T.C.
ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Veri Tabanı Laboratuvarı

Deney Föyü-3

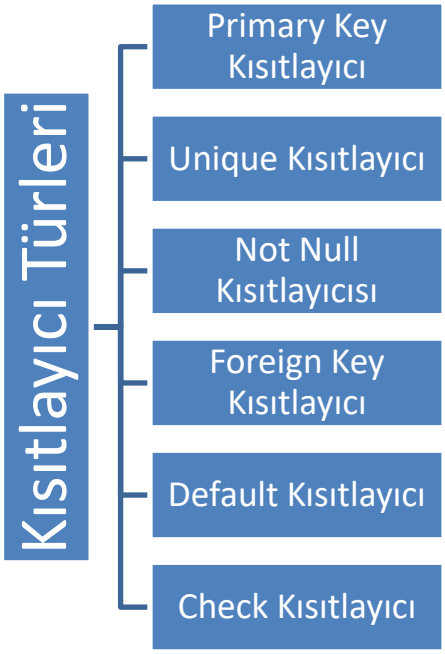
Kısıtlar ve Görünüm

1. **Constraint (Kısıtlayıcı):** Bir veritabanı içerisine eklenen, düzenlenen ve yönetilen verileri tutarlı bir durumda saklamak önemlidir. Veritabanı üzerinde iş kurallarını zorunlu kılmak ve tablolar arasında ilişki kurarak veri bütünlüğünü sağlamak amacıyla oluşturulur. Tablo sütunlarında hangi değerlerin depolanabileceğine veya sınırlanmasına izin veren kuralları tanımlamak için kullanılır. Constraintler veri bütünlüğü hususunda MySql Server'in kullanıcılarına sunduğu yapıdır. Veri üzerindeki mantıksal sınırlamalara kısıt adı verilir. Kısıtların genel olması tercih edilen bir durumdur. Kısıtlar, veri modellerinde bütünlük sağlamak için kullanılır. MySQL kısıtları sütun seviyesi ve tablo seviyesi olmak üzere iki tip olarak sınıflandırılabilir. Sütun düzeyi kısıtlamaları, yalnızca tüm tabloya tablo düzeyi kısıtlamaları uygulandığında bir sütun için geçerli olabilir. Kısıtlamalar, tabloların tanımlanmasıyla beraber oluşan öğelerdir. Kısıtlamalar ile Rule (kural) ve Default'ların (varsayılan) yapabileceği işler yapılabilir.

Kısıtlamalar: Tablo oluşturulurken **CREATE TABLE** komutuyla tanımlanır.

Tablo oluşturulmuşsa **ALTER TABLE** komutuyla tanımlanır. ALTER TABLE komutuyla kullanıldığında sütunlara girilen bilgilerin dikkate alınması gerekir.

Constraint (Kısıtlayıcı)Türleri:



I. **Birincil Anahtar Kısıtlayıcı (Primary Key Constraint):** Bir tablo kaydının benzersiz olmasını sağlayan bir kısıttır. Aynı olmayan değerler girilmesini sağlayarak her kaydın farklı olması garantilenir. Her tablonun en fazla 1 adet Birincil Anahtar Kısıtlayıcısı olabilir. Böylelikle bu kısıt altında, aynı değer iki kez girilmemiş olur. Bir Birincil Anahtar Kısıtlayıcı asla NULL olamaz.

Örnek: "Ogrenciler" tablosu ele alındığında: "ogr_no" alanı primary key olarak tanımlanmak istenirse:

```
CREAT TABLE Ogrenciler
(
ogr_no int NOT NULL,
ad varchar(255) NOT NULL,
soyad varchar(255) NOT NULL,
bolum varchar(255,
PRIMARY KEY(ogr_no)
);
```

Alter Table cümleciği ile primary key tanımlaması:

```
ALTER TABLE Ogrenciler
ADD PRIMARY KEY(ogr_no)
```

Tanımlaması kullanılırken, birden fazla alanın primary key olarak tanımlanmasında:

```
ALTER TABLE Ogrenciler
ADD CONSTRAINT pk_ogr_no PRIMARY KEY(ogr_no,ad)
```

ifadesi kullanılır. Oluşturulan primary key kısıtlayıcısı silinmek istendiğinde ise:

```
ALTER TABLE Ogrenciler
DROP PRIMARY KEY
```

ifadesi kullanılır.

II. **Tekil Alan Kısıtlayıcı (Unique Constraint):** Bir tablodaki primary key constraint dışındaki aday anahtarlar(candidate keys)'a verilen isimdir. Birincil anahtar olan ve tablodaki diğer alanlar içinde aynı içeriğe sahip verilerin olmaması için Tekil Alan Kısıtlayıcı tanımlanır. Bir tekil alan kısıtlayıcı, NULL olabilen birincil anahtar kısıtlayıcı olarak adlandırılır. Unique Kısıtlayıcı ve Primary Key Kısıtlayıcılarının her ikisi de benzer kayıtların girilmesinin engellenmesini sağlar. Bir Primary Key Kısıtlayıcılayıcısı tanımlandığı zaman otomatik olarak Unique kısıtlayıcısda tanımlanmış olur. Burada dikkat edilmesi gereken nokta tablo için birçok Unique kısıtlayıcısı olabilir fakat bir tablonun sadece bir tane Primary Key Kısıtlayıcısı olabilir.

Örnek: ”Ogrenciler” tablosu ele alındığında: “ogr_no” alanı unique olarak tanımlanmak istenirse:

```
CREAT TABLE Ogrenciler
(
ogr_no int NOT NULL,
ad varchar(255) NOT NULL,
soyad varchar(255) NOT NULL,
bolum varchar(255,
UNIQUE(ogr_no)
);
```

Birden fazla alan unique olarak tanımlanmak istenirse:

```
CREAT TABLE Ogrenciler
(
ogr_no int NOT NULL,
ad varchar(255) NOT NULL,
soyad varchar(255) NOT NULL,
bolum varchar(255,
CONSTRAINT kisitlayici UNIQUE(ogr_no,ad)
);
```

Alter Table cümleciği ile unique tanımlaması:

```
ALTER TABLE Ogrenciler
ADD UNIQUE (ogr_no)
```

```
ALTER TABLE Ogrenciler
ADD CONSTRAINT kisitlayici UNIQUE(ogr_no,ad)
```

Oluşturulan unique kısıtlayıcısı silinmek istendiğinde ise aşağıdaki ifade kullanılır:

```
ALTER TABLE Ogrenciler
DROP INDEX kisitlayici
```

III. **Not Null Constraint:** NOT Null kısıtlayıcı bir kolondaki verilerin boş olmamasını sağlar.

Örnek: ”Ogrenciler” tablosu ele alındığında: “ogr_no” alanı unique olarak tanımlanmak istenirse:

```
CREATE TABLE Ogrenciler
(
ogr_no int NOT NULL,
ad varchar(255) NOT NULL,
soyad varchar(255) NOT NULL,
bolum varchar(255)
);
```

IV. **Yabancıl anahtar kısıtlayıcı (Foreign Key Constraint):** Bir tablodaki sütun değerinin, ilişkili olduğu başka bir tablodaki sütun tarafından geldiğini doğrulamak için kullanılan bir kısıttır.

Örnek: Kişiler ve İller tabloları şu şekilde oluşturulsun:

id	ad	soyad	adres	il_kodu
1	Süleyman	Akgül	Yeni Mahalle, Şirin Sokak ,Saray Önü Apt No:13/4	23
2	Harun	Kolyiğit	Yeni Mahalle Şirin, Sokak Pınar ,Apt No:11/2	23
3	Eser	Gürbüz	Cingen Mah., Kimsesizler Sok. ,Huzur Apt. Bodrum Kat	42
4	Sinan	Tunç	Yeşil Mahalle,Tuna Sok.,Beyzade Apt.N0:11/1	70

il_kodu	il_adi
1	Adana
23	Elazığ
34	İstanbul
42	Konya
70	Karaman

Kişiler tablosundaki "il_kodu" sütununun "iller" tablosundan geldiği doğrulanırken foreing kısıtlayıcı tanımlaması:

```
CREATE TABLE kisiler
(
id int NOT NULL ,
ad varchar(255)NOT NULL,
soyad varchar(255)NOT NULL,
adres varchar(255)NOT NULL,
il_kodu int NOT NULL,
PRIMARY KEY(id),
FOREIGN KEY(il_kodu)REFERENCES iller(il_kodu)
)
```

Alter Table cümleciği ile foreing kısıtlayıcı tanımlaması:

```
ALTER TABLE kisiler
ADD CONSTRAINT fk_kisi_il
FOREIGN KEY (il_kodu)
REFERENCES iller(il_kodu)
```

Oluşturulan foreing kısıtlayıcısı silinmek istendiğinde ise aşağıdaki ifade kullanılır:

```
ALTER TABLE kisiler
DROP FOREIGN KEY fk_kisi_il
```

V. Varsayılan Kısıtlayıcı (Default Constraint): Tablodaki herhangi bir alan için girilmesi gereken bir değerin atanmasıdır. Başka bir deyişle bir sütuna veri eklenmemesi durumunda otomatik olarak varsayılan bir değer eklenmesi varsayılan kısıtlayıcı ile sağlanabilir. INSERT komutu için geçerlidir. **Örnek:** kişi bilgilerinin alındığı bir tabloda kişinin uyruğunun girilmesi işleminde varsayılan değer olarak "T.C." atanabilir. CREATE TABLE cümleciğinde default kısıtlayıcı tanımlaması:

```
CREATE TABLE kisiler
(
id int NOT NULL ,
ad varchar(255)NOT NULL,
soyad varchar(255)NOT NULL,
adres varchar(255)DEFAULT 'Adres Belirtilmedi',
il_kodu int
)
```

ALTER TABLE cümleciğinde default kısıtlayıcı tanımlaması:

```
ALTER TABLE kisiler
ALTER adres SET DEFAULT 'Adres Belirtilmedi'
```

Oluşturulan default kısıtlayıcının silinmesi ise:

```
ALTER TABLE kisiler
ALTER adres DROP DEFAULT
```

ile yapılır.

VI. **Kontrol Kısıtlayıcı (Check Constraint):** Belirtilen formata göre verilerin girilmesini sağlar. Kontrol kısıtlayıcı ile belirli tablonun sütunlarında belirli kurallar oluşturulabilir. Böylelikle veritabanına eklenecek girdileri veritabanı katmanında kontrol ederek, sağlam bir alt yapı oluşturma imkanı elde edilir. Kontrol kısıtlayıcılar ile veri bütünlüğünü iş(business) katmanında kontrol edilmeksizin, bu denetimi veritabanı(database) katmanında yapılmasını sağlanır. Örneğin, tabloda tanımlanan T.C.Kimlik Nu. alanına 11 karakterin girilmesi Kontrol Kısıtlayıcı ile sağlanabilir. Server/Client veya Web tabanlı uygulamalarda bu kontrolü client tarafında yapılması tavsiye edilir.
Örnek: Kişiler tablosunda "il_kodu" alanının istenilen formatta girilebilmesini sağlamak için:

```
CREATE TABLE kisiler
(
id int NOT NULL ,
ad varchar(255)NOT NULL,
soyad varchar(255)NOT NULL,
adres varchar(255),
il_kodu int NOT NULL,
CHECK (il_kodu>0)
)
```

Birden fazla alan için kontrol yapılmak istenildiğinde:

```
CREATE TABLE kisiler
(
id int NOT NULL ,
ad varchar(255)NOT NULL,
soyad varchar(255)NOT NULL,
adres varchar(255),
il_kodu int NOT NULL ,
CONSTRAINT chk_kisi CHECK(il_kodu>0 AND soyad LIKE 'A%')
)
```

Kişiler tablosu için ALTER TABLE cümleciği ile tanımlama:

```
ALTER TABLE kisiler
ADD CHECK (il_kodu>0)
```

Birden fazla alan için ALTER TABLE ile tanımlama:

```
ALTER TABLE kisiler
ADD CONSTRAINT chk_kisi CHECK (il_kodu>0 AND soyad LIKE 'A%')
```

Tanımlanan bir alan silinmek istendiğinde:

```
ALTER TABLE kisiler
DROP CHECK chk_kisi
```

ifadeleri kullanılmaktadır.

2. **Görünüm (View):** Sorguları basitleştirmek, sorgu sürelerini kısaltmak ve sistemin daha performanslı çalışmasını sağlamak için kullanılan sanal tablodur. Sıklıkla kullanılan sorguların view kullanarak yapılması performans açısından önemli iyileştirmelerle sonlanır. View sanal bir tablo olarak işlem yapar, ancak bu sanal tablonun daha önceden oluşturulması gerekir. Verileri daha düzenli halde görebilmemizi sağlayan view'ler, Mysql 5.0'den sonra ortaya çıktı. View'lar birer sanal tablodur. Yani bir tablo gibi davranırlar, bu yüzden oluşturulan view'lar üzerinde select işlemleri gerçekleştirilebilir. Görünümler **CREATE VIEW** ile oluşturulur. Görünümler çeşitli amaçlarla kullanılabilir.

- *Tablolardaki belli sütunların ya da satırların kullanımının sınırlandırılması.* Böylelikle, görünümler bir ya da birden çok tablonun belli bir kısmına erişimi denetlemek için kullanılabilir.
- *Karmaşık sorguların detaylarının gizlenmesi*. Veritabanı uygulamaları, karmaşık birleştirme işlemleri içeren sorgulara gerek duyuyorsa, bunlara ilişkin görünümlerin oluşturulması bu tür sorguların kullanımını basitleştirebilir.
- *Eklenen ya da güncellenen değerlerin belirli aralıklarda sınırlandırılması.*

View tanımlanmasında:

```
CREATE VIEW `view_ismi` AS SELECT ifadeler;
```

"CREATE VIEW" view_ismi " tanımlaması MySQL sunucusuna" view_ismi "adlı veritabanında bir görüntüleme nesnesi oluşturmasını söyler. "AS SELECT " ise, görünümlerde paketlenecek SQL ifadeleridir. Bir SELECT ifadesi, bir tablo veya birden çok tablodaki verileri içerebilir. Belirli bir görünümü oluşturan SQL ifadelerini görmek istiyorsak, bunu yapmak için aşağıda gösterilen komut dosyasını kullanabiliriz:

```
SHOW CREATE VİEW 'view_ismi'
```

A. Sql komutlarından view oluşturma ve faydaları:

1) **Veri Güvenliği:** Veri tabanı içinde bulunan tablolardaki bazı sütunlarda bulunan bilgilerin, herkes tarafından görülmesi istenmeyebilir. Örneğin, personelin maaşlarının herkes tarafından görünmesi istenmeyebilir. Bu durumda, Personel adlı temel (base) tablodan, personelview adlı bir view oluşturulabilir.

```
CREATE VIEW personelview
AS SELECT sicil, sos_g_no, ad, soyad, dog_tar, adres, cinsiyet, bol_no, yon_s_g_n
FROM Personel;
```

personelview adlı görünüm, herkesin kullanımına açık, Personel adlı temel (base) tablo ise, yetkili kişiler dışındakilere, erişilemez hale getirilirse, maaşların herkes tarafından erişilebilir bilgi olması önlenmiş olur. Bir view'den bilgi listelenmesi temel tablodan bilgi listelenmesinden farklı değildir. Oluşturulan view'dan tüm personele ait maaş bilgisi dışındaki bilgiler çekilmek istendiğinde aşağıdaki ifade kullanılır:

```
SELECT * FROM personelview;
```

Bir temel tablodan bir view oluşturulurken, temel tablodaki aynı sütun isimlerini kullanmak zorunlu değildir.

Örnek, Parça adlı ve parca_no, parca_ad, pr_no, fiyat ve agirlik adlı sütun (alan) isimlerini içeren tablo kullanılarak oluşturulan parcaview içinde, parca_no yerine parca_no_view, fiyat yerine ucret ve agirlik yerine kilo isimleri kullanılmıştır:

```
CREATE VIEW
parcaview (parca_no_view,ucret,kilo)
AS SELECT par_no,fiyat,ağırlık
FROM Parça;
```

2) **Sorgulamanın daha basit hale gelmesi:** Karmaşık sorgulamalarda, bazı SELECT komutlarının sonuçları diğer SELECT komutlarınca kullanıldığında, sorgulamanın düzenlenmesinde yanlışlıklar yapma olasılığı artar. Karmaşık sorgulamalar, VIEW özelliği kullanılarak daha basit hale getirilebilir. Burada temel fikir şudur: Bir view, bir sorgulama sonucu elde edilen bilgiyi (tabloyu) isimlendirerek oluşturulan bir sanal tablo ise; o halde karmaşık SELECT komutu içinde, sonucu kullanılacak başka bir SELECT komutu kullanmak yerine, bu sonucu bir view olarak isimlendirip, view adını kullanmak. Bazı durumlarda ise, çok sık olarak sorulan karmaşık soruları bir view yapısı içinde saklayarak, daha sonra aynı tip sorgulamalar için bu view yapısı kullanılarak daha basit ifadeler elde edilebilmektedir.

Örnek: Satış bölümünde çalışan personelin herhangi birinden daha düşük maaş alan ve mühendislik bölümünde çalışan kişileri listeleyiniz. (Satış bölümü kodu 2 ve mühendislik bölümü kodu ise 1 kabul ediliyor.)

```
SELECT *
FROM Personel
WHERE maas<ANY(SELECT maas
FROM Personel WHERE bol_no=2) AND
bol_no=1;
```

Aynı örneğin cevabı olan bir tablo view olarak saklanmak istenirse:

```
CREATE VIEW Personelview
AS SELECT *
FROM Personel
WHERE maas<ANY(SELECT maas
FROM Personel WHERE bol_no=2) AND
bol_no=1;
```

Tablo için view oluşturduktan sonra **"**Satış bölümünde çalışan personelin herhangi birinden daha düşük maaş alan ve mühendislik bölümünde çalışan kişileri **"** listelemek için aşağıdaki sorguyu kullanmak yeterli olacaktır:

```
SELECT * FROM Personelview ;
```

3) **Sadece view kullanılarak gerçekleşebilen sorgulamalar**: Bir tablodan elde edilecek bilgiler için, iki kademeli işlem gerektiren sorgulamalarda, ilk adımda bir view oluşturup ikinci adımda esas sorgulamayı bu view yardımı ile gerçekleştirmek, çoğu kez kaçınılmaz bir durumdur.
Örnek: Her bölümde, o bölümdeki ortalama maaştan daha yüksek maaş alanları listeleyiniz. Bu sorunun cevaplandırılması için önce her bölümdeki ortalama maaşların bulunması gereklidir.

```
CREATE VIEW bolum_ort_view(bol_no,ort,maas)
AS SELECT bol_no,AVG(maas)
FROM Personel
GROUP BY bol_no;
```

Daha sonra, yaratılan "bolum_ort_view" yardımı ile (bu view, bölüm no'ları ve bölüm ortalama maaşlarını saklamaktadır) "Her bölümde, o bölümdeki ortalama maaştan daha yüksek maaş alanları" listeleyebiliriz.

```
SELECT *
FROM Personel WHERE bol_no=bolum_ort_view.bol_no .
 AND.maas>ort_maas;
```

4) **Veri bütünlüğünün sağlanması:** View oluşturma esnasında CHECK sözcüğünün kullanılması ile, o view'i oluştururken sağlanması gereken koşulların, daha sonra view içine veri ekleme ya da değişiklik işlemlerinde de ihmal edilmesi engellenmiş olur.

Örnek: Maaşı 25000000'un üstünde olan personelden oluşan bir "kısıt_personel_view" adlı view oluşturulmsun.

```
CREATE VIEW kısıt_personel_view
 AS SELECT FROM Personel
 WHERE maas>25000000,
 WITH CHECK OPTION;
```

Oluşturulan view'e CHECK opsitonu eklendiği için, daha sonra bu view içine maaşı 13000000 olan bir personel eklenmek istendiği zaman şu hata mesajı alınacaktır:

```
Error: NOT enough nonNULL VALUES
```

Eğer CHECK opsiyonu kullanılmasaydı hata mesajı alınmadan bu veri view içine yüklenecektir.

B. **View'ler Üzerinde Ekleme, Silme, Değişiklik İşlemleri** View'ler üzerindeki ekleme, silme ve değişiklik işlemleri esas itibarı ile tablolar üzerinde yapılan benzer işlemlerden çok farklı değildir. Fakat view'ler üzerinde bu tip işlemlerin gerçekleştirilmesinde bazı kısıtlamalarda mevcuttur.

- Bir view'in güncellenebilir nitelikte olması için, bir birleştirme (join) işlemi sonucunda üretilmemiş olması gerekir. Başka bir deyişle, CREATE VIEW komutunda FROM sözcüğünü izleyen kısımda sadece tablo adı bulunmalıdır.
- View içindeki hiçbir kolon bileşik (aggregate) fonksiyonlarca üretilmiş olmamalıdır. (MAX, SUM v.b)
- View'in üretildiği SELECT komutunda DISTINCT, GROUP BY ya da HAVING sözcüklerini içeren parçaların yerine getirilmiş olmamalıdır.

Bu koşulları sağlamayan view'ler sadece okunabilir (Readonly) özellikteki view'lerdir ve üzerlerinde herhangi bir değişiklik yapılamaz.

✓ **View içine satır ekleme**: Daha önceden oluşturulmuş Personel adlı view, ad, soyad ve maas alanlarını içermiş olsun. Bu view, güncellenebilir nitelikte ise, aşağıdaki INSERT komutu ile, aynen tablolarda olduğu gibi kendisine bir satır eklemek mümkün olacaktır:

```
INSERT INTO
Personel VALUES
(&#8216;Ali&#8217;,&#8217;Çakır&#8217;,12000000);
```

Daha önceden, view oluşturulurken, CHECK OPTION alternatifi kullanılmışsa, bu takdirde, ekleme esnasında, view'i oluşturan koşul ihlal ediliyorsa, sistem eklemeye müsaade etmeyecek ve hata mesajı verecektir.

Örnek: Personel adlı tablodan, maaşı 20000000 TL'yi aşan personeli alarak, "kısıt_personel_view" adlı bir view oluşturulduğunda;

```
CREATE VIEW kısıt_personel_view
AS SELECT FROM Personel
WHERE maas>20000000
WITH CHECK OPTION;
```

Şimdi "kısıt_personel_view" view'i içine

```
INSERT INTO
kısıt_personel_view
VALUES
(37261,&#8217;34268152&#8217;,&#8217;Beril&#8217;,
&#8217;Caner&#8217;,{01/04/64},&#8217;Kadıköy&#8217;,.F.,
14000000,2,&#8217;37624158&#8217;);
```

komutu ile maaşı 14000000 olan bir kişi eklenmek istendiğinde, bu komut kabul edilmeyecek ve aşağıdaki hata mesajı alınacaktır:

```
Error:NOT enough nonNULL VALUES
```

Eğer CHECK opsiyonu kullanılmasaydı, hata mesajı verilmeksizin bu satır, view içine eklenecektir.

✓ **View içinden satır silme:** Güncellenebilir bir view içinde satır silme işlemi, tablolardan satır silme işlemi ile aynı şekilde gerçekleştirilir.

✓

Örnek: "kısıt_personel_view" view'i içinden, maaşı 2500000'den az olan kişiler silinmek istenirse;

```
DELETE FROM kısıt_personel_view WHERE maas<2500000;
```

komutunu kullanmak yeterli olacaktır.

✓ **View satırları üzerinde güncelleme işlemi:** Güncellenebilir view'lerde güncelleme işlemi tablolarda yapılan işlemler ile aynı olacaktır.

✓

Örnek: "kısıt_personel_view" adlı view'de sicili 27251 olan kişinin maaşının 37000000 olarak değiştirmek istenirse:

```
UPDATE kısıt_personel_view
SET maas=37000000
WHERE sicil=27251;
```

komutunu kullanmak uygun olacaktır.

✓ **Oluşturulan view'i silmek:** Tabloların silinmesine benzer şekilde, sistemde oluşturulan bir view, DROP VIEW komutu ile silinebilir.

```
DROP kısıt_personel_view ;
```

Bir view'in silinmesi ile, o view'e bağlı olarak oluşturulmuş diğer bütün view'ler ve bu view ile ilişkili önceliklerin de tümü silinmiş olacaktır.

C. Rapor Edilecek Sorular !

- **Aşağıdaki özellikleri içeren tabloları oluşturunuz.**

Tablo Adı: client_master

client_master

columnname	datatype	size
client_no	varchar2	6
name	varchar2	20
address1	varchar2	30
address2	varchar2	30
city	varchar2	15
state	varchar2	15
pincode	number	6
bal_due	number	10,2

Bu tabloya ilişkin veriler

Clientno	Name	city	pincode	state	bal.due
0001	Ivan	Bombay	400054	Maharashtra	15000
0002	Vandana	Madras	780001	Tamilnadu	0
0003	Pramada	Bombay	400057	Maharashtra	5000
0004	Basu	Bombay	400056	Maharashtra	0
0005	Ravi	Delhi	100001		2000
0006	Rukmini	Bombay	400050	Maharashtra	0

TabloAdı:

Product_master

Columnname	datatype	size
Product_no	varchar2	
Description	varchar2	
Profit_percent	number	
Unit_measure	varchar2	
Qty_on_hand	number	
Reoder_lvl	number	
Sell_price	number	
Cost_price	number	

Veriler:

Data for Product Master:

Product No.	Desciption	Profit %	Unit Percent	Qty measured	Reorder on hand lvl	Sell price	Cost price
P00001	1.44floppies	5	piece	100	20	525	500
P03453	Monitors	6	piece	10	3	12000	11200
P06734	Mouse	5	piece	20	5	1050	500
P07865	1.22 floppies	5	piece	100	20	525	500
P07868	Keyboards	2	piece	10	3	3150	3050
P07885	CD Drive	2.5	piece	10	3	5250	5100
P07965	540 HDD	4	piece	10	3	8400	8000
P07975	1.44 Drive	5	piece	10	3	1050	1000
P08865	1.22 Drive	5	piece	2	3	1050	1000

Tablo Adı: Salesman_Master

Columnname	**Datatype**	**Size**	**Attributes**
Salesman_no	varchar2	6	Primary key/first letter must start with 's'
Sal_name	varchar2	20	Not null
Address	varchar2		Not null
City	varchar2	20	
State	varchar2	20	
Pincode	Number	6	
Sal_amt	Number	8,2	Not null, cannot be 0
Tgt_to_get	Number	6,2	Not null, cannot be 0
Ytd_sales	Number	6,2	Not null, cannot be 0
Remarks	Varchar2	30	

Tablo adı: Sales_Order

Columnname	Datatype	Size	Attributes
S_order_no	varchar2	6	Primary/first letter must be 0
S_order_date	Date	6	Primary key reference clientno of client_master table
Client_no	Varchar2	25	
Dely_add	Varchar2	6	
Salesman_no	Varchar2	6	Foreign key references salesman_no of salesman_master table
Dely_type	Char	1	Delivery part(p)/full(f),default f
Billed_yn	Char	1	
Dely_date	Date		Can not be lessthan s_order_date
Order_status	Varchar2	10	Values ('in process';'fulfilled';back order';'canceled

Tablo Adı: Sales_Order_Details

Column	Datatype	Size	Attributes
S_order_no	Varchar2	6	Primary key/foreign key references s_order_no of sales_order
Product_no	Varchar2	6	Primary key/foreign key references product_no of product_master
Qty_order	Number	8	
Qty_disp	Number	8	
Product_rate	Number	10,2	

Tablolara girilecek veriler :

Data for sales_man master table

Salesman_ no	Salesman name	Address	City	Pin code	State	Salamt	Tgt_to_get	Ytd Sales	Rem
500001	Kiran	A/14 worli	Bom bay	400002	Mah	3000	100	50	Goo
500002	Manish	65,narim an	Bom bay	400001	Mah	3000	200	100	Goo
500003	Ravi	P-7 Bandra	Bom bay	400032	Mah	3000	200	100	Goo
500004	Ashish	A/5 Juhu	Bom bay	400044	Mah	3500	200	150	Goo

Data for salesorder table:

S_orderno	S_orderdate	Client no	Dely type	Bill yn	Salesman no	Delay date	Orderstatus
019001	12-jan-96	0001	F	N	50001	20-jan-96	Ip
019002	25-jan-96	0002	P	N	50002	27-jan-96	C
016865	18-feb-96	0003	F	Y	500003	20-feb-96	F
019003	03-apr-96	0001	F	Y	500001	07-apr-96	F
046866	20-may-96	0004	P	N	500002	22-may-96	C
010008	24-may-96	0005	F	N	500004	26-may-96	Ip

Data for sales_order_details table:

S_order no	Product no	Qty ordered	Qty disp	Product_rate
019001	P00001	4	4	525
019001	P07965	2	1	8400
019001	P07885	2	1	5250
019002	P00001	10	0	525
046865	P07868	3	3	3150
046865	P07885	10	10	5250
019003	P00001	4	4	1050
019003	P03453	2	2	1050
046866	P06734	1	1	12000
046866	P07965	1	0	8400
010008	P07975	1	0	1050
010008	P00001	10	5	525

- **Yapılacaklar:**
    1. salesman_master tablosu üzerinde tgt_to_get değeri 200 den büyük olanlar için bir view oluşturunuz.
    2. product_master tablosu üzerinde product_view isminde bir view oluşturunuz ve sütun isimlerini pro_no, desc, profit, Unit_measure, qty olacak şekilde sırasıyla değiştiriniz.
    3. product_view isimli view'den Qty_on_hand değeri '10' olan product isimlerini getiren bir sorgu yazınız
    4. sipariş tarihi 10 gün geçen siparişleri müşteri isimleri ve ürün isimleri olarak listeleyen bir sql sorgusu yazınız.
    5. sales_order tablosunu kullanarak günlük siparişleri listelemeye yarayan bir view oluşturunuz.(Bu view her çalıştırıldığında sistem tarihini alarak o güne ilişkin siparişleri listelemelidir.)

D. Kaynakça

https://www.w3resource.com/mysql/creating-table-advance/constraint.php
https://docplayer.biz.tr/69195770-Bmb202-veritabani-yonetimi-ders-7-mysql-giris-view-gorunum-index-indeks-constraints-kisitlayicilar.html
https://dev.mysql.com/doc/refman/8.0/en/create-view.html