VERİ TABANI VE YÖNETİM SİSTEMLERİ

Dr. Öğretim Üyesi

Alper Talha KARADENİZ

GENEL DERS İÇERİĞİ

1. View
2. Stored Procedure
3. Stored Procedure’lerde Parametre Kullanımı
4. Trigger

VIEW NEDİR?

SQL Server'da "View", bir veya birden fazla tablodan veri çeken SELECT sorgularına sanal bir tablo gibi davranan yapılardır. View’ler fiziksel olarak veri tutmazlar; temel olarak tanımlandıkları sorgunun çalıştırılmasıyla sonuç üretirler. Bu nedenle "sanal tablo" olarak adlandırılırlar. View'ler, özellikle tekrar eden sorguların yönetimini kolaylaştırmak ve uygulama güvenliğini artırmak amacıyla sıklıkla tercih edilir.

VIEW KULLANIM AMACI

View’lerin kullanılmasının temel nedenlerinden biri, karmaşık sorguların basitleştirilmesi ve standartlaştırılmasıdır. Bununla birlikte kullanıcıların sadece belirli alanlara erişmesini sağlamak, veri güvenliğini artırmak, uygulama kodlarının sadeleşmesini sağlamak ve veriye erişimi soyutlamak gibi önemli avantajlar da sağlar. Ayrıca, sistemdeki gerçek tablo yapıları değişse bile view üzerinden erişim devam ettirilebilir, bu da bakım kolaylığı sağlar.

VIEW OLUŞTURMA

Bir view oluşturulurken temel olarak SELECT sorgusu yazılır ve bu sorgu CREATE VIEW ifadesi ile tanımlanır. View'ler çoğunlukla tabloların belirli sütunlarına odaklanarak kullanıcıya sadece ihtiyaç duyduğu kısmı sunar. Bu yönüyle veri erişiminde özelleştirme ve soyutlama işlevi görür. Ayrıca, kullanıcıların doğrudan karmaşık sorgularla uğraşmasını engelleyerek hataların önüne geçilmesini sağlar.

```
CREATE VIEW view_adi
AS SELECT sütun1, sütun2
FROM tablo_adi
WHERE koşul;
```

Bu yapı sayesinde sık kullanılan sorgular bir kere yazılır ve daha sonra SELECT * FROM view_adi; şeklinde tekrar tekrar kullanılabilir. Bu durum, özellikle büyük projelerde kod tekrarını önler ve kodun daha okunabilir olmasını sağlar.

VIEW

Her view güncellenebilir değildir. Bir view’ün güncellenebilir olması için genellikle tek tabloya dayanması ve karmaşık işlemler (JOIN, TOP, DISTINCT, GROUP BY gibi) içermemesi gerekir. Aksi takdirde, SQL Server bu view üzerinden veri güncelleme işlemlerine izin vermez. Bu nedenle, veri güncellemesi yapılacaksa view’ün içeriği dikkatle tasarlanmalıdır.

Zamanla sistemdeki ihtiyaçlara göre view'lerde güncellemeler yapılabilir. Bu durumda mevcut view, CREATE OR ALTER VIEW veya ALTER VIEW komutlarıyla güncellenebilir. Artık kullanılmayan view’ler ise DROP VIEW komutu ile sistemden kaldırılabilir. View'lerin düzenli olarak gözden geçirilmesi, sistem performansı ve veri yönetimi açısından önemlidir.

VIEW ÖRNEKLER

```
CREATE VIEW aktif_personeller
AS SELECT ad, soyad, pozisyon
FROM personel
WHERE durum = 'aktif';
```

- Bu view, personel tablosundan sadece "aktif" durumda olan personellerin ad, soyad ve pozisyon bilgilerini çeker. Özellikle kurum içi listelerde sadece çalışmakta olan personellerin gösterilmesi için uygundur.

VIEW ÖRNEKLER

```
CREATE VIEW musteri_siparisleri
AS SELECT m.ad, m.soyad, s.urun_adi, s.siparis_tarihi
FROM musteriler m JOIN siparisler s
ON m.musteri_id = s.musteri_id;
```

- Bu görünüm, iki tabloyu birleştirerek müşterilerin yaptığı siparişleri gösterir. Her bir müşteriye ait hangi ürünleri ne zaman sipariş ettiğini takip etmek isteyen bir işletme için oldukça faydalıdır.

VIEW ÖRNEKLER

```sql
CREATE VIEW sadece_istanbul_musterileri
AS
SELECT * FROM musteriler
WHERE sehir = 'İstanbul'
WITH CHECK OPTION;
```

- Bu view, sadece İstanbul'da bulunan müşterileri gösterir. WITH CHECK OPTION ifadesi sayesinde bu görünüm üzerinden başka şehirde müşteri eklenmesi ya da mevcut kaydın şehir bilgisinin değiştirilmesi engellenir. Bu, veri tutarlılığı sağlar.

VIEW ÖRNEKLER

```
CREATE VIEW aylik_satis_ozeti
AS SELECT MONTH(siparis_tarihi)
AS ay, SUM(toplam_tutar)
AS toplam_satis
FROM siparisler GROUP BY MONTH(siparis_tarihi);
```

- Bu görünüm, her ay için yapılan satışların toplam tutarını gösterir. Özellikle satış yöneticileri veya muhasebe personeli için aylık performans analizinde kullanılır. Güncelleme yapılamaz çünkü GROUP BY içeriyor.

VIEW ÖRNEKLER

```
CREATE VIEW kullanici_bilgileri
AS
SELECT kullanici_id, ad, soyad, email
FROM kullanicilar;
```

- Bu view, kullanicilar tablosundaki bazı hassas sütunları (örneğin şifre, TC kimlik no) gizleyerek sadece gerekli alanların gösterilmesini sağlar. Özellikle kullanıcı paneli gibi alanlarda kullanılır.

VIEW ÖRNEKLER

```sql
CREATE VIEW calisan_maaslari
AS
SELECT calisan_id, ad, maas
FROM calisanlar;
```

- Bu view, yalnızca tek bir tabloya (calisanlar) dayandığı ve herhangi bir JOIN, GROUP BY vb. içermediği için güncellenebilir. Bu view üzerinden maaş güncellemesi yapılabilir:

```sql
UPDATE calisan_maaslari
SET maas = maas + 1000
WHERE calisan_id = 5;
```

VIEW ÖRNEKLER

```sql
CREATE VIEW stokta_olmayan_urunler
AS
SELECT urun_adi, kategori
FROM urunler
WHERE stok_miktari = 0;
```

- Bu view, stoğu tükenmiş ürünleri listeler. Otomatik raporlar veya stok yenileme işlemleri için sıklıkla kullanılır. Kullanıcıya sadece eksik ürünleri göstererek işlem kolaylığı sağlar.

VIEW ÖRNEKLER

```sql
CREATE VIEW yuksek_maasli_calisanlar
AS
SELECT ad, soyad, maas
FROM calisanlar
WHERE maas > (SELECT AVG(maas) FROM calisanlar);
```

- Bu görünüm, maaşı kurum ortalamasının üzerinde olan çalışanları listeler. Performans değerlendirmeleri ve terfi analizlerinde kullanılabilir. Alt sorgu içerdiğinden biraz daha gelişmiş bir yapıya sahiptir.

VIEW ÖRNEKLER

```sql
CREATE VIEW en_yeni_urunler
AS
SELECT urun_adi, eklenme_tarihi
FROM urunler
WHERE YEAR(eklenme_tarihi) = YEAR(GETDATE());
```

- Bu view, yalnızca bu yıl eklenen ürünleri listeler. ORDER BY ifadesi doğrudan view içinde kullanılamaz; ama view'den veri çekerken sıralama yapılabilir:

```sql
SELECT * FROM en_yeni_urunler
ORDER BY eklenme_tarihi DESC;
```

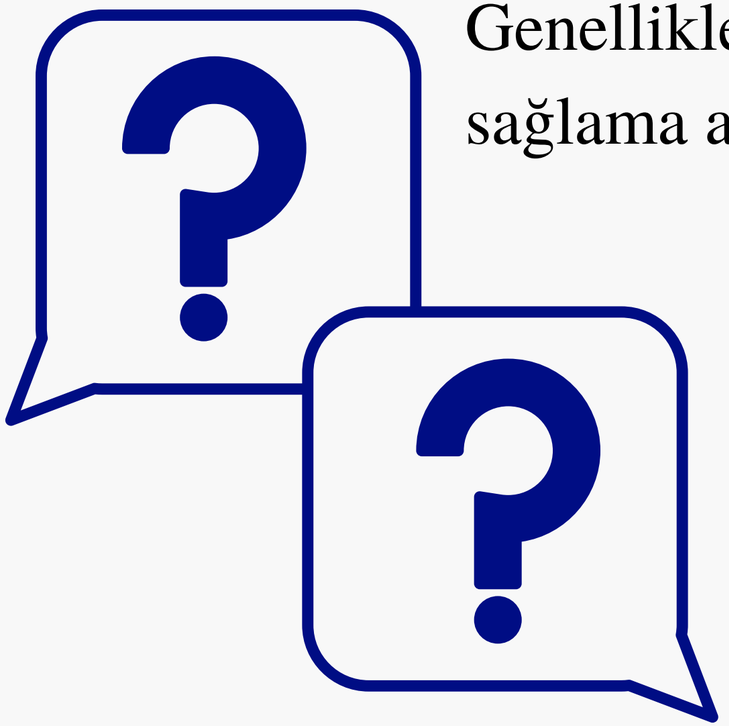
VIEW ÖRNEKLER

```sql
CREATE VIEW personel_siparis_ozeti
AS
SELECT p.ad, COUNT(s.siparis_id)
AS toplam_siparis
FROM personel p LEFT JOIN siparisler s
ON p.personel_id = s.personel_id
GROUP BY p.ad;
```

- Bu görünüm, her personelin kaç sipariş aldığını raporlamak için kullanılır. Satış performans takibi veya prim sistemleri için ideal bir yapıdır.

STORED PROCEDURE NEDİR?

Stored Procedure, önceden yazılmış ve veritabanında saklanan SQL komutları kümesidir. Uygulamalarda sık kullanılan işlemleri tekrar tekrar yazmak yerine, bu prosedür çağrılarak kullanılabilir. Genellikle performans artırma, kod tekrarını azaltma, ve güvenlik sağlama amaçlarıyla kullanılır.



STORED PROCEDURE NEDEN KULLANILIR?

Stored Procedure'ün avantajları:

- Kod tekrarını önler
- Modüler programlamayı destekler
- Veritabanında çalıştığı için ağ trafiğini azaltır
- Parametre alarak esneklik sağlar
- Yetkilendirme ve güvenlik kontrolü yapılabilir
- Karmaşık işlemleri basitleştirir

SP İÇİN KISITLAMALAR

Her nesnenin işlevi doğrultusunda bazı kurallara uyması, kısıtlarının olması gerekir. Sproc'lar da belirli kısıtlamalara tabidir.

Bir sproc şu ifadeleri içeremez:

- CREATE PROCEDURE
- CREATE DEFAULT
- CREATE RULE
- CREATE TRIGGER
- CREATE VIEW

Bir sproc yukarıdaki ifadeleri içeremez. Ancak bir başka sproc'tan ya da view, fonksiyon, tablo gibi hemen hemen her nesneden veri alabilir

STORED PROCEDURE GENEL KULLANIM

```
CREATE PROCEDURE prosedur_adi
AS BEGIN  -- SQL ifadeleri END;
```

- CREATE PROCEDURE komutu ile oluşturulur
- BEGIN ... END blokları arasında SQL komutları yer alır
- Çağırmak için:

  EXEC prosedur_adi;

STORED PROCEDURE ÖRNEKLER

```
CREATE PROCEDURE get_personel_by_id  @id
INT
AS
BEGIN
SELECT * FROM personel
WHERE personel_id = @id;
END;
```

- Bu prosedür, parametre olarak aldığı id değerine göre personel bilgilerini getirir.
- Çağırmak için:
  EXEC get_personel_by_id @id = 5;

STORED PROCEDURE ÖRNEKLER

```
CREATE PROCEDURE yeni_personel_ekle
@ad NVARCHAR(50),
@soyad NVARCHAR(50),
@maas
INT
AS
BEGIN
INSERT INTO personel (ad, soyad, maas)
VALUES (@ad, @soyad, @maas);
END;
```

Bu SP, kullanıcıdan ad, soyad ve maaş bilgilerini alarak yeni personel kaydı ekler.

STORED PROCEDURE ÖRNEKLER

```
CREATE PROCEDURE kontrol_maas  @maas
INT
AS
BEGIN
IF
@maas >= 15000
PRINT 'Yüksek maaş'
ELSE  PRINT 'Normal maaş'
END;
```

Parametre olarak verilen maaş değerine göre ekrana değerlendirme çıktısı verir.

Stored procedure içinde kontrol yapıları kullanılabilir.

STORED PROCEDURE ÖRNEKLER

```
CREATE PROCEDURE ornek_hata_yakalama
AS
BEGIN
BEGIN
TRY  -- Bilinçli hata örneği
DECLARE @x INT = 1 / 0;
END
TRY
BEGIN
CATCH
PRINT 'Hata oluştu: ' + ERROR_MESSAGE();
END
CATCH
END;
```

TRY-CATCH blokları sayesinde stored procedure içinde oluşabilecek hatalar kontrol altına alınır.

STORED PROCEDURE ÖRNEKLER

```
CREATE PROCEDURE personel_maas_guncelle
@id INT,
@yeni_maas
INT
AS
BEGIN
UPDATE personel
SET maas = @yeni_maas
WHERE personel_id = @id;
END;
```

```
CREATE PROCEDURE personel_sil  @id
INT
AS
BEGIN
DELETE
FROM personel
WHERE personel_id = @id;
END;
```

Stored procedure sadece veri eklemek için değil, veri güncelleme ve silme işlemleri için de yaygın olarak kullanılır.

STORED PROCEDURE ÖRNEKLER

```
ALTER PROCEDURE get_personel_by_id  @id
INT
AS
BEGIN
SELECT ad, soyad
FROM personel
WHERE personel_id = @id;
END;
```

```
DROP PROCEDURE get_personel_by_id;
```

- ALTER komutu ile prosedür içeriği değiştirilebilir
- DROP komutu ile veritabanından silinir

NOCOUNT OTURUM PARAMETRESİNİN KULLANIMI

NOCOUNT, bir oturum parametresidir. NOCOUNT'un kullanımı şu şekildedir:

SET NOCOUNT {ON | OFF}

SQL Server'da her işlemden sonra etkilenen kayıt sayısı hesaplamak ve mesaj geri döndürmek için kullanılır.

```
CREATE PROC pr_bilgi
AS
SET NOCOUNT OFF
SELECT ad, soyad, dogum_tarihi
FROM cocuk

EXEC pr_bilgi
```

STORED PROCEDURE İÇİN İZİNLERİ YÖNETMEK

Bir prosedür oluşturulduktan sonra kullanıcının sproc'u kullanabilmesi için izin verilmesi gerekir.

Bir sproc'u public role kapatmak;
DENY ON prosedur_ismi TO public

Bir kullanıcıya sproc'a erişim izni vermek için;
GRANT ON prosedure_ismi TO kullanici_ismi

STORED PROCEDURE'LER HAKKINDA BILGI ALMAK

SQL Server'da oluşturduğumuz Stored Procedure nesnelerinin takibi ve yönetimi için Microsoft, bazı sistem prosedürleri ve view'leri oluşturmuştur. Bu view ve prosedürleri kullanarak kendi oluşturduğumuz prosedürler hakkında bilgi alabiliriz.

Sys.Procedures kullanarak prosedürler hakkındaki bilgileri listeleyelim.

```
SELECT Name, Type, Type_Desc, Create_Date, Modify_Date
FROM Sys.Procedures;
```

Tam olarak istediğimiz prosedür ya da prosedürleri belirtmek için isimlendirmelerimize göre arama da yapabiliriz.

```
SELECT Name, Type, Type_Desc, Create_Date, Modify_Date
FROM Sys.Procedures
WHERE NAME LIKE 'p%'
```

sys.sql_modules kullanarak prosedürler hakkındaki bilgileri listeleyelim.

```
SELECT * FROM Sys.Sql_Modules;
```

STORED PROCEDUR'LERDE PARAMETRE KULLANIMI

Prosedürler hem dışarıdan parametre alabilir, hem de içerideki işlem sonucunda oluşacak bir değeri dışarıya parametre olarak gönderebilir.

Bir sproc, en fazla 1024 parametre alabilir.

Prosedürden dışarıya gönderilen parametreye döndürülen parametre (return parameter) denir.

GİRDİ PARAMETRESİ

Sproc'un işlevselliğini ve programsal yeteneklerini artırmak için kullanılan özelliklerin başında, girdi parametreleri gelir. Girdi parametre kullanımı ile, dışarıdan bir değer sproc'a parametre olarak gönderilir ve sproc içerisinde programsal olarak kullanılır.

```
CREATE PROC proje_ara(@ID nvarchar(30))
as
IF @ID IS NOT NULL
BEGIN
SELECT proje_ad,baslama_tarihi,planlanan_bitis_tarihi
from proje
where proje_no=@ID
END
EXEC proje_ara '2'
```

ÇIKIŞ PARAMETRESİ

Çıktı parametreleri, OUTPUT parametre (output parameter) olarak bilinir. Kendisine gönderilen parametreyi, içerisindeki işlem doğrultusunda değer ile doldurarak dışarıya gönderir. Bu işlem, içeriden dışarıya doğru gerçekleşir.

Prosedür, kendisini çağıran başka bir prosedüre ya da dışarıya gönderilen değeri alacak başka bir alıcıya, kendi ürettiği değeri gönderir.

RETURN İFADESİ

Çıkış parametrelerini kullanmaya gerek kalmadan Stored Procedure içerisinden değer almayı sağlar.

```
ALTER PROC demo_HesapMakinesi
(
 @sayi1 INT,
 @sayi2 INT,
 @islem SMALLINT
)
AS
IF @islem IS NOT NULL
IF(@islem = 0)
RETURN(@sayi1 + @sayi2);
ELSE IF(@islem = 1)
RETURN(@sayi1 - @sayi2);
ELSE IF(@islem = 2)
RETURN(@sayi1 * @sayi2);
ELSE IF(@islem = 3)
RETURN(@sayi1 / @sayi2);
ELSE
RETURN(0);
```

```
DECLARE @result INT;
EXEC @result=demo_HesapMakinesi 7,6,2
select @result
```

ÖRNEKLER

SORU: Personel adı girildiğinde o personele ait doğum tarihi bilgilerini gösteren SP yazınız

```
CREATE PROC per_bilgi(@ad nvarchar(30))
as
IF @ad IS NOT NULL
BEGIN
SELECT ad, dogum_tarihi
from personel
where personel.ad=@ad
END
```

- `EXEC per_bilgi 'Ahmet'`

ÖRNEKLER

SORU: Doğum yerine göre maas ortalamasını gösteren PROC yazınız.
(EXEC dy_bilgi ‘Ankara’)

```
CREATE PROC dy_bilgi (@dyer nvarchar(30))
as
IF @dyer IS NOT NULL
BEGIN
SELECT AVG(maas)
from personel,il,ilce
where personel.dogum_yeri=ilce.ilce_no and
ilce.il_no=il.il_no and il.il_ad=@dyer
END

EXEC dy_bilgi 'Ankara'
```

ÖRNEKLER

SORU: Personel numarası girildiğinde o personelin adı, soyadı ve unvan bilgilerini gösteren SPROC yazınız.

```
CREATE PROC per_unvan (@ID nvarchar(30))
as
IF @ID IS NOT NULL
BEGIN
SELECT ad,soyad,unvan_ad
from personel,unvan
where personel.unvan_no=unvan.unvan_no and personel.personel_no=@ID
END

EXEC per_unvan '1'
```

ÖRNEKLER

SORU: Personel tablosunda personel isimlerinde kayıt aramak amacıyla bir PROC oluşturunuz

```
Create Procedure Kayit_Ara
@Kayit nvarchar(30)
as
select * from personel where ad LIKE '%' + @Kayit+ '%'

EXEC Kayit_Ara 'A'
```

ÖRNEKLER

SORU: Temiz dünya projesinde çalışan personellerin adı, soyadı ve doğum tarihlerini listeleyen bir PROC yazınız.

```
CREATE PROC proje_per
as
BEGIN
SELECT personel.ad,personel.soyad,personel.dogum_tarihi
from personel,proje,gorevlendirme
where personel.personel_no=gorevlendirme.personel_no and
gorevlendirme.proje_no=proje.proje_no and proje.proje_ad='TEMİZ DÜNYA'
END


EXEC proje_per
```

ÖRNEKLER

SORU: Önceki soruda verilen PROC için Temiz dünya projesinde çalışan teknisyen olan sütunu personeller olarak değiştiren komut satını yazınız.

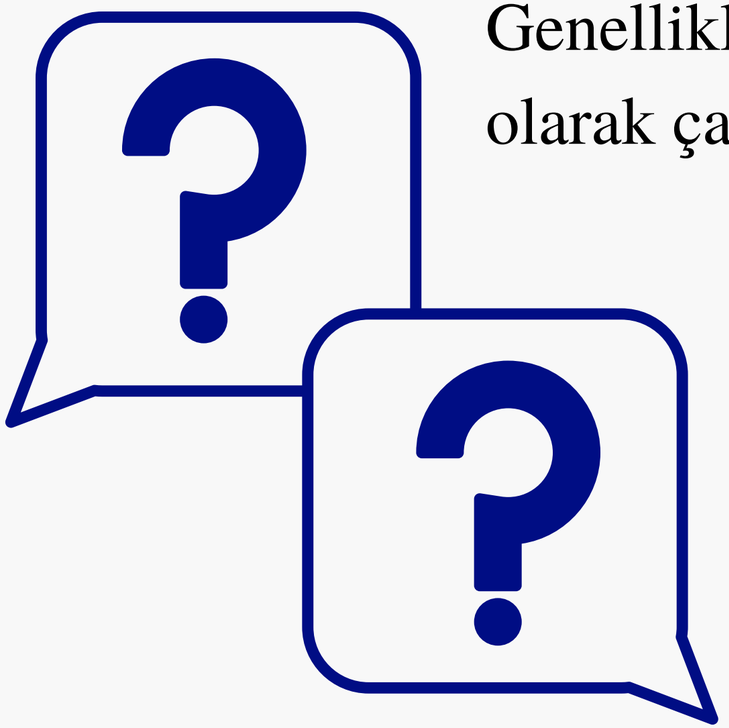
```
ALTER PROC proje_per
as
BEGIN
SELECT personel.ad,personel.soyad,personel.dogum_tarihi
from personel,proje,gorevlendirme,unvan
where personel.personel_no=gorevlendirme.personel_no and
personel.unvan_no=unvan.unvan_no and gorevlendirme.proje_no=proje.proje_no and
proje.proje_ad='TEMİZ DÜNYA' and unvan.unvan_ad='TEKNİSYEN'
END


EXEC proje_per
```

TRIGGER NEDİR?

Trigger, bir veritabanında belirli olaylar meydana geldiğinde otomatik olarak çalışan, veri ya da sistemle ilgili değişimlerde otomatik olarak tetiklenen Stored Procedure'lerdir

Genellikle INSERT, UPDATE veya DELETE işlemlerine tepki olarak çalıştırılır. (DML Tetikleyici)



Trigger’ların Amaçları

- Veri tutarlılığını sağlamak
- Otomatik kontrol mekanizmaları oluşturmak
- Günlük kayıt (log) işlemleri
- Karmaşık iş kurallarını otomatik yürütmek

TRIGGER

- Bir tablo da gerçekleşen sorgu sonucuna göre başka bir sorgunun çalışmasını sağlayan komuttur.
- Trigger’lar, veritabanında belirli işlemler gerçekleştiğinde otomatik olarak devreye giren yapılardır ve genellikle iki şekilde çalışır: AFTER veya FOR ifadesiyle tanımlanan trigger’lar, ilgili INSERT, UPDATE ya da DELETE işlemi tamamlandıktan sonra tetiklenir ve genellikle veri üzerinde işlem sonrası loglama veya denetim için kullanılır; INSTEAD OF trigger’lar ise, özellikle görünümler (VIEW) üzerinde kullanılır ve işlemler veritabanına yansıtılmadan önce devreye girerek varsayılan işlemin yerine kendi tanımlanan işlemini yürütür, böylece görünüm üzerinden dolaylı olarak veri güncelleme imkânı sağlar.

INSERT TRIGGER

```
CREATE TRIGGER ekleme
ON personel
AFTER INSERT
AS
BEGIN
SELECT 'Yeni Kayıt Eklendi';
END


INSERT INTO personel VALUES
('Ali','AYDIN','E','1995.03.25',1,'2001.03.01',1,1,40,2500,350);
```

```
CREATE TRIGGER ekleme

ON personel

AFTER INSERT

AS

BEGIN

SELECT personel.ad,personel.soyad,maas*1.2 as 'Yeni Maaş' from personel where
maas<4000

END
```

```
INSERT INTO personel VALUES
('Ali','AYDIN','E','1995.03.25',1,'2001.03.01',1,1,40,2500,350);
```

Kullanıcı bir kayıt ekledikten sonra personel tablosunu listeleyen trigger oluşturunuz

```
create trigger trg_Listele
on personel
after insert
as
begin
select * from personel
end


INSERT INTO personel VALUES
('Mehmet','Çınar','E','2000.03.25',1,'2005.03.01',1,1,40,2500,350);
```

DELETE TRIGGER

```
Create TRIGGER trg_KullaniciSil
ON personel
AFTER DELETE
AS
BEGIN
SELECT deleted.ad + ' kullanıcı adına ve ' + deleted.soyad
+ ' soyadına sahip kullanıcı silindi.' FROM deleted;
END;


DELETE FROM personel WHERE personel_no = 28;
```

SORU: 10A sınıfının kız öğrencileri silinmeyen trigger yazınız.

```
Create trigger trg_10AKizOgrenciSilme on ogrenci
after delete
as
begin
   if(exists(select * from deleted where cinsiyet='K' and sinif='10A'))
    begin
     raiserror('10A sınıfındaki kız öğrencileri silemezsiniz!!',0,0)
     rollback transaction
    end
end

delete from ogrenci where cinsiyet='K' and sinif='10A'
```

UPDATE TRIGGER

```sql
CREATE TRIGGER trg_Guncelleme
ON proje
AFTER UPDATE
AS
BEGIN
Select 'Güncelleme İşlemi Gerçekleşti'
END;

SELECT * FROM proje

UPDATE proje SET planlanan_bitis_tarihi= GETDATE() WHERE proje_ad = 'SESSİZ ORTAM'
```

```
create trigger miktarGuncelleKontrol
on siparis
for insert as
begin
declare @satisMiktar int
declare @urunid tinyint
declare @stokMiktar int
select @satisMiktar =satismiktar from siparis
select @stokMiktar=miktar from urunler where urunId in(select urunId from inserted)
if(@stokMiktar>@satisMiktar)
begin
select @satisMiktar=satisMiktar,@urunid=urunId from inserted
update urunler set miktar=miktar-@satisMiktar where urunId=@urunid
end
else
begin
rollback transaction
raiserror('girilen sipariş miktarı fazla',16,1)
end
end
```

INSTEAD OF TRIGGER

- Tablo ya da view'e fiziksel olarak veri ekleme işlemi yapılmadan önce veriyi inceleyerek, farklı bir işlem tercih edilmesi gereken durumlarda kullanılabilir.
- INSTEAD OF Trigger'lar genel olarak view üzerinden veri ekleme işlemi için kullanılır.
- Bir tablo veya view yapısında INSERT, UPDATE veya DELETE işlemlerini atlayıp, bunun yerine tetikleyici içinde tanımlanan diğer ifadeleri yerine getirmektedir.

KISACA: INSTEAD OF tetikleyicisi, belirlenen işlem gerçekleşirken devreye girer ve kendi içinde tanımlanan komutları icra etmeye başlar. Yani, belirlenen işlemin yerine geçer.

INSTEAD OF TRIGGER

```
CREATE TRIGGER ins_dene
ON personel
instead of INSERT
AS
BEGIN
Select 'Ekleme işlemi yapılabilir'
END;
```

```
INSERT INTO personel VALUES
('Ali','kan','E','1990.03.25',5,'2002.03.01',1,1,35,2500,350)
```

INSTEAD OF TRIGGER

```
CREATE TRIGGER no_silme
ON hesap
instead of delete as
begin
raiserror('Kayıt Silinemez',16,1)
rollback transaction
end

delete from hesap
```

ALTER TRIGGER

```
ALTER TRIGGER trigger_adi
ON tablo_adi
AFTER INSERT, UPDATE
AS
BEGIN
    PRINT 'Trigger güncellendi';
END;
```

DROP TRIGGER

```
DROP TRIGGER trigger_adi;
```