUES('123456789', 'Ali Can', 'Istanbul', 93, '12-Aug-20');
INSERT INTO students VALUES('234567890', 'Veli Han', 'Istanbul', 95, '13-Aug-20');
INSERT INTO students VALUES('345678901', 'Ayse Tan', 'Berlin', 95, '13-Aug-20');
```

```
SELECT *
FROM students;
```

```
CREATE TABLE students_information
(
students_id char(9),
students_phone char(10) UNIQUE,
students_avg_score number(4,2) NOT NULL,
CONSTRAINTS id_fk FOREIGN KEY(students_id) REFERENCES students(students_id)
);
```

```
INSERT INTO students_information VALUES('123456789', '4071234567', 78);
INSERT INTO students_information VALUES('234567890', '4071234598', 91);
INSERT INTO students_information VALUES('345678901', '4071230000', 93);
```

```
SELECT *
FROM students_information;
```

--students tablosundan 'Ali Can' ın adresini ve grade'ini alınız.students\_information  
--tablosundan da Ali Can'ın tlf numarası ve average score(ortalamasını) alınız  
-- ve bunları bir tabloda birleştirerek gösteriniz

```
SELECT students_name AS id_name, students_address AS address_phone, students_grade
AS avg_score_grade
FROM students
WHERE students_id = '123456789'
```

UNION

```
SELECT students_id , students_phone, students_avg_score
FROM students_information
WHERE students_id = '123456789'
```

--UNION kullanırken üstteki tablo ile alttaki tablo sütun sayıları aynı olması lazım.  
--eşleştirilen sütunların data typleri aynı olması lazım yoksa hata verir.  
--UNION işlemi aynı recod(satır)ları sadece bir kere yazar.  
--UNION işlemi kullanırsanız Tabloda tekrarlı record göremezsiniz.

--aynı query'i union yaptığımızda bunu görürüz.

```
SELECT students_id, students_address, students_grade
FROM students
WHERE students_id = '123456789'
```

UNION

```
SELECT students_id, students_address, students_grade
FROM students
WHERE students_id = '123456789';
```

UNION ALL

--olsun ben ikisinide görmek istiyorum dersiniz o zaman UNION ALL kullanabilirsiniz.  
SELECT students\_id, students\_address, students\_grade  
FROM students  
WHERE students\_id = '123456789'

UNION ALL

```
SELECT students_id, students_address, students_grade
FROM students
WHERE students_id = '123456789';
```

INTERSECT

--iki tane query'nin(sorgu) ortak sonuçlarını görmek için kullanılır  
--iki sorgulamada kesişim olanları göster  
-- INTERSECT işleminde kesişimde ortak eleman bulamazsa no data founda yazar error vermez.

```
SELECT students_grade AS grade_avg_score
```

```
FROM students
```

```
INTERSECT
```

```
SELECT students_avg_score  
FROM students_information;
```

--Grade'i 94'ten küçük olanlarla, average score'u 80 den büyük olanların kesişimini gösteriniz.

```
SELECT students_grade  
FROM students  
WHERE students_grade < 94
```

```
INTERSECT
```

```
SELECT students_avg_score  
FROM students_information  
WHERE students_avg_score > 80;
```

```
CREATE TABLE employees  
(  
    id number(9),  
    name varchar2(50),  
    state varchar2(50),  
    salary number(20),  
    company varchar2(20)  
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');  
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');  
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');  
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');  
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');  
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');  
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

--salary'si 3000 den çok , 2000'den az olan işçilerin tüm bilgilerini tekrar olmadan  
--bir tabloda gösteren kodu yazınız.

```
SELECT *  
FROM employees  
WHERE salary > 3000
```

## UNION

```
SELECT *  
FROM employees  
WHERE salary < 2000
```

--salary'si 3000'den çok 2000'den az olan işçilerin ortak isimlerini gösteren kodu yazınız.

```
SELECT name  
FROM employees  
WHERE salary > 3000
```

## INTERSECT

```
SELECT name  
FROM employees  
WHERE salary < 2000
```

--IBM, APPLE ve GOOGLE da çalışıp 3000 den fazla salary'si olan işçileri gösteren kodu yazınız

```
SELECT name  
FROM employees  
WHERE company IN ('IBM','APPLE','GOOGLE')
```

## INTERSECT

```
SELECT name  
FROM employees  
WHERE salary > 3000;
```

## MINUS

--MINUS işlemi ilk query(sorgu)'nun sonuçları arasından ikinci query'de olanları  
--silme için kullanılır

--google'da çalışmayıp 2000'den düşük maaş alanların isimlerini ve company isimlerini  
--gösteren kodu yazınız

```
SELECT name, company  
FROM employees  
WHERE salary < 2000
```

## MINUS

```
SELECT name, company  
FROM employees  
WHERE company = 'GOOGLE';
```

--ismi Eddy Murphy olup Texas'ta yaşamayanları ve statelerini gösterenleri gösteren kodu yazınız

```
SELECT name, state  
FROM employees  
WHERE name = 'Eddie Murphy'
```

MINUS

```
SELECT name, state  
FROM employees  
WHERE state = 'Texas';
```

## JOIN

--JOINS: iki tablodaki dataları birleştirmeye yarar.  
--JOINS tabloları birleştirmeye, UNION sorgulamaları birleştirmeye yarar.  
--5 tane JOINS var.  
-- 1) INNER JOIN : iki tablodaki ortak dataları gösterir.  
-- 2) LEFT JOIN : ilk tabloda var olan dataları gösterir.  
-- 3) RIGHT JOIN :ikinci tabloda var olan dataları gösterir.  
-- 4) FULL JOIN: iki tablodaki var olan tüm dataları gösterir.  
-- 5) SELF JOIN: tabloyu kendi kendiyile birleştirme işlemi.

```
CREATE TABLE companies  
(  
    company_id number(9),  
    company_name varchar2(20)  
);
```

```
INSERT INTO companies VALUES(100, 'IBM');  
INSERT INTO companies VALUES(101, 'GOOGLE');  
INSERT INTO companies VALUES(102, 'MICROSOFT');  
INSERT INTO companies VALUES(103, 'APPLE');
```

```
SELECT *  
FROM companies;
```

-- order: sipariş demek

```
CREATE TABLE orders  
(  
    order_id number(9),  
    company_id number(9),  
    order_date date  
);
```

```
INSERT INTO orders VALUES(11, 101, '17-Apr-2020');
INSERT INTO orders VALUES(22, 102, '18-Apr-2020');
INSERT INTO orders VALUES(33, 103, '19-Apr-2020');
INSERT INTO orders VALUES(44, 104, '20-Apr-2020');
INSERT INTO orders VALUES(55, 105, '21-Apr-2020');
```

```
SELECT *
FROM orders;
```

--Companies table'daki Company id'leri ile orders table'daki company id'leri  
--aynı olan order'ların campanyi name'lerini, order id'lerini ve order date'lerini  
-- gösteren bir talo oluşturan kodu yazınız.

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies
```

INNER JOIN orders -- companies tablosu orders tablosuyla birleştir, bu birleşimden  
aşağıdaki kurala göre alacaklarını al.  
ON companies.company\_id = orders.company\_id;

-- 1) SELECT'den sonra Tabloda görmek istediğiniz sütun isimlerini yazarken  
-- tablo ismi + nokta + sütun ismi şeklinde yazınız.  
-- 2) FROM'dan sonra iki tablo ile çalıştığımızdan FROM'dan sonra Tablo ismi  
-- yazarken birinci tablo ismi + INNER JOIN + ikinci tablo ismi yazmalıyız.  
-- 3) JOIN'i hangi kurala göre yapacağımızı belirtmelisiniz bunun için de ON + kuralınızı  
-- yazmalısınız.  
-- 4) INNER JOIN yerine sadece JOIN'de yazılabilir.

## LEFT JOIN

--LEFT JOIN'de ilk tablodaki tüm datalar gösterilir.İlk tablodaki datalara ikinci tablodan  
--gelen ek datalar var ise bu ek datalar ortak datalar için gösterilir ancak ortak olmayan  
--datalar için o kısımlar boş bırakılır.

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies
```

```
LEFT JOIN orders
ON companies.company_id = orders.company_id;
```

## RIGHT JOIN

--RIGHT JOIN'de ilk tablodaki tüm datalar gösterilir. İlk tablodaki datalara ikinci tablodan gelen ek datalar var ise bu ek datalar ortak datalar için gösterilir ancak ortak olmayan datalar için o kısımlar boş bırakılır.

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies
```

RIGHT JOIN orders

```
ON companies.company_id = orders.company_id;
```

## FULL JOIN

--FULL JOIN'de iki tablodada var olan tüm datalar gösterilir. genellikle bazı datalar boş kalabilir.

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies
```

FULL JOIN orders

```
ON companies.company_id = orders.company_id;
```

## DAY 7

### WILDCARDS

--Wildcards ==> % \_ [ ]

```
CREATE TABLE customers
```

```
(
customer_id number(10) UNIQUE,
customer_name varchar2(50) NOT NULL,
income number(6)
);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1001, 'John', 62000);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1002, 'Jane', 57500);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1003, 'Brad', 71000);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1004, 'Manse', 42000);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1005, 'Can', 57500);
```



```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1006, 'Cin', 71000);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1007, 'Con', 42000);
```

```
SELECT *
FROM customers
```

% : Bir veya birden fazla karakter gösterir.

--İsmi J harfi ile başlayan müşterilerin tüm bilgilerini gösteren bir tablo  
-- oluşturmak için kod yazınız.

```
SELECT *
FROM customers
WHERE customer_name LIKE 'J%';
```

LIKE gibi demek J% : J ile başlasın devamı ne olursa olsun demektir.

--İsmi 'e' ile biten müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan  
-- kodu yazınız

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '%e';
```

--İsminin içinde 'n' olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan  
-- kodu yazınız

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '%n%';
```

\_ : Sadece bir karakteri temsil eder, ne olduğu önemli değil.

--İsmi dört harfli olup son üç harfi 'ohn' olan müşterilerin müşteri adı ve gelirlerini  
--gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '_ohn';
```

--İsmi dört harfli olup son iki harfi 'ne' olan müşterilerin müşteri adı ve gelirlerini  
--gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '__ne';
```

--İsmi dört harfli olup son üçüncü harfi 'n' olan müşterilerin müşteri adı ve gelirlerini  
--gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '__n_';
```

--ikinci harfi 'a' olan müşterilerin müşteri adı ve gelirlerini  
--gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '_a%';
```

--Üçüncü harfi 'n' olan en az 5 harfli isme sahip olan müşterilerin müşteri adı ve gelirlerini  
--gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE '__n__%';
```

--'B' ile başlayıp 3.harfi 'a' olan isimlere sahip olan müşterilerin müşteri adı ve gelirlerini  
--gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE customer_name LIKE 'B_a%';
```

[ ] : sembol içine koyulan harflerden herhangi birisi olabilir manasında

--İlk harfi 'C' olan son harfi 'n' olan ikinci harfi 'a' veya 'i' olan 3 harfli isimlere sahip olan  
müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name,income
FROM customers
WHERE REGEXP_LIKE(customer_name, 'C[ai]n');
--REGEXP: Regularexpaction
```

--İlk harfi 'C' olan son harfi 'n' olan ikinci harfi 'a' dan 'k'ya tüm harfler olan 3 harfli isimlere  
sahip olan müşterilerin

--müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız

```
SELECT customer_name,income
FROM customers
WHERE REGEXP_LIKE(customer_name, 'C[a-k]n');
```

--İçinde 'a' veya 'n' olan tüm isimlere sahip olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu  
oluşturan kodu yazınız.

```
SELECT customer_name, income
FROM customers
WHERE REGEXP_LIKE(customer_name, '[an](*)');
```

--'J' veya 'M' ile başlayan tüm isimlere sahip olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name, income
FROM customers
WHERE REGEXP_LIKE(customer_name, '^[JM](*)');
```

**^** : ile başlayan manasında

**(\*)** : herşey manasında

--İlk harfi 'J' olmayan tüm isimlere sahip olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name, income
FROM customers
WHERE customer_name NOT LIKE 'J%';
```

--'a' içermeyen tüm isimlere sahip olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name, income
FROM customers
WHERE customer_name NOT LIKE '%a%';
```

--İkinci harfi 'a' olmayan isimlere sahip olan müşterilerin müşteri adı ve gelirlerini gösteren tabloyu oluşturan kodu yazınız.

```
SELECT customer_name, income
FROM customers
WHERE customer_name NOT LIKE '_a%';
```

## DAY 8

```
CREATE TABLE customers_products
(
    product_id number(10),
    customer_name varchar2(50),
    product_name varchar2(50)
);
```

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

```
SELECT *
FROM customers_products;
```

PRODUCT_ID	CUSTOMER_NAME	PRODUCT_NAME
10	Mark	Orange
10	Mark	Orange
20	John	Apple
30	Amy	Palm
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

**PIVOT: Satırları sütuna dönüştürmek için kullanılır. Özet Tablo(Excel)**

```
SELECT * FROM (SELECT customer_name, product_name FROM customers_products)
PIVOT
(COUNT(product_name) FOR product_name IN ('Orange', 'Apple', 'Palm', 'Apricot'));
--product nameleri say product name için içinde orange, apple, palm, apricot
```

CUSTOMER_NAME	'Orange'	'Apple'	'Palm'	'Apricot'
Amy	0	0	1	0
Mark	2	1	0	0
Adem	1	0	0	0
Eddie	0	1	0	0
John	0	1	0	1

**SELECT SUM**

```
SELECT * FROM (SELECT product_id, product_name FROM customers_products)
PIVOT
(SUM (product_id) FOR product_id IN (10,20,30,40));
```

PRODUCT_NAME	10	20	30	40
Orange	30	–	–	–
Apple	–	60	–	–
Palm	–	–	30	–
Apricot	–	–	–	40

[Download CSV](#)

**DISTINCT:** sütun içerisinde geçen dataları tekrarsız olarak gösteren kod.

--product name leri tekrarsız olarak gösteren kodu yazınız.

```
SELECT DISTINCT product_name
FROM customers_products;
```

PRODUCT_NAME
Orange
Apple
Palm
Apricot

[Download CSV](#)

--kac farklı meyve var bunun sayısını gösteren kodu yazınız

```
SELECT COUNT(DISTINCT product_name) AS meyve_cesit_sayisi
FROM customers_products;
```

MEYVE_CESIT_SAYISI
4

## ALTER TABLE: Tablolar üzerinde değişiklik yapma

```
CREATE TABLE employees
(  
  id number(9),  
  name varchar2(50),  
  state varchar2(50),  
  salary number(20),  
  company varchar2(20)  
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');  
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');  
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');  
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');  
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');  
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');  
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

```
SELECT *  
FROM employees;
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

## ALTER TABLE

ADD: Tabloya yeni sütun ekleme

```
ALTER TABLE employees  
ADD gender varchar2(20)
```

ID	NAME	STATE	SALARY	COMPANY	GENDER
123456789	John Walker	Florida	2500	IBM	–
234567890	Brad Pitt	Florida	1500	APPLE	–
345678901	Eddie Murphy	Texas	3000	IBM	–
456789012	Eddie Murphy	Virginia	1000	GOOGLE	–
567890123	Eddie Murphy	Texas	7000	MICROSOFT	–
456789012	Brad Pitt	Texas	1500	GOOGLE	–
123456710	Mark Stone	Pennsylvania	2500	IBM	–

--Tabloya country sütunu ekleyin her bir satır içinde 'The USA' olsun

ALTER TABLE employees

ADD country varchar2(50) DEFAULT 'The USA';

ID	NAME	STATE	SALARY	COMPANY	GENDER	COUNTRY
123456789	John Walker	Florida	2500	IBM	–	The USA
234567890	Brad Pitt	Florida	1500	APPLE	–	The USA
345678901	Eddie Murphy	Texas	3000	IBM	–	The USA
456789012	Eddie Murphy	Virginia	1000	GOOGLE	–	The USA
567890123	Eddie Murphy	Texas	7000	MICROSOFT	–	The USA
456789012	Brad Pitt	Texas	1500	GOOGLE	–	The USA
123456710	Mark Stone	Pennsylvania	2500	IBM	–	The USA

--Tabloya aynı anda birden fazla sütun ekleme

ALTER TABLE employees

ADD (number\_of\_kid number(2),  
marital\_status varchar2(30) DEFAULT 'Single'  
);

ID	NAME	STATE	SALARY	COMPANY	GENDER	COUNTRY	NUMBER_OF_KID	MARITAL_STATUS
123456789	John Walker	Florida	2500	IBM	–	The USA	–	Single
234567890	Brad Pitt	Florida	1500	APPLE	–	The USA	–	Single
345678901	Eddie Murphy	Texas	3000	IBM	–	The USA	–	Single
456789012	Eddie Murphy	Virginia	1000	GOOGLE	–	The USA	–	Single
567890123	Eddie Murphy	Texas	7000	MICROSOFT	–	The USA	–	Single
456789012	Brad Pitt	Texas	1500	GOOGLE	–	The USA	–	Single
123456710	Mark Stone	Pennsylvania	2500	IBM	–	The USA	–	Single

### DROP COLUMN: Tablodan sütun silme

ALTER TABLE employees

DROP COLUMN country;

ID	NAME	STATE	SALARY	COMPANY	GENDER	NUMBER_OF_KID	MARITAL_STATUS
123456789	John Walker	Florida	2500	IBM	–	–	Single
234567890	Brad Pitt	Florida	1500	APPLE	–	–	Single
345678901	Eddie Murphy	Texas	3000	IBM	–	–	Single
456789012	Eddie Murphy	Virginia	1000	GOOGLE	–	–	Single
567890123	Eddie Murphy	Texas	7000	MICROSOFT	–	–	Single
456789012	Brad Pitt	Texas	1500	GOOGLE	–	–	Single
123456710	Mark Stone	Pennsylvania	2500	IBM	–	–	Single

### RENAME COLUMN: Sütun ismini değiştirme

ALTER TABLE employees

RENAME COLUMN gender TO gender\_male\_or\_female;

ID	NAME	STATE	SALARY	COMPANY	GENDER_MALE_OR_FEMALE	NUMBER_OF_KID	MARITAL_STATUS
123456789	John Walker	Florida	2500	IBM	–	–	Single
234567890	Brad Pitt	Florida	1500	APPLE	–	–	Single
345678901	Eddie Murphy	Texas	3000	IBM	–	–	Single
456789012	Eddie Murphy	Virginia	1000	GOOGLE	–	–	Single
567890123	Eddie Murphy	Texas	7000	MICROSOFT	–	–	Single
456789012	Brad Pitt	Texas	1500	GOOGLE	–	–	Single
123456710	Mark Stone	Pennsylvania	2500	IBM	–	–	Single



### RENAME TO: Tablo ismini deęiřtirme

```
ALTER TABLE employees  
RENAME TO workers;
```

WORKERS

Table

Status: Valid

Created 26 minutes ago

### MODIFY: Tablo sřtunlarının yapısını deęiřtirme

```
ALTER TABLE workers  
MODIFY id number(9) NOT NULL;
```

#	Column	Type	Length	Precision	Scale	Nullable
1	ID	NUMBER	22	9	0	No

### --Birden fazla sřtunun yapısını aynı anda deęiřtirmek

```
ALTER TABLE workers  
MODIFY (state char(45) NOT NULL,  
        company char(30)  
);
```

#	Column	Type	Length	Precision	Scale	Nullable
1	ID	NUMBER	22	9	0	No
2	NAME	VARCHAR2	50			Yes
3	STATE	CHAR	45			No
4	SALARY	NUMBER	22	20	0	Yes
5	COMPANY	CHAR	30			Yes

## SQL Interview Questions

```
CREATE TABLE students
(
  id number(9),
  name varchar2(50),
  state varchar2(50),
  salary number(20),
  company varchar2(20)
);
```

```
INSERT INTO students VALUES(123456789, 'Johnny Walk', 'New Hampshire', 2500, 'IBM');
INSERT INTO students VALUES(234567891, 'Brian Pitt', 'Florida', 1500, 'LINUX');
INSERT INTO students VALUES(245678901, 'Eddie Murphy', 'Texas', 3000, 'WELLS FARGO');
INSERT INTO students VALUES(456789012, 'Teddy Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO students VALUES(567890124, 'Eddie Murphy', 'Massachuset', 7000,
'MICROSOFT');
INSERT INTO students VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'TD BANK');
INSERT INTO students VALUES(123456719, 'Adem Stone', 'New Jersey', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	Johnny Walk	New Hampshire	2500	IBM
234567891	Brian Pitt	Florida	1500	LINUX
245678901	Eddie Murphy	Texas	3000	WELLS FARGO
456789012	Teddy Murphy	Virginia	1000	GOOGLE
567890124	Eddie Murphy	Massachuset	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	TD BANK
123456719	Adem Stone	New Jersey	2500	IBM

```
CREATE TABLE workers01
(
  id number(9),
  name varchar2(50),
  state varchar2(50),
  salary number(20),
  company varchar2(20)
);
```

```

INSERT INTO workers01 VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO workers01 VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO workers01 VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO workers01 VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO workers01 VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO workers01 VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO workers01 VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');

```

```

SELECT *
FROM workers01;

```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

--her iki tablodaki ortak id ve isimleri gösteren kodu yazınız

```

SELECT id,name
FROM students

```

```

INTERSECT

```

```

SELECT id,name
FROM workers01;

```

ID	NAME
456789012	Brad Pitt

[Download CSV](#)

--students tablosunda kac farklı state'den student var.

```

SELECT COUNT (DISTINCT state) AS num_of_states
FROM students;

```

NUM_OF_STATES
6

[Download CSV](#)

**MOD METODU: çift veya tek sayıları filtreleme.**

--id si çift sayı olan öğrencilerin tüm bilgilerini gösteren kodu yazınız

```
SELECT *
FROM students
WHERE MOD(id,2)=0;
```

ID	NAME	STATE	SALARY	COMPANY
456789012	Teddy Murphy	Virginia	1000	GOOGLE
567890124	Eddie Murphy	Massachuset	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	TD BANK

----id si tek sayı olan öğrencilerin tüm bilgilerini gösteren kodu yazınız

```
SELECT *
FROM students
WHERE MOD(id,2)=1;
```

ID	NAME	STATE	SALARY	COMPANY
123456789	Johnny Walk	New Hampshire	2500	IBM
234567891	Brian Pitt	Florida	1500	LINUX
245678901	Eddie Murphy	Texas	3000	WELLS FARGO
123456719	Adem Stone	New Jersey	2500	IBM

--Table'da kaç tane record olduğunu gösteren kodu yazınız.

```
SELECT COUNT(*) AS num_of_record
FROM students;
```

NUM_OF_RECORD
7

[Download CSV](#)

**ROWNUM: Tabloda istenilen sayıda satır sayısını gösterme**

--workers01 tablosunda en yüksek ücreti olan işçinin tüm bilgilerini gösteren kodu yazınız

```
SELECT *
FROM workers01
WHERE ROWNUM = 1; -- sadece 1 satır göster demek
ORDER BY salary DESC
```

--workers01 tablosunda en yüksek ücreti olan işçinin tüm bilgilerini gösteren kodu yazınız

```
SELECT *
FROM workers01
WHERE ROWNUM = 1; -- sadece 1 satır göster demek
ORDER BY salary DESC
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM

[Download CSV](#)

--en yüksek ücreti alan işçinin tüm bilgilerini gösteren kodu yazınız.(SUBQUERY)

```
SELECT *
FROM workers01
WHERE salary = (SELECT MAX(salary) AS Max_salary
                FROM Workers01
);
```

--en yüksek ücreti alan işçinin tüm bilgilerini gösteren kodu yazınız.(SUBQUERY)

```
SELECT *
FROM workers01
WHERE salary = (SELECT MAX(salary) AS Max_salary
                FROM Workers01
);
```

ID	NAME	STATE	SALARY	COMPANY
567890123	Eddie Murphy	Texas	7000	MICROSOFT

[Download CSV](#)

--ikinci en yüksek maaşı alan işçinin tüm bilgilerin gösteren kodu yazınız

```
SELECT MAX(salary)
FROM workers01
WHERE salary < (SELECT MAX(salary)
                FROM workers01
                );
```

-- < select max(salary) demek en büyükten küçük olanlar demek bunların içinden  
-- max olanı göster diyor.

MAX(SALARY)
3000

[Download CSV](#)

--ikinci en düşük maaşı gösteren kodu yazınız

```
SELECT MIN(salary)
FROM workers
WHERE salary > (SELECT MIN(salary)
                FROM workers01
                );
```

MIN(SALARY)
1500

[Download CSV](#)

## DAY 9

--Salary'si en yuksek olan isci disindaki iscilerin tum bilgilerini  
--salary'e gore buyukten kucuge dizerek gosteren kodu yaziniz.

```
CREATE TABLE employees  
(  
  id number(9),  
  name varchar2(50),  
  state varchar2(50),  
  salary number(20),  
  company varchar2(20)  
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');  
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');  
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');  
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');  
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');  
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');  
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

```
SELECT *  
FROM employees  
WHERE salary != (SELECT MAX(salary)  
                 FROM employees)  
ORDER BY salary DESC;
```

ID	NAME	STATE	SALARY	COMPANY
345678901	Eddie Murphy	Texas	3000	IBM
123456710	Mark Stone	Pennsylvania	2500	IBM
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
456789012	Brad Pitt	Texas	1500	GOOGLE
456789012	Eddie Murphy	Virginia	1000	GOOGLE

[Download CSV](#)

--ikinci en yüksek maaş alan kişinin tüm bilgilerini gösteren kodu yazınız.

```
SELECT *
FROM (SELECT *
      FROM employees
      WHERE salary != (SELECT MAX(salary)
                      FROM employees)
      ORDER BY salary DESC)
WHERE ROWNUM = 1;
```

ID	NAME	STATE	SALARY	COMPANY
345678901	Eddie Murphy	Texas	3000	IBM

[Download CSV](#)

--2.Yol

```
SELECT *
FROM employees
ORDER BY salary DESC
OFFSET 1 ROW -- 1.satırı göremezden gel
FETCH NEXT 1 ROW ONLY; -- 1 satır sonrakini getir sadece
```

ID	NAME	STATE	SALARY	COMPANY
345678901	Eddie Murphy	Texas	3000	IBM

[Download CSV](#)



```
CREATE TABLE workers
(
  id number(2),
  name varchar2(20),
  title varchar2(60),
  boss_id number(2)
);
```

```
INSERT INTO workers VALUES(1, 'Ali Can', 'SDET', 2);
INSERT INTO workers VALUES(2, 'John Walker', 'QA', 3);
INSERT INTO workers VALUES(3, 'Angie Star', 'QA Lead', 4);
INSERT INTO workers VALUES(4, 'Amy Sky', 'CEO', 5);
```

```
SELECT *
FROM workers;
```

ID	NAME	TITLE	BOSS_ID
1	Ali Can	SDET	2
2	John Walker	QA	3
3	Angie Star	QA Lead	4
4	Amy Sky	CEO	5

[Download CSV](#)

### SELF JOIN

SELF JOIN iki tablo ile çalışır. Bir tabloyu iki tablo gibi düşündük birine w1 diğerine de w2 dedik.

--Her işçinin patronunu gösteren tabloyu oluşturunuz

SELECT w1.name AS worker\_name, w2.name AS boss\_name --w1 tablosundan name'i al  
worker\_name olarak adlandır

-- w2 tablosundan name'i al boss\_name olarak adlandır.

FROM workers w1 INNER JOIN workers w2 --burada w1 ve w2 adında aynı iki tablo  
oluşturmuş olduk.

ON w1.boss\_id = w2.id; -- w1 deki boss\_id w2 deki id'ye eşit olsun.

WORKER_NAME	BOSS_NAME
Ali Can	John Walker
John Walker	Angie Star
Angie Star	Amy Sky

[Download CSV](#)

3 rows selected.

UPPER: Bir sütundaki dataların büyük harfle yazılmasını sağlar.

LOWER: Bir sütundaki dataların küçük harfle yazılmasını sağlar.

INITCAP: Bir sütundaki dataların ilk harfini büyük yapar diğerlerini küçük.

```
SELECT INITCAP(name), UPPER(state), LOWER(company)
FROM employees;
```

INITCAP(NAME)	UPPER(STATE)	LOWER(COMPANY)
John Walker	FLORIDA	ibm
Brad Pitt	FLORIDA	apple
Eddie Murphy	TEXAS	ibm
Eddie Murphy	VIRGINIA	google
Eddie Murphy	TEXAS	microsoft
Brad Pitt	TEXAS	google
Mark Stone	PENNSYLVANIA	ibm

[Download CSV](#)