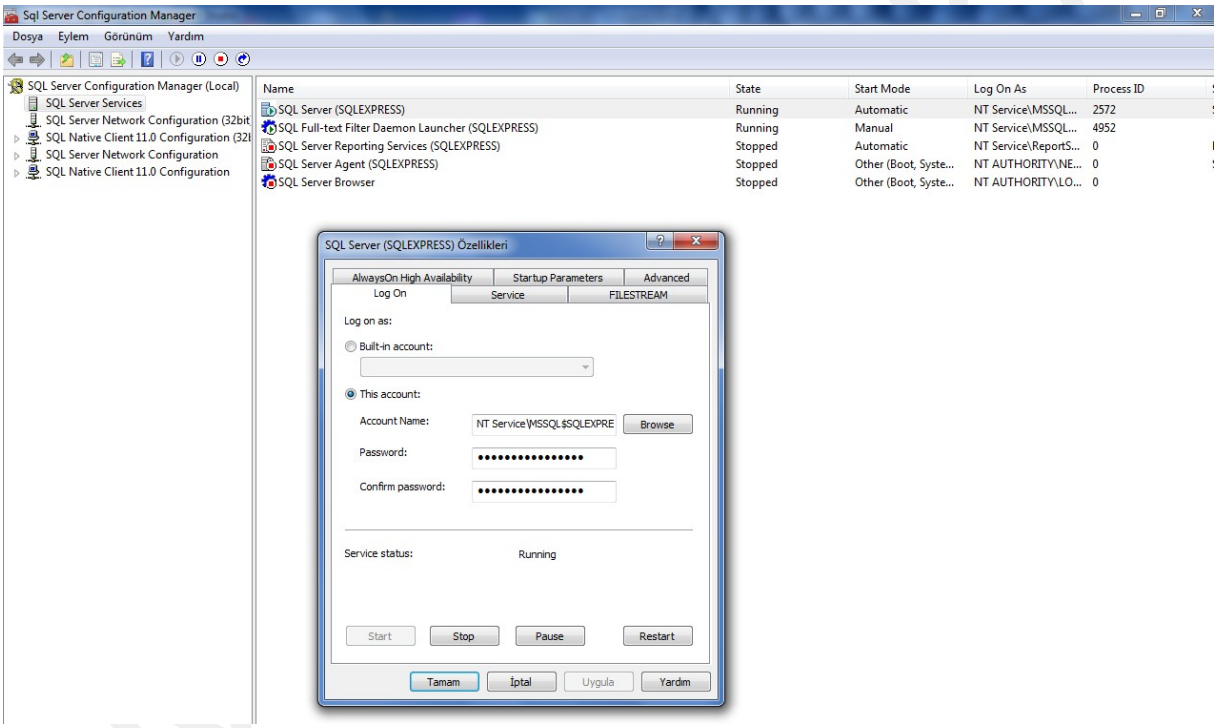
İçindekiler

1. SQL Server Araçları

1.1 SQL Server Configuration Manager

Bu araç sql servislerinin durumunu yönetmeye yarar. Hangi servisler çalışır durumdadır, istenen servislerin kapatılması ve sql server port numarası gibi ağ ayarlarının yapılmasını sağlayan araçtır. Bu ekranda görünen SQL Server hizmetinin açık olması veritabanı servisinin açık olduğu anlamına gelir. SqlServer hizmeti kapalıysa veritabanına bağlanılamaz.



1.2 Management Studio

Management Studio programı SQL Serverı yönetmek, iş zekası araçlarını kullanmayı kolaylaştıran bir araçtır. Management Studio kullanılarak tablolar tanımlanabilir, kullanıcılar yetkilendirilebilir, veritabanı yedeklenebilir ve dah bir çok veritabanı işlemi yapılabilir. Ders boyunca bu aracı kullanarak veritabanı kavramları incelenecektir. Management Studio ile kendi yazdığımız SQL kodlarının veritabanı üzerinde çalıştırılması da sağlanabilmektedir. Aşağıda programın genel bir arayüzü görülmektedir. Bu arayüzdeki tüm bölünmleri sırasıyla açılayalım.

1.2.1 Veritabanına Bağlanma

Management Studio kullanılarak veritabanına bağlanmak için aşağıdaki ekran kullanılır. Program ilk açıldığında hangi sunucuya bağlanmak istediğimizi soran bir pencere açılır. Bu pencere kullanılarak veritabanına bağlanıp işlem yapılabilir.

Eğer uzaktaki bir sunucuya bağlanarak işlem yapılmak isteniyorsa sunucunun IP adresinin girilmesi gerekir. Uzaktaki sunucuya bağlanmak için IP ile birlikte login olmak için sql server giriş türü seçilir ve kullanıcı adı ile parola girilir.

Eğer kendi bilgisayarımızda bulunan sunucuya bağlanmak istiyorsak login olmak için Windows seçeneği kullanılabilir. Bu durumda kullanıcı adı ve parola gerekmez. Zaten windows kullanıcımız sql server için yetkilidir. Kendi bilgisayarımızdaki veritabanına bağlanmak için ip yerine localhost yazılır. Nokta işareti de localhost yerine kullanılabilir. SQL Express kurulu ise bu durumda göz at diyerek SqlServerExpress instance ismi seçilir.

1.3 Neler Öğrendik?

1. Veritabanı nedir?
2. SQL Nedir?
3. İlişkisel Veritabanı Yönetim Sistemleri
4. İlişkesel Veri
5. Veritabanı yönetim sistemleri uygulamaların neresinde yer alır?
6. Neden veritabanı yönetim sistemlerini kullanırız?
7. Management Studio Kullanmak
8. SQL dili kullanılarak veritabanı oluşturmak
9. Managemenet studio kullanarak sql kod yazma ekranını açmak
10. Sql veri tipleri neler?
11. Tablo nasıl oluşturulur?
12. Sql kodu kullanarak tablo oluşturmak.
13. Temel veritabanı kavramları (Veritabanı, Şema, Tablo, Alan, Veri Tipi,Login, User, Rol, Satır, Sütun, Sonuç kümesi)
14. SQL Server kullanıcı mimarisine giriş(Login tanımlama, user tanımlama)
15. Configuration manager aracını kullanarak sql server servislerinin durumunu görüntüleme
16. Sql profiler aracını kullanarak sorguları takip etmek
17. Managemenet studio kullanarak sql servera bağlanmak

2. Temel SQL Komutları

2.1 Nesne Oluşturma Komutu – Create

Sql dilinde nesne oluşturmak için create anahtar kelimesi kullanılır. Aşağıda örnek bir tablo oluşturan komutlar görülmektedir.

```
--create anahtar kelimesi nesne oluşturmak için kullanılır
create table YedekParca
(
No int identity(1,1) primary key,
Ad nvarchar(100),
Fiyat decimal(10,2)
)
```

2.2 Nesne Düzenleme – Alter

Var olan bir veritabanı nesnesini düzenlemek için alter komutu kullanılır. YedekParca tablosuna yeni bir sütun eklenmek istenirse aşağıdaki komut kullanılır.

```
--yeniden düzenleme komutu alter komutudur
alter table YedekParca
add Tarih datetime
```

2.3 Nesneleri Kaldırmak – Drop

Veritabanında var olan bir nesne tamamen kaldırılmak istenirse drop komutu kullanılır. Bir veritabanını içindeki tüm tablolarla birlikte kaldırmak için aşağıdaki komut kullanılabilir.

```
drop database Galeri
```

Veritabanından bir tabloyu kaldırmak istiyorsak aşağıdaki komut kullanılır.

```
drop table YedekParca
```

2.4 Veri Ekleme Komutu – Insert

Veritabanında bulunan bir tabloya veri eklemek istiyorsak insert into komutunu kullanırız. Bu komutla birlikte verinin hangi tabloya ekleneceği, tablonun hangi sütunlarına veri ekleneceği belirtilir. Marka tablosuna bir kayıt ekleyen komut aşağıdaki gibidir.

```
insert into Marka(Ad)
values('Honda')
```

Yukarıdaki komut Marka tablosunun ad sütununa Honda kelimesini yeni birkayıt olarak ekler. Birden fazla sütun varsa veri ekleme işlemi aşağıdaki gibi yapılır.

```
insert into Kullanici
(Ad,Soyad,DogumTarihi,Cinsiyet,Eposta,TcKimlikNo,
Adres,EhliyetTipi,Telefon)
values('Mustafa','Yahya','1997-08-11','1',
'mustafa.yahya@deneme.com','1245678988'
,'Söğüt / Bilecik','E','0505****')
```

2.5 Veri Seçme – Select

Select anahtar kelimesi en temel sql komutlarından biridir. Tablolardaki verilerin seçilerek listelenmesi için kullanılır. Kullanici tablosundaki tüm kayıtları listeleyen temel bir select sorgusu aşağıdaki gibidir.

```
select * from Kullanici
```

Sorguda kullanılan "*" karakteri tablodaki tüm sütunların görüntüleneceği anlamına gelir. Eğer tablodaki tüm sutunlar yerine sadece istenen sütünlar listelenmek istenirse * yerine sütun isimleri belirtilmelidir. Aşağıdaki sorgu kullanıcıların yalnızda Ad, Soyad ve DoğumTarihi alanlarını listeler.

```
select Kullanici.Ad,
Kullanici.Soyad,
Kullanici.DogumTarihi
from Kullanici
```

2.6 Kayıtları Filtreleme - Where

Tablolardan tüm kayıtların değil de sadece belli şartlara uyanların listelenmesi istenirse where anahtar kelimesi kullanılır. Select sorgusunun sonuna where anahtar kelimesi eklenerek filtreleme yapılır. Aşağıdaki sorgu Ad alanı 'Mehmet' olan tüm kayıtların listelenmesini sağlar.

```
select * from Kullanici
where Kullanici.Ad='Mehmet'
```

Sadece belli sütunlar listelenmek istenirse aşağıdaki sorgu kullanılır. Aşağıdaki sorgu doğumTarihi alanı belirtilen tarihten büyük olan kayıtların ad ve soyadlarını listeler.

```
select Kullanici.Ad,Kullanici.Soyad
from Kullanici
where Kullanici.DogumTarihi>'1995-01-01'
```

2.6.1 Birden Çok Şartı Birleştirme

Where anahtar kelimesi kullanılarak birden fazla şart ifadesi birleştirilerek sorgulanabilir. Birden çok şart and veya OR operatörleriyle birleştirilerek kullanılabilir. Aşağıdaki sorguyu inceleyiniz.

```
select * from Kullanici
where Ad='Metin' and Adres='Sakarya'
```

Yukarıdaki sorgu adı mehmet olan ve adresi Sakarya olan kayıtların listelenmesini sağlar. Şartlar AND operatörü ile birleştirildiği için her iki şartı birlikte karşılayan kayıtlar listelenecektir.

Aşağıdaki sorgu birçok şartın parantezler yardımıyla birleştirilmesini göstermektedir. Böyle bir kullanımda parantezlerin doğru yerde olması istenen verinin elde edilmesi için çok önemlidir.

```
select * from Kullanici
where (Ad='Ali' and DogumTarihi>'1999-01-01') OR (Adres='Söğüt / Bilecik')
```

2.7 Veri Silme – Delete

Veri silme işlemi delete komutuyla yapılır. Delete komutu kullanılırken hangi verilerin silineceği where şart cümleciği ile belirtilir. Eğer şart belirtilmezse tablodaki tüm kayıtlar geri gelmemek üzere silinebilir. Bu sebeple delete komutu çalıştırılırken doğru şartın yazılmış olup olmadığı birkaç kez kontrol edilmelidir. Kontrol amacıyla önce silinecek kayıtları listeleyen bir select yazılabilir. Select sonucunda gelen veriler silinmek istenen veriler ise o zaman bu şartı delete için de kullanabiliriz.

Aşağıdaki sorgu No alanındaki veri 1 olan kaydın tablodan silinmesini sağlar. Belirtilen şartlara uygun kayıt yoksa herhangi bir silme işlemi yapılmaz.

```
delete from Kullanici where No='1'
```

Select komutunda olduğu gibi delete komutunda da birden fazla AND veya OR ile birleştirilebilir.

2.8 Verileri Güncellemek – Update

Update komutu verileri güncellemek için kullanılır. Bir tablodaki istenen bir verinin değiştirilmesi istendiğinde kullanılır. Update komutu kullanılırken mutlara bir şart ifadesiyle hangi verilerin güncellenmek istendiği belirtilmelidir. Eğer herhangi bir şart belirtilmezse tüm kayıtlar güncellemeden etkilenecektir.

Güncelleme yapılırken tablodaki bir veya daha fazla sütunun içeriği değiştirilebilir. Aşağıdaki komut No alanı 10 olan kaydın Ad alanını 'Yunus' olarak, eposta alanını 'yunus.colak@deneme.com' olarak değiştirir. Şarta uyan bir kayıt yoksa herhangi bir güncelleme yapılmaz.

```
update Kullanici set Ad='Yunus',Eposta='yunus.colak@deneme.com'
where Kullanici.No='10'
```

3. Veritabanı Tasarımı

Veritabanı tasarım konusu örnek bir uygulama üzerinden incelenecektir. Bu bölümde bir veritabanı tasarımı yapılırken nelere dikkat edilmesi gerektiği, hangi sıralamayla tasarım adımlarının ilerlediği irdelenecektir.

Misal:

Bir veritabanında kullanıcıların görevleri tutulmak istenmektedir. Her kullanıcı birden fazla görev alabilecek şekilde kayıtlar tutulacaktır. Sistemde kullanıcıların unvanları da saklanacaktır. Görevin veriliş tarihi, görev açıklaması, bitirme tarihi bilgileri sistemde tutulacaktır. İstendiğinde hangi kullanıcının hangi görevleri yaptığı bilgisine erişilecektir. Sistemde proje bilgileri de saklanacaktır. Birçok farklı projedeki görevler sistemden takip edilecektir. Hangi projede hangi personelin görevlendirildiği, görevlerin hangi proje kapsamında olduğu kayıt edilecektir. Böylece hangi projee için kaç iş yapılmış bilgisi elde edilmek istenmektedir. İstendiğinde hangi kullanıcının kaç görevi var, hangi projede kaç personel görev almış bilglerine ulaşılacaktır. Görevler Açık, Kapalı, Çözüldü, İptal Edildi gibi çeşitli türlerde olacak şekilde işaretlenecektir.

3.1 Nesnelerin Belirlenmesi

Sistem analiz dokümanı incelendikten yapılması gereken ilk işlem nesnelerin belirlenmesidir. Nesne, hakkında veri saklanması gereken herhangi bir şeydir. Mesela fatura bilgileri sistemden alınabilecekse fatura bir nesnedir. Sipariş bilgileri tutulması gereken bir sistemde ise sipariş bir nesne olarak adlandırılır. Veritabanında saklanması gereken her nesne bir tabloya karşılık gelir. Öncelikli adım sistemdeki bu nesnelerin neler olduğunun çıkarılmasıdır.

Nesneler belirlenirken müşteri ile toplantılar yapılarak her nesne tanımı ile neyin kat edildiği açık şekilde tanımlanır. Bu aşamada müşteri ile sistem analistinin aynı dilden konuşmaları için terim birliği önemli bir konudur. Aynı terimden müşteri de, sistem analisti de aynı şeyi anlamalıdır. Birden fazla anlama gelebilecek terimler karışıklığa sebep olmamak için açık olarak tanımlanmalıdır. Yukarıdaki örnekte görev terimiyle anlatılmak istenen, proje terimi ile anlatılmak istenen kavramların tanımlanması sistemin anlaşılırlığını arttıracaktır.

Sistem nesnelerinin belirlenmesi aşamasında müşteri ile yoğun şekilde iletişime geçilerek metinde net olarak tanımlanmayan hususların netleştirilmesi sağlanmalıdır. Yukarıdaki örnek uygulamada sistemde tutulması gereken nesneleri aşağıdaki gibi sıralayabiliriz:

Personel, Unvan, Görev, Proje, GörevDurum.x

3.2 Nesneler Arasındaki İlişkilerin Belirlenmesi

Nesneler arasındaki ilişkilerin belirlenmesi doğru veritabanı tasarımı için hayati derecede önemlidir. Yanlış ilişkiler yanlış veri yapısına ve müşteri isteklerinin karşılanmamasına sebep olabilir. Her bir nesnenin hangi nesne ile ne şekilde ilişkili olduğu ihtiyaca uygun şekilde belirlenir. Bu aşamada müşteriye sık sık soru sorularak nesneler arasındaki ilişkilerin nasıl olması gerektiği anlaşılmaya çalışılır. Yukarıdaki örnekte ilişkiler belirlenirken müşteriye aşağıdaki soruların sorulması uygun olacaktır. Çünkü bu soruların cevabı metinde yoktur ve ilişkilerin doğru belirlenmesi için bu durum önemlidir.

1. Bir görev birden fala personele verilebilir mi?
2. Bir görev aynı anda birden çok projeyle ilişkili olabilir mi?
3. Bir projede birden fazla personel görev alabilir mi?

Yukarıdaki soruların cevapları Personel – Proje, Görev-Personel, Proje-Görev ilişkilerinin nasıl olması gerektiğini belirleyecektir. Eğer bir görev birden çok personele aynı anda verilebilecekse o zaman personel ile görev tablosu çoka çok bir ilişkilendirme ile ilişkilendirilecektir. Bu ilişkilendirme işlemi üçüncü bir geçiş tablosu tanımlanarak personel ve görev tabloları ilişkilendirilecektir.

Tabloları Oluşturmak

Belirlenen her bir nesne karşılığında veritabanında bir tablo oluşturulur.

Unvan Tablosunun Oluşturulması

```
create table Unvan
(
No int identity(1,1) primary key,
Ad nvarchar(250) not null,
KisaAd nvarchar(250),
MaasCarpani decimal(4,2)
)
```

Personel Tablosunun Oluşturulması

Personel tablosuna personelin unvan bilgileri girilirken ilgili unvan numarası girilir. UnvanNo alanı Unvan tablosundaki No alanını temsil etmektedir. Bu sebeple bu alanın Unvan tablosunun No alanı ile ilişkili olduğu sql sunucusuna söylenmelidir. Bu işlemi yapmak için kısıtlayıcılar (Constraint) tanımlanır. Bir tablodaki misafir anahtar alanlar tanımlanırken Contraint anahtar kelimesi kullanılır. Personel tablosu tanımlanırken oluşturulan kısıtlayıcıya dikkat ediniz.

```
create table Personel
(
No int identity(1,1)primary key,
Ad nvarchar(max) not null,
Soyad nvarchar(max)not null,
TcKimlikNo char(11),
DogumTarihi date,
IseBaslamaTarihi date,
```

```
Cinsiyet bit,
UnvanNo int,
Constraint FK_UnvanNo Foreign Key(UnvanNo)
References Unvan(No)
)
```

Proje Tablosunun Oluşturulması

Proje tablosu oluşturulurken "her projeye bir tane sorumlu personel atanabilecektir, bir personel birden çok projdene sorumlu olabilir" cümlesine göre ilişikilendirme yapmalıyız. Bir projeye yalnız bir personel sorumlu olarak atanıyorsa proje ile personel arasında bire çok ilişki vardır. Bu ilişki tanımını SorumluPersonelNo alanı üzerinden yapıyoruz. Aşağıdaki tanıma dikkat ediniz.

```
create table Proje
(
No int identity(1,1)primary key,
Ad nvarchar(max)not null,
Aciklama nvarchar(max),
BaslamaTarihi date,
BitisTarihi date,
SorumluPersonelNo int,
Constraint FK_SorumluPersonelNo Foreign Key(SorumluPersonelNo)
References Personel(No)
)
```

Görev Tablosunun Oluşturulması

Görev tablosu proje ve GorevDurum tablolarıyla ilişkili olarak tanımlanmalıdır. Görevler ile personeller ilişkilendirilirken çoka çok bir ilişki söz konusu olduğu için ayrı bir tablo daha oluşturulacaktır. Gorev tablosunda iki alanın misafir anahtar olarak tanımlandığına dikkat ediniz.

```
create table Gorev
(
No int identity(1,1)primary key,
Ad nvarchar(max) not null,
BaslamaTarihi datetime,
BitisTarihi datetime,
GorevDurumNo int,
ProjeNo int,
Constraint FK_GorevDurumNo Foreign Key(GorevDurumNo)
References GorevDurum(No),

Constraint FK_ProjeNo Foreign Key(ProjeNo)
References Proje(No)

)
```

PersonelGorev Tablosunun Tanımlanması

Personel nesnesi ile Gorev nesnesi arasında çoka çok bir ilişki söz konusudur. Neden bu tür bir ilişkiye ihtiyacımız var? Çünkü analiz dokümanında bir görevin birden çok personele aynı anda verilebileceği söylenmiştir. Bir personele da aynı anda birden çok görev verilebileceğinden bu iki nesne çoka çok ilişki ile tanımlanmalıdır. Bu ilişki türü için üçüncü bir geçiş tablosuna ihtiyaç duyarız. Tabloda tanımlanmış olan misafir anahtar alanlarına dikkat ediniz.

```
create table PersonelGorev
(
```

```
	No int identity(1,1)primary key,
	PersonelNo int,
	GorevNo int,
	AktifMi bit,
	AtamaTarihi datetime,
	SonlanmaTarihi datetime,

	Constraint FK_PG_PersonelNo Foreign Key(PersonelNo)
	References Personel(No),

	Constraint FK_PG_GorevNo Foreign Key(GorevNo)
	References Gorev(No)
)
```

Örnek:

Emlakçı işlemleri için bir veritabanı tasarlanacaktır. Sistemde daire bilgileri tutulacaktır. Dairelerin hangi apartmanda olduğu, adresleri kayıt edilecektir. Dairelerin özellikleri de kayıt altına alınacaktır. Kombili, sobalı, ankastre gibi özellikler eklenebilecektir. Özelliklerle ilgili belli bir sayı yok kullanıcı ihtiyaç duydukça yeni özellik tanımlayabilecektir. Dairelerin oda sayısı da kayıt edilecek. Sistemde daireleri kiralayan kişi bilgileri de saklanacaktır. Hangi daire kim tarafından ne zaman kiralanmış, kaç liraya kiralanmış bilgileri kayıt edilecektir. Şu anda kiralık olan daireler, boş daireler, dolu daireler, satılık olan daire bilgileri sisteme kayıt edilecektir. Dairelerin il ilçe bilgileri de sistemde kayıt altına alınacaktır...

4. Kayıtları Filtrelemek

Daha önce temel olarak kayıtların nasıl filtrelendiğini incelemiştir. Bu bölümde biraz daha detaylı olaraka filtreleme örnekleri incelenecektir.x

Like Operatörü

```
select * from Personel
where Personel.Ad like '%me%' or Soyad like '%ki%'
```

Null Değer Kontrolü

SicilNo alanı null olmayan personel listesi

```
select * from Personel
where Personel.SicilNo is not null

SicilNo alanı null olan personel listesi
select * from Personel
where Personel.SicilNo is null
```

Aralık Sorgulama

```
select * from Unvan
where Unvan.MaasCarpani between 4 and 7
```

Tarih aralığı sorgulama

```
select * from Personel
where Personel.IseBaslamaTarihi between '2010-01-01' and '2015-01-01'
```

Dönen Kayıt Sayısını Kısıtlamak

Personel tablosunda yalnız 3 kaydın listenlemesini sağlayan sorgu:

```
select top(3) * from Personel
```

Rasgele 5 Kayıt Listeleme

```
select top(5) NEWID() as Numara, * from Personel
order by Numara
```

Metin Birleştirmek

```
select Personel.Ad + Personel.Soyad from Personel
select Personel.Ad +' ' + Personel.Soyad from Personel
```

Metin birleştirme yapılırken iki metin türünden alan + operatörü ile birleştirilebilir. Ancak sayı ve metin alanları birleştirmek için sayı alanını metin türüne dönüştürülmesi gerekir. Aşağıdaki sorguya dikkat edin.

```
select Personel.Ad +' ' + Personel.Soyad + Convert(nvarchar,No)
from Personel
```

Karakter Fonksiyonları

Büyük küçük harfe dönüştürme

```
select
Lower(Ad)as Ad,
Upper(Soyad)as Soyad,
TckimlikNo,
DogumTarihi
from Personel
```

Bir metindeki karakter sayısını veren fonksiyon

```
select Ad,Soyad,Len(Ad)as KarakterSayisi from Personel
where Len(Ad)<5
```

Tarih İşlemleri

Tarih işlemleri için aşağıda belirtilen sistem fonksiyonları kullanılabilir. Bunların dışında farklı fonksiyonlar için sistem fonksiyonlarına bakabilirsiniz.

```
select *,
Convert(nvarchar,Personel.DogumTarihi,104),
Format(Personel.DogumTarihi,'dd.MM.yyyy','tr-TR'),
YEAR(DogumTarihi) as DogumYılı,
Month(DogumTarihi) as DoğumAyı,
DAY(DogumTarihi) as DogumGunu,
Format(DogumTarihi,'M','tr-TR'),
```

```
DATEPART(dw,DogumTarihi),
DATEADD(YY,-2,DogumTarihi)
from Personel
where Personel.DogumTarihi>'1985-01-25'
```

Sistem tarihini elde etmek için GetDate() fonksiyonu kullanılır.

NULL Değer Sorgulama

NULL alanları kontrol etmek için is null operatörü kullanılır. SciliNo alanına veri girilmemiş personelleri listelemek için aşağıdaki gibi kullanılır.

```
select * from Personel
where Personel.SicilNo is null
```

NULL olmayan kayıtları seçmek için not null kullanılır.

```
select * from Personel
where Personel.SicilNo is not null
```

Kayıtları Tekil Olarak Listeleme

Kayıtları tekil olarak listelemek için distinct anahtar kelimesi kullanılır. Sorgu sonucunda tüm sütunları aynı olan, tekrar eden kayıtlar varsa o kayıtlar tek olarak listelenir.

```
select distinct Personel.Ad
from Personel
order by Personel.Ad
```

5. Alt Sorgularla Çalışmak

5.1 Tek Sonuç Döndüren Alt Sorgular

Bu türden alt sorgular çalıştığında sadece tek bir kayıt döndüren sorgulardır. Bu tür kullanımda alt sorgu sonucunda birden fazla kayıt dönmesi durumunda sorgu hata vercektir.

Her bir personelin UnvanNo alanı yerine bu unvanın adını listelemek için aşağıdaki gibi bir sorgu kullanılabilir.

```
select
Personel.Ad,
Personel.Soyad,
Personel.UnvanNo,
(select Ad from Unvan where Unvan.No=Personel.UnvanNo)as UnvanAdı
from Personel
```

Tekil sonuç döndüren alt sorgular şart cümlesi oluşturmak için de kullanılabilir. Mesela unvan ismi Tekniker olan personeli listeleyen sorgu aşağıdaki gibi kullanılabilir.

```
select * from Personel
```

```
where (select Ad from Unvan where No=Personel.UnvanNo)='Tekniker'
```

IN Operatörü

Bir alandaki veriyi bir dizi değerle karşılaştırmak istediğimizde in operatörü kullanılır. Mesela unvan numarası 1,3,5,7 değerlerinden biri olan personelleri aşağıdaki gibi listeleyebiliriz.

```
select * from Personel
where Personel.UnvanNo  in('5','7','3','1')


select * from Personel
where Personel.UnvanNo not in('5','7','3','1')
```

Çoklu Sonuç Döndüren Alt Sorgular

Çoklu sonuç döndüren bir sorgunun döndürdüğü değerler in operatörüyle kullanılabilir. PersonelGorev tablosunda kaydı bulunan yani görev atanmış personel listesini veren sorguyu inceleyiniz.

```
select * from Personel
where Personel.No  in( select PersonelNo from PersonelGorev )
```

Hiç görev atanmamış personelleri listeleyen sorgu için ise not in operatörü kullanılabilir.

```
select * from Personel
where Personel.No not in( select PersonelNo from PersonelGorev )
```

Örnek:

Herhangi bir projede proje sorumlusu olan personellerin listesini veren sorguyu alt sorgu kullanarak yazınız.

```
select * from Personel
where Personel.No in( select SorumluPersonelNo from Proje )
```

Örnek:

Bağlı bir görev bulunmayan Projelerin isimlerini listeleyen sql kodunu alt sorgu kullanarak yazınız.

```
select Proje.Ad from Proje
where Proje.No not in(select ProjeNo from Gorev)
```

Örnek:

Sistemde telefon numarası girilmemiş olan personellerin listesini veren sorguyu alt sorgu kullanarak yazınız.

```
select * from Personel
where Personel.No not in( select PersonelNo from PersonelTelefon)
```

Türetilmiş Tablolar

Bir sorgunun sonucu sanki bir tabloymuş gibi yeniden sorgulanmak istendiğinde türetilmiş tablolar kullanılabilir. Bu işlem için yazılan sorgu parantez içine alınarak bir takma isim verilir. Bu isim kullanılarak sorgulama yapılır.

```
select * from
(
	select
		Personel.Ad as PersonelAdı,
		Personel.Soyad as PersonelSoyadı,
		DogumTarihi as DogumTarihi,
		Year(DogumTarihi) as DogumYılı,
		Month(DogumTarihi)as DogduguAy
	from Personel
)
as PersonelBilgi
where PersonelBilgi.DogduguAy in('2','5','7')
```

Türetilmiş tablo bir sorgunun sonucuna bir takma ad vermek ve o takma ismi kullanarak yeniden bir selecek sorgusu yazmaktan ibarettir.

6. Tabloları Birlikte Sorgulamak

İnner Join

Left Join

Right Join

Full Join

7. Sorgu Sonuçlarını Birleştirmek -Union

Union

Union all

Personel

No	Ad	Soyad	UnvanNo
60	Abdülkadir		11
61	Ali Osman		12
62	İlker		13
63	Mustafa		14
64	Orhan		15
65	Esra		NULL

Unvan

No	Ad	KısaAd
10	Teknisyen	
11	Tekniker	
12	Mühendis	
13	İşçi	
14	Şef	
15	Müdür	
16	İş Makinası Operatörü	

No	Ad	Soyad	UnvanAdı	UnvanNo	ProjeAdı
60	Abdülkadir		Tekniker	11	Mobil Uygulama
61	Ali Osman		Mühendis	12	Web Serviz
62	İlker		İşçi	13	Personel Sistemi
63	Mustafa		Şef	14	Karar Destek Sistem
64	Orhan		Müdür	15	Ağ Yönetimi

```
select
	Personel.No,
	Personel.Ad,
	Personel.Soyad,
	Personel.UnvanNo,
	Unvan.Ad as Unvan,
	Unvan.KisaAd,
	Unvan.MaasCarpani,
	Personel.DogumTarihi,
	Proje.Ad as Proje,
	Proje.BaslamaTarihi
From Personel
inner join Unvan on Personel.UnvanNo=Unvan.No
inner join Proje on Personel.No=Proje.SorumluPersonelNo
```

Örnek

```
select
	Personel.No,
	Personel.Ad,
	Personel.Soyad,
	Personel.UnvanNo,
	Unvan.Ad as Unvan,
	Unvan.KisaAd,
	Unvan.MaasCarpani,
	Personel.DogumTarihi,
	Proje.Ad as Proje,
	Proje.BaslamaTarihi
From Personel
```

```
inner join Unvan on Personel.UnvanNo=Unvan.No
inner join Proje on Personel.No=Proje.SorumluPersonelNo
where Proje.BaslamaTarihi >'2005-01-01'
```

Örnek:

Her bir personelin adını, soyadını, personele verilen görevin adını ve görev tarihini listeleyen sorguyu yazınız.

Veritabanı Yönetim Sistemleri II

Gruplama, prosedür, fonksiyon, indeks, view, trigger…

Tabloları birlikte sorgulama bilgisinin üzerine prosedür, gruplama, trigger, view, index, sistem fonksiyonları kullanımıyla ilgili açıklama ve örnekler

8. Verileri Gruplayarak Sorgulama

1.1.Group By Deyimi Kullanımı

Verileri gruplayarak sorgulamak demek belirlenen bir alanı aynı olan kayıtları grup kabul ederek her bir grupla ilgili verileri elde etmek demektir. Mesela Urun tablosu Markaya göre gruplanarak sorgulandığında her bir marka ayrı bir grup olarak belirlenecektir. Böylece her bir markada kaç ürün bulunduğu bilgisini gruplamalı fonksiyonlar yardımıyla elde edebiliriz. Grup verileri elde edilirken grup üzerinde çalışan gruplama fonksiyonları kullanılır. En bilinen gruplama fonksiyonları aşağıdaki gibidir.

1. **Count():** Grubun eleman sayısını verir.
2. **Max(Fiyat):** Belirtilen alanın gruptaki en yüksek değerini verir.
3. **Min(Fiyat):** Belirtilen alanın gruptakin en düşük değerini gösterir.
4. **Avg(Fiyat):** Beliritilen alanın grup için ortalaması hesaplanır.
5. **Sum(Fiyat):** Belirtilen alanın toplamını verir.

Yukarıdaki fonksiyonlar oluşan her bir grup üzerinde ayrı ayrı çalışır ve sonuç üretir. Gruplama fonksiyonlarının ürettiği sonuçlar bir sütun olarak sonuç kümesinde listelenir.

Grop By deyimi kullanılmadan gruplama fonksiyonları kullanılırlırsa tüm tablo tek bir grup kabul edilerek işlem yapılır.

Misal:

Yolcu tablosunda kaç tane kayıt olduğunu listeleyen sql kodunu yazınız.

Çözüm:

Soruda herhangi bir alana göre gruplama yapılamdığı için tüm tablodaki kayıtların sayısı istenmektedir.

```
select count(*) as YolcuSayisi from Yolcu
```

Yukarıdaki örneği geliştirerek yolcuların yaş ortalamasını da hesaplamak isteyebiliriz. Yolcu sayısı ve yaş ortalamasını bize verecek olan sorguda her bir personelin yaşının hesaplanması ve Avg fonksiyonuyla ortalamasının bulunması gerekir. Önce yolcuların yaş bilgisini nasıl elde edebileceğimize bakalım. Yaş bilgisi elde etmek için doğum tarihi alanı kullanılır. Doğum tarihinin yılı şu anki yıldan çıkarılacak yaş hesaplanabilir.

```
select *,
       YEAR(Getdate())-YEAR(Yolcu.DogumTarihi) as Yaş
from Yolcu
```

Yukarıdaki sorguda yaş bilgisini elde etmek için YEAR isimli sistem fonksiyonu kullanılmıştır. YEAR fonksiyonu verilen bir tarih alanının yılını elde etmemizi sağlar. Şu anki yılı elde etmek için Getdate() fonksiyonundan gelen tarihin yılı kullanılmıştır. Getdate() fonksiyonu şu anki tarih bilgisini veren sistem fonksiyonudur.

Şimdi bu hesaplamadan yola çıkarak Yolcu tablosunda kaç kayıt olduğunu ve ortalama yaş bilgisini hesaplayabiliriz. Yolcular için yaş hesaplama işlemi Avg fonsiyonunda belirtilirse yaş ortalaması elde edilmiş olacaktır.

```
select
	count(*) as YolcuSayisi,
	Avg(YEAR(getdate())-Year(Yolcu.DogumTarihi)) as OrtalamaYas
from Yolcu
```

Gruplama fonksiyonları bir sorgude birlikte kullanılabilir. Böylece her bir fonksiyon sonucu bir sütun olarak görüntülenebilir.

```
select
	Count(*) as Sayi,
	Avg(Year(getdate())-Year(Yolcu.dogumTarihi)) as YasOrtalamasi,
	Max(Year(getdate())-year(Yolcu.DogumTarihi)) as EnYasliYolcu,
	Min(Year(getdate())-year(Yolcu.DogumTarihi)) as EnGencYolcu
from Yolcu
```

Yukarıdaki sorgularda tüm tablo tek bir grup gibi sorgulanarak bir takım veriler elde edilmiştir. Sorguda tablodaki tüm veriler tek bir grup gibi hesaplanmıştır. Bu verilerin belirtilen bir alana göre gruplanarak elde edilmesi istenebilir. Bu durumda group by deyimini kullanırız.

Mesela bilet tablosundaki veriler için bir sorgu hazırlayalım. Her bir yolcunun kaç bilet satın aldığını listelemek istiyoruz. Bu durumda yolcuya göre gruplama yapılmalıdır. Gruplama fonksiyonları her bir yolcu için ayrı ayrı çalışarak hesaplama yapmalıdır. Bunu sağlamak için group by deyimi ile bilet tablosundaki yolcu alanına göre bir gruplama yaptığımızı belirtememiz gerekir.

```
--Her bir yolcunun kaç bilet aldığını listeleyen sorgu
select
	Bilet.YolcuNo,
	Y.Ad,
	Y.Soyad,
	Count(*) as BiletSayisi
from Bilet
inner join Yolcu Y on Bilet.YolcuNo=Y.No
group by Bilet.YolcuNo, Y.Ad, Y.Soyad
```

Group by kullanıldığında group by önüne yazılan alanları aynı olan kayıtlar bir grup kabul edilir ve gruplama fonksiyonları oluşan bu gruplar üzerinde ayrı ayrı çalıştırılır.

1.2.Birden Fazla Alana Göre Gruplama Yapmak

Yukarıdaki sorguda yolcuya göre gruplama yapılırken yolcunun ad soyad alanıyla birlikte yolcu numarası da gruplamada belirtilmiştir. Bu sayede ad soyadı aynı olan farklı yolcuların farklı grup olarak değerlendirilmesi sağlanmıştır. Yukarıdaki örnekten görüldüğü gibi birden fazla alana göre gruplama yapılabilmektedir.

Eğer yolcuların bilet sayısıyla birlikte her bir yolcunun toplam kaç lire bilet parası ödediğini de listelemek istersek ücretin bulunduğu tabloyu da sogruya dahil etmeliyiz.

```
select
```

```
	Bilet.YolcuNo,
	Y.Ad,
	Y.Soyad,
	Count(*) as BiletSayisi,
	Sum(T.Ucret) as ToplamUcret
from Bilet
inner join Yolcu Y on Bilet.YolcuNo=Y.No
inner join Takvim T on Bilet.TakvimNo=T.No
group by Bilet.YolcuNo, Y.Ad, Y.Soyad
```

Bir gruplama yapıldığında sorgu sonucunda yalnızca gruplama yapılan alan veya gruplama fonksiyonlarıyla elde edilen veriler listelenebilir. Gruplamaya dahil edilmeyen bir veri sonuç kümesinde gösterilemez.

Birden fazla alana göre gruplama yapılabilir. Bu durumda belirtilen iki alanı da aynı olan kayıtlar grup kabul edilerek işlem yapılır.

Misal:

Her bir tur için kaç bilet satıldığını listeleyen bir sql sorgusu yazınız. Sorgu sonucunda turun adı ve kaç tane bilet satıldığı bilgisi bulunacaktır. Sorgu sonucunda satılan bilet sayısıyla birlikte toplam kaç liralık bilet satıldığı bilgisi de bulunacaktır.

Çözüm:

```
select
	Tur.Ad,
	Count(*) as BiletSayisi,
	Sum(T.Ucret) as ToplamUcret
from Tur
inner join Takvim T on Tur.No=T.TurNo
inner join Bilet B on T.No=B.TakvimNo
group by Tur.Ad
```

Tur tablosu bilet tablosuyla birlikte sorgulanarak bir veri listesi elde edilmiştir. Elde edilen veri listesi Tur adına göre gruplandığında gruplama fonksiyonları ile gruplardaki eleman sayıları ve toplam bilet ücretleri elde edilebilir.

Misal:

Hangi araç hangi tur için kaç kez kullanılmıştır verisini döndüren sorguyu yazınız.

```
select
	A.Plaka,
	Tur.Ad,
	Count(*) as KullanımSayisi
from Takvim T
inner join Arac A on T.AracNo=A.No
inner join Tur on T.TurNo=Tur.No
group by A.Plaka, Tur.Ad
```

Yukarıdaki sorguda gruplama işlemi arapç plakası ve tur adına göre iki alan üzerinden yapılmıştır. Bu sayede araç plakası ve tur adı aynı olan kayıtlar bir grup kabul edilerek gruplar oluşturulmuştur.

1.3.Gruplamalar üzerinde şart koşmak

Gruplamadan elde edilen verilere göre bir takım şartlar yazılabilir. Mesela 3 taneden fazla bilet satın almış yolcuların listesi gibi bir veri ihtiyacımız olabilir. Bu durumda gruplar üzerinde şart oluşturarak filtreleme yapmamız gerekir.

Misal:

Her bir gruptan kaç bilet satıldığın listeleyen bir sorgu yazmıştık. Şimdi 3 biletten daha fazla bilet satılan turların bir listesini elde etmek istiyoruz. Bu durumda group by anahtar kelimesinden sonra bir filtreleme ihtiyacı söz konusudur. Gruplama işleminden sonra gruplama üzerinde bir şart koşmak istiyorsak **having** anahtar kelimesi kullanılır. Bu anahtar kelime gruplanmış olan bir sorgunun belli şartalar uygun olan gruplarının listelenmesini sağlar.

```
select
       Tur.Ad,
       Count(*) as BiletSayisi,
       Sum(T.Ucret) as ToplamUcret
from Tur
inner join Takvim T on Tur.No=T.TurNo
inner join Bilet B on T.No=B.TakvimNo
group by Tur.Ad
having Count(*)>3 --Sum(T.Ucret)>2000
```

Yukarıdaki sorguda grup eleman sayısı 3'ten büyük olan kayıtların listelenmesi sağlanmıştır. İstenirse toplam ücret değeri belli bir değerden büyük olan kayıtlar da listelenebilir. Having ifadesi ile filtreleme yapılırken gruplama fonksiyonlarından dönen değerlere göre filtreleme yapıldığına dikkat ediniz. Normal filtreleme kelimesi olan where anahtar kelimesi group by ifadesinden sonra kullanılamaz. Where ifadesi yalnızca sorgudan hemen sonraki filtrelemeler için kullanılabilir. Eğer gruplamadan önce sadece belli şartala uyan kayıtların listelenmesi isteniyorsa where group by kelimesinden önce kullanılabilir. Mesela bilet fiyatı 200TL üzerinde olan kayıtlar alınıp gruplansın isteniyorsa aşağıdaki gibi bir sorgu kullanılır.

```
select
       Tur.Ad,
       Count(*) as BiletSayisi,
       Sum(T.Ucret) as ToplamUcret
from Tur
inner join Takvim T on Tur.No=T.TurNo
inner join Bilet B on T.No=B.TakvimNo
where T.Ucret>200
group by Tur.Ad
having Count(*)>3 --Sum(T.Ucret)>2000
```

Misal:
Her bir yolcunun hangi gezi türünde kaç bilet aldığını ve hangi gezi türüne toplam ne kadarlık ücret harcadığını listeleyen sorguyu yazınız.

Çözüm:

```
select
       Yolcu.Ad,
       Yolcu.Soyad,
       GeziTuru.Ad as GeziTürü,
       count(*) as BiletSayisi,
       Sum(Takvim.Ucret) as ToplamUcret
from Yolcu
```

```
inner join Bilet on Yolcu.No=Bilet.YolcuNo
inner join Takvim on Bilet.TakvimNo=Takvim.No
inner join Tur on Takvim.TurNo=Tur.No
inner join GeziTuru on Tur.GeziTurNo=GeziTuru.No
group by Yolcu.No,Yolcu.Ad,Yolcu.Soyad,Gezituru.Ad
order by yolcu.Ad, yolcu.Soyad
```

Gruplama işleminden sonra veriler sıralamaya tabi tutulabilir. Soruda bizden yolcu bilgileri ve gezi türleri istenmektedir. Bu sebeple yolcu bilgileri ve geti türü adına göre bir gruplama yapılmıştır. Yolcu bilgisi ve gezi türü aynı olan kayıtlar bir grup kabul edilerek gruplar oluşturulmuştur.

NOT: Veriler üzerinde belli bir alana göre gruplama yapıldığında sorgu sonucunad sadece gruplama yapılan alanlar ve gruplama fonksiyonlarının hesapladığı veriler gösterilebilir. Gruplama işlemi haricindeki alanlar listelenemez. Bu sebeple gruplama yapıldığında select ile from arasında gösterilecek sütunlar olarak group by önüne yazılan alanlar belirtilir.

1.4.Cube Deyimi Kullanımı

Gruplama ile elde edilen bir veri listesinin son satırında toplam değer elde edilmek isteniyorsa cube deyimi kullanılabilir. Aşağıdaki kod Urun tablosunu Marka adına göre gruplayarak döndürür. Bu sorguyu cube deyimi ile nasıl kullanabileceğimize bir bakalım.

```
select
Marka.Ad,
Count(*) as UrunSayisi
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
group by Marka.Ad
```

Yukarıdaki group by komutu aşağıdaki sonuç listesini döndürecektir.

Ad	UrunSayisi
Adidas	6
Arçelik	3
Asus	1
Beko	2
Nike	4
Samsung	4
Toshiba	1
Vestel	5

Yukarıdaki veri listesinde son satırda Urun sayılarının toplamı elde edilmek istenirse Cube deyimi kullanılabilir. Cube deyimi group by ile birlikte kullanılır.

```
select
Marka.Ad,
Count(*) as UrunSayisi
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
group by Cube(Marka.Ad)
```

Yukarıdaki komut çalıştırıldığında aşağıdaki liste elde edilir. Elde edilen listede son satır olarak toplam ürün sayısının verildiğini görebilirsiniz.

Ad	UrunSayisi
Adidas	6
Arçelik	3
Asus	1
Beko	2
Nike	4
Samsung	4
Toshiba	1
Vestel	5
NULL	26

Cube deyimi kullanılırak gruplama yapıldığında tek bir alana göre gruplama yapılırsa son satırda ilgili alanın toplam değeri elde edilir. Birden fazla alana göre gruplama yapıldığı durumda Cube deyiminin nasıl sonuç verdiğini inceleyelim. Bu durumda biraz daha farklı bir sonuçla karşılacağız.

Önce birden fazla kolona göre gruplama yapan sorgumuzun nasıl bir sonuç döndürdüğünü inceleyelim.

```
Select
Marka.Ad as Marka,
Reyon.Ad as Reyon,
Count(*) as UrunSayisi
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
inner join Reyon on Urun.ReyonNo=Reyon.No
group by Marka.Ad, Reyon.Ad
order by Marka.Ad
```

Marka adına ve reyon adına göre yapılan gruplama sonucunda aşağıdaki gibi bir liste dönecektir. Burada marka adı ve reyon adı aynı olan ürünler bir grup kabul edilere k sorgulama yapılmıştır.

Marka	Reyon	UrunSayisi
Adidas	Beyaz Eşya	2
Adidas	Futbol	1
Adidas	Mutfak	1
Adidas	Ofis	2
Arçelik	Akıllı Telefon	1
Arçelik	Dağcılık	1
Arçelik	Kayak	1
Asus	Elektronik	1
Asus	Ev Elektroniği	1
Beko	Beyaz Eşya	1
Beko	Mobilya	1
Beko	Oturma Grubu	1
Nike	Akıllı Telefon	1
Nike	Basketbol	1
Nike	Spor	1
Nike	TV - Görünt...	1
Samsung	Akıllı Telefon	1

Birden fazla alana göre yapılan gruplama işleminde Cube deyimi kullanıldığında gelen sonucu inceleyelim.

Marka	Reyon	UrunSayisi
NULL	Spor	1
NULL	TV - Görüntüleme	1
NULL	NULL	29
NULL	Tenis	1
Adidas	NULL	6
Adidas	Ofis	2
Adidas	Mutfak	1
Adidas	Futbol	1
Adidas	Beyaz Eşya	2
Arçelik	Akıllı Telefon	1
Arçelik	Dağcılık	1
Arçelik	Kayak	1
Arçelik	NULL	3
Asus	NULL	2
Asus	Dağcılık	1
Asus	Elektronik	1
Beko	Dağcılık	1
Beko	Beyaz Eşya	1
Beko	Mobilya	1
Beko	NULL	3
Nike	NULL	4

Yukarıdaki sonuçta görüldüğü üzere her bir markada kaç ürün olduğu bilgisi her bir markadan sonra belirtilmiştir. Ayrıca her bir reyonda kaç ürün bulunduğu bilgisi de listelenmektedir. Gruplanan veri ayrıca marka ve reyon bilgisine göre de özetlenerek sonuç kümesine yeni satırlar eklenmiştir.

1.5.Rollup Deyimi

Cube deyimi kullanılırken birden fazla alana göre gruplama yapıldığında gruplanan verilerin her alana göre ayrıca gruplanarak bize bir takım sonuçlar döndürüldüğünü görmüştük. Rollup deyimi böyle bir kullanımda daha düzenli bir sonuç döndürecektir. Yapılan gruplama işlemi içten dışa doğru toplama yapılarak listenecektir. Gruplama yapılan ilk alana göre satır arası toplamlar listeye eklenecek ve son satırda da tüm toplam değer elde edilecektir.

```
Select
Marka.Ad as Marka,
Reyon.Ad as Reyon,
Count(*) as UrunSayisi
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
inner join Reyon on Urun.ReyonNo=Reyon.No
group by Rollup(Marka.Ad,Reyon.Ad)
```

Yukarıdaki komut çalıtırıldığında aşağıdaki sonuç kümesi elde edilir.

Marka	Reyon	UrunSayisi
Beko	NULL	3
Nike	Akıllı Telefon	1
Nike	Basketbol	1
Nike	Spor	1
Nike	TV - Görün...	1
Nike	NULL	4
Samsung	Akıllı Telefon	1
Samsung	Elektronik	2
Samsung	Tenis	1
Samsung	NULL	4
Toshiba	Oturma Gr...	1
Toshiba	NULL	1
Vestel	Beyaz Eşya	2
Vestel	Dağcılık	1
Vestel	Mobilya	1
Vestel	Ses Sistem...	2
Vestel	NULL	6
NULL	NULL	29

Yukarıdaki sonuç kümesinde görüldüğü üzere ara toplamlar hesaplanmış ve listeye eklenmiştir. Ancak ara toplam değerinin bulunduğu satırlarda reyon sütünün Null olara geldiğini görmekteyiz. Bu değerin null yerine daha anlamlı bir ifade ile doldurulması sağlanabilir. Bu işlem için Grouping fonksiyonu kullanılır.

1.6.Grouping Kullanımı

Grouping ifadesi bir sütunun Rollup deyimi ile oluşturulup oluşturmadığını kontrol eder. Rollup deyimi ile oluşturulmuşsa 1 değilse 0 değerini döndürür. Bu fonksiyonun gruplama sorgumuzda kullarak reyon sütunundaki null alanlara istenen metni yazdırabiliriz.

```
Select
Marka.Ad as Marka,

Case when Grouping(Reyon.Ad)=1 Then Marka.Ad+' Ara Toplam'
       else Reyon.Ad
End as Reyon,

Count(*) as UrunSayisi
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
inner join Reyon on Urun.ReyonNo=Reyon.No
group by Rollup(Marka.Ad,Reyon.Ad)
```

Yukarıdaki sorguda reyon sütunu oluşturulurken case when ifadesi ile bir kontrol yapıldığına dikkat ediniz.

Marka	Reyon	UrunSayisi
Adidas	Ofis	2
Adidas	Adidas Ara Toplam	6
Arçelik	Akıllı Telefon	1
Arçelik	Dağcılık	1
Arçelik	Kayak	1
Arçelik	Arçelik Ara Toplam	3
Asus	Dağcılık	1
Asus	Elektronik	1
Asus	Asus Ara Toplam	2

Sorgu sonucunu incelediğimizde ara toplamların marka adıyla birlikte ara toplam olarak belirtildiğini görebiliriz.

1.7.Dinamik Sql Kodları Çalıştırma

Dinamik sql kodu demek bir stringi ifadenin sql kodu olarak yorumlanarak çalıştırlması demektir. String birleştirme yoluyla oluşturulmuş olan metin tipinde bir değişkende yazan değer sql kodu olarak yorumlanarak çalıştırılabilir. Bu tür işlemler için **Exec** fonksiyonu kullanılır. Aşağıdaki örnekte tanımlanan bir string değişkene bir sql sorgusu string olarak eklenmektedir. Daha sonra bu string tipteki değişkenin içeriğinde yazan ifade Exec fonksiyonu ile sql kodu olarak yorumlanmakta ve çalıştırılmaktadır.

```
declare @sorgu nvarchar(max)

set @sorgu='select * from '

set @sorgu= @sorgu+' Kullanici'

Exec(@sorgu)
```

Sorgu çalışıtırılmadan önce string değişkene bir metin ilavesi yapıldığını görüyoruz. Yani başlangıçta sorguda tablo adı olmadan metin yazılmıştır. Daha sonra bu metin değişkene Kullanici ifadesi eklenmiştir. Böylece kullanıcı tablosunu sorgulayacak bir sql sorgusu metni elde edilmiştir. Oluşturulan bu metnin sql komutu olarak yorumlanarak çalıştırılması için Exec fonksiyonuna gönderilmiştir.

String metinler birleştirilerek oluşturulan sql sorgularını yukarıda belirtilen şekilde çalıştırabiliriz. Dışardan kullanıcı verilerinin alındığı bir sistemde Exec fonksiyonu kullanılarak sorgu oluşturulması cidd güvenlik açıklarına sebep olabilir. Dikkatli olunması gerekir. Bu anlamda dinamik sorgu ihtiyacımız olan yerlerde kendimiz sql kodu yazacağımız zaman Exec fonksiyonunu kullanılması başka yerlerde kullanılmaması uygun olacaktır.

1.8.Pivot Operatörü Kullanımı

Satırlardan sütun oluşturmak için pivot operatörü kullanılır. Zaman zaman satırlardan gelen verileri kullarak sütun isimlerinin oluşturulması istenebilir. Satırlarla gelen verileri sütunlara dönüştürmek istiyorsak aşağıdaki gibi bir sorgu kullanılabilir.

```
with urunBilgi(UrunAdi,Marka,UrunAd)
as
```

```
(
	select Urun.Ad,Marka.Ad,Urun.Ad
	from Urun
	inner join Marka on Urun.MarkaNo=Marka.No
)
select
P.*
from urunBilgi
PIVOT
(
	Count(UrunAdi)
	For Marka in([Arçelik],[Samsung],[Nike],[Beko],[Adidas],[Vestel],[Asus])
) as P
```

Yukarıdaki sorguda urunBilgi isimli bir tablo türetilmiştir. Bu tabloda üç tane sütunun listelendiğini görmekteyiz. Amacımız türetilmiş tablo ile satır olarak gelen marka bilgilerini kullanarak sütunlar oluşturmak ve her bir marka isminin altında ilgili üründen kaç tene olduğu bilgisi görüntülemektir. Burada ürün adı alanını iki farklı sütunda da kullandığımıza dikkat ediniz. Pivot operatörü bir sütunu kullanarak sütunları oluştururken diğer satırdaki veriyi de gruplama için kullanır, üçüncü bir satırdaki veri de pivot tablonun satır bilgisini oluşturacaktır. Her bir satırda ürün adları görüntülenecek ve her bir markanın altında o üründen kaç tane olduğu bilgisi gösterilecektir. Count(urunAdi) isimli fonksiyon ürün adlarına göre gruplama yaparak sayma işlemini yapacaktır. For anahtar kelimesi ile de sütunlara bölünecek verinin hangi alan olduğu belirtilir ve sütün isimleri köşeli parantez içinde belirtilir.

Yukarıdaki sorgu aşağıdaki gibi bir sonuç döndürecektir. Sorgu sonucu incelendiğinde belirtilen sütun isimlerine göre markalardaki ürün sayıları ilgili markanın sütunu altında görüntülenmektedir.

UrunAd	Arçelik	Samsung	Nike	Beko	Adidas	Vestel	Asus
Akıllı Telefon	0	1	0	0	0	0	0
Ayakkabı	0	0	0	0	1	0	0
Bere	0	0	0	0	3	0	0
Buhar Makinesi	0	1	0	1	0	0	0
Bulaşık Makinesi	0	0	0	0	0	1	0
Buzdolabı	1	0	0	1	0	0	1
Çamaşır Makinesi	0	0	0	0	0	1	0
Eldiven	0	0	0	0	1	0	0
Fırın	2	1	0	0	0	0	0
Futbol Topu	0	0	3	0	0	0	0
Katı Meyve Sıkacağı	0	0	0	0	1	1	1
Led TV	0	1	0	0	0	0	0

1.9.Dinamik Pivot Sorgusu Oluşturma

Pivot sorguları oluşturulurken satırlar sütunlara dönüştürülür. Sütünlara gelecek olan isinler pivot sorgusu içinde elle köşeli parantez içinde belirtilir. Marka adlarına göre bir pivot yapılacaksa aşağıdaki gibi sütunlara gelecek marka adları belirtilmelidir.

```
PIVOT
(
	Count(UrunAdi)
	For Marka in([Arçelik],[Samsung],[Nike],[Beko],[Adidas],[Vestel],[Asus])
) as P
```

Yukarıdaki ifadede bulunan sütün isimlerinin elle yazılıyor olması sorun teşkil edebilir. Bu markaları elle yazmak yerine köşeli parantez içindeki bu ifadeler marka tablosundaki kayıtlardan oluşturulabilir. Yani marka tablosunda bulunan tüm marka isimleri tek bir string değişken olarak tanımlanarak bir araya getirilebilir. Oluşturulan bu string ifade sql sorgusuna eklenerek pivot sorgusunun dinamik oluşması sağlanabilir. Bu durumda pivot sorgusunun da string bir değişken olarak yazılması ve en sonunda Exec fonksiyonu ile çalıştırılması gerekir.

Marka tablosundaki kayıtların bir metin değişkeninde toplanması için bir cursor kullanılabilir. Aşağıdaki gibi bir cursor tanımıyla marka isimleri bir string değişkende toplanabilir.

```
declare @markalar nvarchar(max)
declare @markaAd nvarchar(max)

declare markaListe cursor
for
select Ad from Marka

open markaListe
fetch next from markaListe into @markaAd
set @markalar='['+@markaAd+']'

while @@FETCH_STATUS=0
Begin
	fetch next from markaListe into @markaAd
	if @@FETCH_STATUS=0
	set @markalar=@markalar+',['+@markaAd+']'
End
close markaListe
deallocate markaListe
```

Yukarıdaki sorgu ile @markalar isimli değişkende marka isimleri köşeli parantez içinde olacak şekilde elde edilir. @markalar isimli değişkenin içeriği kontrol edildiğinde aşağıdaki gibi olduğu görülecektir.

```
1   [Samsung],[Vestel],[Beko],[Asus],[Nike],[Arçelik],[Adidas],[Toshiba],[Sarar],[Toyota]
```

@markalar değişkeninin içeriğine dikkat edilirse pivot sorgusunda sütunları oluşturan sorgu yapısında olduğu anlaşılır. Bu değişkeni bir pivot sorgu metinine ilave edebiliriz. String birleştirme işlemi kullanılarak string olarak yazılan bir pivot sorgusuna bu değişken ilave edilebilir. Böylece pivot sorgusunun tamamı bir string değişken içinde oluşturulmuş olur. Oluşan bu sorgu Exec fonksiyonu ile çalıştırılarak sorgu sonucu elde edilebilir. Aşağıda string olarak yazılmış bir pivot sorgusuna @markalar değişkeni içeriğinin nasıl eklendiği görülmektedir.

```
declare @pivotSorguMetni nvarchar(max)
set @pivotSorguMetni='with urunListe(UrunAd,Marka,UrunNo)
                      as
                      (
                       select Urun.Ad,Marka.Ad,Urun.No
                      from Urun
                      inner join Marka on Urun.MarkaNo=Marka.No
                      )
                      select p.* from urunListe
                      PIVOT
                      (
                      Count(UrunNo)
                      For Marka in ('+@markalar + ')' + ') as P'
```

```
Exec(@pivotSorguMetni)
```

Yukarıdaki sorguların tamamını bir araya getirip çalıştırırsak marka adlarına göre sütünalara parçalanmış dinamik çalışan bir pivot sorgusu elde etmiş oluruz. Böylece marka isimlerini elle yazmak yerine marka tablosunda elde etmiş oluruz.

```
declare @markalar nvarchar(max)
declare @markaAd nvarchar(max)

declare markaListe cursor
for
select Ad from Marka

open markaListe
fetch next from markaListe into @markaAd
set @markalar='['+@markaAd+']'

while @@FETCH_STATUS=0
Begin
	fetch next from markaListe into @markaAd
	if @@FETCH_STATUS=0
	set @markalar=@markalar+',['+@markaAd+']'
End
close markaListe
deallocate markaListe

declare @pivotSorguMetni nvarchar(max)

set @pivotSorguMetni='with urunListe(UrunAd,Marka,UrunNo)
                      as
                      (
                      select Urun.Ad,Marka.Ad,Urun.No
                      from Urun
                      inner join Marka on Urun.MarkaNo=Marka.No
                      )
                      select p.* from urunListe
                      PIVOT
                      (
                      Count(UrunNo)
                      For Marka in ('+@markalar + ')' + ') as P'

Exec(@pivotSorguMetni)
```

Exec fonksiyonu bir string ifadenin sql komutu olarak yorumlanarak çalıştırılmasını sağlar. Dinamik sql sorguları çalıştırmak istediğimizde kullanılabilir. Yani bir sorguda tablo adının parametrelerin belli şartlara göre değişebilmesi ihtiyacı bulunan bir sorgu varsa bu sorguyu metin olarak oluşturup daha sonra da Exec fonksiyonu ile çalıştrabiliriz. Yukarıda oluşturulan string değişken içeriklerinin Exec fonksiyonu ile çalıştırıldığı görülmektedir. Dolayısıyla dinamik pivot işlem yapabilmek için Exec fonksiyonunu ne işe yaradığı ve nasıl kullanıldığının da bilinmesi gerekmektedir.

Dinamik pivot sorgusunu oluştururken türetilmiş tablolar, cursolar, döngüler ve dinamik sorgu çalıştırma özelliklerini bir arada kullandık. Dolayısıyla bu sorguya anlamak için bu konulara vakıf olunması gerektiği açıktır.

2. Bir Sorgunun Sonucunu Yeni Bir Tabloda Saklamak

Hazırlanan bir sorgu sonucunun yeni bir tabloya eklenmesi ihtiyacı söz konusu olabilir. Böyle bir durumda bir sorgunun sonucu kalıcı olarak bir tabloya eklenebilir. Bazı sorgu sonuçlarını kayıt altına alınması istene durumlarda bu işlem yapılabilir. Genellikle geçici tablolar oluşturmak için de bu yöntem tercih edilebilir.

Bir sorgunun sonucunu bir tabloda saklamak istiyorsak select ile from arasında into anahtar kelimesi ile tablo adı belirtilir. Aşağıdaki örnek uygulaması inceleyiniz.

```
select
urun.ad as Urun,marka.ad as Marka,Reyon.ad as Reyon,urun.Fiyat as Fiyat
into UrunMarkaBilgi
from Urun
inner join Reyon on Urun.ReyonNo=Reyon.no
inner join Marka on Urun.MarkaNo=Marka.No
```

Yukarıdaki sorguda sorgu sonucu döndürülen sütun isimlerinin birbirinden farklı verilmiş olduğuna dikkat edin. Eğer bir sorgunun sonucu bir tabloda saklanacaksa sorgu sonucu dönen kayıtlarda her bir sütun ismi farklı olmalıdır. Çünkü bir tabloda aynı isimli iki sütun bulunamaz. Yukarıdaki sorgu UrunMarkaBilgi isimli bir tablo oluşturur ve sorgu sonucunu bu tabloya kaydeder.

3. Rütbeleme Fonksiyonları ile Kayıtları Sıralamak

Bu fonksiyonlar satır fonksiyonları olarak da adlandırılır. Satırlara numara vermek, gruplanan bir sorguda her bir grup için satır numarasının yeniden başlamasını sağlamak gibi bir takım işlemler için bu fonksiyonlar kullanılır.

3.1.Row_Number() Fonksiyonu

Satır numarası üretmek için kullanılır. Belirtilen bir alana göre kayıtlar sıralandıktan sonra her bir kayıt için artan bir sıra numarası üretir.

```
select
ROW_NUMBER() OVER(order by Urun.Ad) as SiraNo,
Urun.Ad,marka.ad,Urun.Fiyat
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
```

Yukarıdaki sorgu ürün adına göre kayıtları sıralar ve her bir satıra artan sıralamada bir numara verir. Aşağıdaki sonucu döndürür.

SiraNo	Ad	ad	Fiyat
1	Akıllı Telefon	Samsung	2000,00
2	Ayakkabı	Adidas	500,00
3	Bere	Adidas	15,00
4	Bere	Adidas	25,00
5	Bere	Adidas	150,00
6	Buhar Makinesi	Samsung	3500,00
7	Buhar Makinesi	Beko	550,00
8	Bulaşık Makinesi	Vestel	1900,00
9	Buzdolabı	Beko	2800,00
10	Buzdolabı	Asus	2800,00
11	Buzdolabı	Arçelik	150,00
12	Çamaşır Makinesi	Vestel	2543,00

Row_Number fonksiyonu kullanılırken Over anahtar kelimesi ile birlikte kullanılır. Numaralandırmanın ne üzerinde yapılacağı Over anahtar kelimesi ile belirtilir. Mesela numaralama ürün adına göre sıralanan kayıtlara göre verilecekse yukarıdaki gibi bir kulanım yapılır. Satır numaraları belirtilen bir sütundaki verilere göre yeniden başlayacak şekilde ayarlanabilir. Bunun için parçalı satır numarası oluşturma yöntemi kullanılır.

3.2.Parçalı Satır Numarası Oluşturmak

Numaralandırmanın markaya göre parçalanması isteniyorsa aşağıdaki sorgu kullanılabilir. Markaya göre parçalamak demek numaralandırmanın her markada yeniden başlatılması anlamına gelir.

Her bir markada yeniden başlayan bir sıra numarası verilmesi isteniyorsa aşağıdaki gibi Row_Number kullanılabilir.

```
select
ROW_NUMBER() OVER(Partition By Marka.Ad order by Urun.Ad) as SiraNo,
Marka.Ad as Marka,Urun.Ad,Urun.Fiyat
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
```

Yukarıdaki sorgunun sonucu aşağıdaki gibi olacaktır.

SiraNo	Marka	Ad	Fiyat
1	Adidas	Ayakkabı	500,00
2	Adidas	Bere	15,00
3	Adidas	Bere	25,00
4	Adidas	Bere	150,00
5	Adidas	Eldiven	25,00
6	Adidas	Katı Meyve Sıkacağı	550,00
1	Arçelik	Buzdolabı	150,00
2	Arçelik	Fırın	1000,00
3	Arçelik	Fırın	850,00
1	Asus	Buzdolabı	2800,00
2	Asus	Katı Meyve Sıkacağı	500,00
1	Beko	Buhar Makinesi	550,00

Her bir markada numaralamanın yeniden başladığına dikkat ediniz.

3.2.1. Rank ve Dense_Rank Fonksiyonları

Her bir marka grubu için aynı sıra numarsı üretilmek isteniyorsu bu fonksiyonlar kullanılır. Numaralandırma mantıkları farklıdır. Rank fonksiyonu her bir marka grubuna farklı bir numara verirken o andaki satır numarasını kullanarak yeni markaya o numarayı verir. Dense_Rank fonksiyonu ise her bir marka grubuna ardışık artan sıralamada bir numara verir.

```
select
ROW_NUMBER() OVER(Partition By Marka.Ad order by Urun.Ad) as SiraNo,
RANK() OVER(order by Marka.Ad) as RankSiraNo,
DENSE_RANK() OVER(order by Marka.Ad) as DenseRankSiraNo,
Marka.Ad as Marka,Urun.Ad,Urun.Fiyat
from Urun
inner join Marka on Urun.MarkaNo=Marka.No
```

Yukarıdaki kod incelendiğinde Rank ve DenseRank fonkisoyanlarının birbirinden farklı mantıkla sıra numarası ürettiğini görebiliriz.

SiraNo	RankSiraNo	DenseRankSiraNo	Marka	Ad	Fiyat
1	1	1	Adidas	Ayakkabı	500,00
2	1	1	Adidas	Bere	15,00
3	1	1	Adidas	Bere	25,00
4	1	1	Adidas	Bere	150,00
5	1	1	Adidas	Eldiven	25,00
6	1	1	Adidas	Katı Meyve Sıkacağı	550,00
1	7	2	Arçelik	Buzdolabı	150,00
2	7	2	Arçelik	Fırın	1000,00
3	7	2	Arçelik	Fırın	850,00
1	10	3	Asus	Buzdolabı	2800,00
2	10	3	Asus	Katı Meyve Sıkacağı	500,00
1	12	4	Beko	Buhar Makinesi	550,00

Sonuç kümesi incelendiğinde Rank fonksiyonunun her bir markaya ayrı bir numara verdiğini görmekteyiz. Ancak numaralar ardışık bir şekilde artamamaktadır. Rank fonksiyonu markaya sıra numarası verirken o anki markanın bulunduğu satır numarasını kullanır. Bir sonraki markaya gelene kadar aynı numarayı kullanır. Dense_Rank fonksiyonu ise her bir markaya ardışık artan bir sıralamada numara verir.

4. Stored Prosedur (Saklı Yordam)

4.1.Stored Prosedur Nedir?

Birden fazla işlemi tek bir paket altında toplayıp bir isim vererek çağırma işlemidir. Programlama dillerindeki fonksiyon mantığına benzer. Ancak t-sql dilindeki fonksiyon başka bir yapıdır. Onunla karıştırılmamalıdır. Fonksiyonlar ayrıca incelenecektir.

Birden fazla sql sorgusu tek bir paket altına alınarak kullanılmak istendiğinde sorgu prosedür olarak tanımlanır.

Tanımlanan prosedürler prosedür belleğinde saklanır. Veritabanı yönetim sistemi kapatılıp açıldığında bellek silinir. Bu durumda prosedür ilk çağırıldığı zaman bellekte yeniden oluşturulur. Prosedür yazılan bir sql scriptinin derlenmiş olarak bellekte saklanması demektir.

4.2.Strored Prosedur Tipleri

4.2.1. CLR Strored Prosedur

CLR ortamında herhangi bir dil kullanılarak kodlanmış prosedürlerdir. Karmaşık sorguların yazılması gelişmiş programlama dili komutları kullanılarak gerçekleştirilebilir.

4.2.2. Sistem Strored Prosedurleri

Sistem üzerinde belli işleri yapmak, belli sistem nesneleri hakkında bilgi almak üzere hazır olarak sunulan prosedürlerdir. Sistem prosedürleri değiştirilemez. MS SQL sisteminde yüzlerce sistem prosedürü bulunur.

Sp_helpdb bir sistem storde prosedürüdür. İsmi belirtilen veritabanı hakkında bilgi almak için kullanılır. Market isimli veritabanı hakkında bilgi almak istiyorsak aşağıdaki gibi kullanılır.

```
sp_helpdb 'Market'
```

4.2.3. Kullanıcı Tanımlı Strored Prosedurler

Programcının sql dilini kullanarak veritabanı üzerinde oluşturduğu prosedürlerdir. Girdi parametresi olan prosedürler tanımlanabilir. Tanımlanan prosedürler üzerinde yetkilendirmeler yapılabilir. Çıktı parametreli prosedürler de tanımlanabilir.

Programcı tarafından yazılan sql scriptleri bir prosedür ismi verilerek paketlenir. Daha sonra bu isim ile çağrılarak kullanılır.

4.3.Strored Prosedur Nasıl Çalıştırılır?

Sql komutları çalıştırılırken veritabanı yönetim sistemi tarafından aşağıdaki adımlardan geçerler.

4.3.1. Ayrıştırma

Yazılan prosedürün geçerli olup olmadığının denetlenmesi aşamasıdır. Sql ifadelerinin geçeriliği denetlenir.

4.3.2. Derleme

Bir önceki aşamada elde edilen sonuç ele alınarak çalıştırma planı oluşturulur. Güvenlik kontrolleri yapılır. Kullanılacak indeksler belirlenir.

4.3.3. Çalıştırma

Çalışma planında oluşturulan plan ele alınarak işlem gerçekleştirilir. Sonuç programcıya gösterilir.

Not:

Yukarıdaki üç aşama prosedür ilk kez çalıştırılırken uygulanır. Prosedür oluşturulduktan sonra tekrar çalıştırıldığında üç aşamanın tamamı uygulanmaz. Prosedürün derlenmiş hali hafızada saklanır. Böylece tekrar çağrıldığında tüm işlemler baştan başlamaz. Bu da prosedürün daha performanslı

çalıştığı anlamına gelir. İlk iki şama atlanarak çalıştırıldığı için presedür daha hızlı cevap verecektir. Normal bir sql kodu çalıştırmaktansa prosedur oluşturmanın getirdiği fayda tam da budur.

4.4.Stored Prosedur Neden Kullanılır?

9. Çok karmaşık sorgular, peş peşe çalışması gereken sql komutları prosedür oluşturularak tanımlanabilir. Bu sayede bu işlem bloklarına ihtiyaç duyulduğunda tek yapılması gereken prosedürü ismiyle çağırmaktır.
10. Prosedürler normal scriptlere göre daha hızlı çalışır. Performans amacıyla kullanılabilir.
11. Sql server açıldığında çalışma üzere bir prosedür tanımlanabilir.
12. Güvenlik amacıyla kullanılabilirler. Bir sql kullanıcısının erişim yetkisinin olmadığı tablolara prosedür kullanılarak erişim sağlanabilir. Prosedürler parametre denilen yapılarlar dışardan veri alarak çalışabilirler. Bu durum sql injeksiyon açıklarına karşı da bir önlem sayılabilir.
13. Sistem prosedürleri kullanılarak sistem hakkında bilgi alınabilir.

4.5.Stored Prosedur Oluşturma

Prosedür tanımlanırken *Create proc* anahtar kelimesi kullanılır. *With* anahtar kelimesi ile seçenekler tanımlanabilir. Ancak *with* kelimesi zorunlu değildir. *With* kelimesi ile prosedür kodlarının şifrelenmesi gibi bir takım ek prosedür özellikleri tanımlanır. *As* anahtar kelimesinden sonra prosedürde çalıştırılacak sql komutları yazılır.

```
CREATE PROC prosedur_Adı
with
As
--prosedürün çalıştıracağı sql kodları
```

NOT: Prosedürleri bir sorgu içinde tablo gibi sorguya ekleyere kullanamayız. Ancak bir prosedürün sonucu başka bir tabloya kayıt eklemek için kullanılabilir.

Prosedürü yeniden düzenlemek ve değiştirmek için alter deyimi kullanılır.

```
ALTER proc UrunBilgiGetir
As
...sql kodları
```

4.5.1. Parametresiz Prosedür Tanımlama

Parametresiz prosedürler çalıştırılırken dışardan herhangi bir veri almayan prosedürlerdir. Prosedür çalışır ve sonuç döndürür. Prosedür ismi kullanılarak prosedür çalıştırılır ve sonuç listesi elde edilir. Prosedür peş peşe çalıştırılan farklı sorgularla birlikte çalıştırılacaksa o zaman exec prosedur_Adi şeklinde exec anahtar kelimesi kullanılarak çalıştırılır.

NOT:

Aynı isimli iki prosedür tanımlanamaz. Prosedür isimleri tekil olmalıdır. Aynı bir prosedür ikinci kez oluşturulmaya çalışıldığında (ikinci kez create proc prosedur_Adi) sistem aşağıdaki hata mesajını verir.

```
There is already an object named 'UrunReyonBilgi' in the database.
```

Örnek:

Ürün bilgilerini listeleleyen bir prosedür yazalım.

```
CREATE proc UrunBilgi
As
select
       U.Ad,U.Fiyat,M.Ad,U.ReyonNo
from Urun U
inner join Marka M on U.MarkaNo=M.No
```

Örnek:

Sipariş bilgilerini döndüren bir prosedür yazalım. Prosedür hangi markalı üründen kaç sipariş verildiğini listeleyecektir.

```
CREATE proc SiparisUrunSayi
as
Select M.Ad,count(*)as UrunSayisi
from Siparis S
inner join Urun U on S.UrunNo=U.No
inner join Marka M on U.MarkaNo=M.No
group by M.Ad
```

Örnek:

Sipariş veren kullanıcıların ner birinin adını soyadını ve sipariş verdiği ürünün adını ve fiyatını listeleyen prosedür yazalım.

```
create proc KullaniciSiparis
As
select K.Ad,K.Soyad,U.Ad,U.Fiyat
from Siparis S
inner join Musteri M on S.MusteriNo=M.No
inner join Kullanici K on M.KullaniciNo=k.No
inner join Urun U on s.UrunNo=U.No
```

Örnek:

Her bir ürünün adını, fiyatını ve bağlı olduğu reyonun adını listeleyen prosedürü yazınız.

```
create proc UrunReyonBilgi
as
Select U.Ad,U.Fiyat,R.Ad as ReyonAd
from Urun U
inner join Reyon R on R.No=U.ReyonNo
```

4.5.2. Prosedürde Değişiklik Yapmak

Oluşturulmuş bir prosedürü tekrar değiştirmek veya düzenlemek istediğimizde ALTER PROC deyimi kullanılır. Create yerine alter kullanıldığında var olan bir proseduru yeniden tanımlayabilir ve üzerinde değişiklik yapabiliriz. Üzerinde değişiklik yapılan prosedürün değişiklikten önceki haline ulaşılamaz. Bu sebeple değişiklik yapılırken dikkatli davranmak önemlidir. Yukarıdaki prosedürde Marka bilgisinin de listelenmesi istendiğinde prosedür aşağıdaki gibi güncellenebilir. Alter kelimesi kullanılarak prosedüre yeni özellikler dahil edilebildiğine dikkat ediniz. Hiç var olmayan bir prosedür alter kelimesi ile kullanılamaz. Bu durumda hata alırız.

```
ALTER proc SiparisBilgi
as
select
	K.Ad,K.Soyad,U.Ad,U.Fiyat,Marka.Ad as Marka
from Siparis S
inner join Urun U on S.UrunNo=U.No
inner join Musteri M on S.MusteriNo=M.No
inner join Kullanici K on M.KullaniciNo=K.No
inner join Marka on U.MarkaNo=Marka.No
```

Var olan bir prosedürü tamamen silmek için *drop proc* prosedur_Adı deyimi kullanılabilir. Silinen bir prosedür tekrardan geri getirilemez. Yeniden tanımlanması gerekir.

4.5.3. Strored Prosedürlerde Parametre Kullanımı

Stored prosedür çalışırken dışardar bir takım veriler alabilir. Mesela bir tabloya bir kayıt ekleyen bir prosedür tanımlanabilir. Bir tablodan sorgu yaparken bir takım şartlara göre sorgu yapılması sağlanabilir. Bir veya daha fazla parametreli prosedür tanımlanabilir. Parametre denilen şey prosedür çalışırken belirtilmesi gereken verilerdir. Bu türlü parametrelere girdi parametreleri denir.

Bir prosedür 1024 tane parametre alabilir.

NOT: Prosedure bir parametre tanımlandıysa mutlaka parametre ile birlikte çalıştırılması gerekir. Parametre tanımlanmış bir prosedürü paremetresiz çalıştırmak istediğimizde hata alırız.

4.5.4. Girdi Parametreleriyle Prosedür Tanımlama

Prosedüre girdi parametresi tanımlanırken @ karakteri kullanılır. Parametrenin adı ve veri tipi belirtilir. Birden fazla parametre tanımlanacaksa aralarına virgül koyularak tanımlanır.

Parametre aşağıdaki şekilde tanımlanır. Parametre isminden önceki @ karakterine dikkat ediniz. Parametre isminden sonra parametrenin veri tipi mutlaka belirtilmelidir.

```
@markaNumarasi int
```

Parametreli prosedür kullanılırken parametrelere tanımlandığı sıra ile veri girilmelidir. Sayı türünden parametreler yazılırken tek tırnak içinde yazılabileceği gibi doğrudan da yazılabilir. Metin türünden parametreler ise mutlata tek tırnak içinde yazılmalıdır. Aşağıdaki kullanımların tümü geçerli kullanımdır.

```
MarkaSatisSayi 'Vestel'
KullaniciSiparis 7
KullaniciSiparis '7'
```

Örnek:

Marka numarası ile çalışan bir prosedür tanımlayalım. Prosedür ilgili markaya ait ürünlerin bir listesini döndürsün.

```
CREATE proc MarkaUrunBilgi
@markaNumarasi int
as
select U.*
from Urun U
inner join Marka M on U.MarkaNo=m.No
```

```
where M.No=@markaNumarasi
```

Örnek:

Kullanıcı numarasını parametre olarak alan bir prosedür tanımlayınız. Prosedür numarası belirtilen kullanıcının siparişlerini listeleyecektir. Kullanıcının sipariş tablosundaki bilgileri ve sipariş verdiği ürün bilgileri listelenecektir.

```
CREATE proc KullaniciSiparisBilgi
@kullaniciNumarasi int
As
Select s.*,u.* From Siparis S
inner join Musteri m on s.MusteriNo=m.No
inner join kullanici k on m.KullaniciNo=k.No
inner join Urun U on s.UrunNo=U.No
where k.No = @kullaniciNumarasi
```

NOT: s.* ifadesi sipariş tablosundaki tüm sütunların listeleneceğini belirtir. U.* ifadesi ürün tablosundaki tüm sütünların listeleneceğini belirtir. Burdaki * ifadesi tablodaki tüm alanların gösterileceğini belirtir.

Parametre sayısı birden fazla olabilir. İki parametre ile çalışan bir prosedür tanımlayalım. Prosedür marka tablosunun alanlarına yeni bir kayıt ekleme işlemini yapacaktır. Prosedürler yalnızca sorgulama için değil tabloya veri eklemek için de kullanılabilir.

Örnek:

Marka tablosuna yeni bir kayıt ekleyen prosedür yazınız.

```
create proc MarkaEkle
@markaAdi nvarchar(max),@markaAdresi nvarchar(max)
as
insert into Marka(Ad,Adres)
values(@markaAdi,@markaAdresi)
```

Prosedür aşağıdaki gibi kullanılabilir.

MarkaEkle ‘Anka’,’Erzurum’

Örnek:

Numarası belirtilen ürününü fiyatını güncelleyen bir prosedür yazınız.

Çözüm:

Veriler güncellenirken mutlara birincil anahtar bir alan bilinmelidir. Hangi kaydın güncelleneceği bu alanlara bakılarak belirlenir. Belle şartlar dahilinde güncelleme yapılır. Bu sebeple güncelleme için kullanıcak filtreleme alanları mutlaka belirtilmelidir. Örneğimizde filtreleme amaçlı olarak urun numarası kullanılmaktadır.

NOT: Prosedür parametreleri verilirken, parametrelerin tanımlandığı sıraya göre verilir. Mesela ilk parametre marka adı, ikinci parametre marka adresi olarak tanımlandıysa, prosedürü çağırırken ilk girilen değer markaAd parametresine, ikinci girilen değer markaAdres paramertresine alınır.

```
create proc UrunFiyatGuncelle
@urunNumarasi int, @fiyat money
```

```
as
update Urun set fiyat=@fiyat
where Urun.No=@urunNumarasi
```

Prosedürü aşağıdaki gibi kullanabiliriz.

UrunFiyatGuncelle 54, 245.55

Örnek:

Parametre olarak adı belirtilen markadan kaç tane ürün satıldığını listeleyen sql kodunu yazınız. (Marka adı parametre olarak prosedüre verilecektir.)

```
create proc MarkaSatisSayi
@markaAdi nvarchar(max)
as
select
       M.Ad,
       count(*) as SatisSayisi
from Siparis S
inner join Urun U on s.UrunNo=U.No
inner join Marka M on U.MarkaNo=M.No
where m.Ad=@markaAdi
group by M.Ad
```

MarkaSatisSayi 'Beko'

Örnek:

Belirtilen bir marka numarasından yine numarası belirtilen bir reyonda kaç tane bulunduğunu listeleyen bir prosedür yazınız. Prosedür marka numarasını ve reyon numarasını parametre olarak alacaktır. Belirtilen reyonda belirtilen markadan kaç tane ürün bulunduğu listelenecektir.

Çözüm:

```
create proc MarkaReyonBilgi
@markaNumarasi int, @reyonNumarasi int
as
select M.Ad,count(R.No) from Marka M
inner join Urun U on M.No=U.MarkaNo
left join Reyon R on U.ReyonNo=R.No
where M.No=@markaNumarasi and R.No=@reyonNumarasi
group by M.Ad
```

Yukarıda tanımlanmış olan prosedürü kullanmak için aşağıdaki gibi bir komut çalıştırılabilir.

```
markaReyonBilgi 7, 11
```

Örnek:

Sipariş bilgilerinde arama yapacak bir prosedür tanımlayınız. Prosedür parametre olarak aldığı bir metni sipariş veren kullanıcı ad, soyad alanında, urun ad alanında, marka ad alanında arayacak ve uygun sonuçları listeleyecektir. Sonuç kümesi olarak sipariş verilen ürün adı, kullanıcı ad soyadı, marka adı bilgileri listelenecektir.

```
create proc Ara
@arananMetin nvarchar(max)
as
select
	S.Tarih,
	U.Ad as Urun,
	U.fiyat,
	k.Ad as KullanıcıAdı,
	k.Soyad,
	M.Ad as Marka
from Siparis S
inner join Urun U on S.UrunNo=U.No
inner join Marka M on U.MarkaNo=M.No
inner join Musteri MU on S.MusteriNo=MU.No
inner join Kullanici k on mu.KullaniciNo=k.No
where k.Ad like'%'+@arananMetin+'%' or
	k.soyad like '%'+@arananMetin+'%'or
	u.Ad like '%'+@arananMetin+'%' or
	M.ad like '%'+@arananMetin+'%'
```

4.5.5. Tablo Tipinde Parametre Alan Stored Prosedürler

Tablo tipinde bir değişken prosedürlerde parametre olarak kullanılabilir. T-sql dilinde değişken tanımlar gibi tablo tipli değişkenler de oluşturulabilir. Değişken tanımlamayı hatırlayalım. Sql değişken tanımlama için ***declare*** anahtar kelimesi kulanılır.

Tablo tipinde değişken tanımlama özelliği 2008 ve sonraki sürümlerde geçerlidir. Daha eski sürümlerde tanımlanamaz.

```
declare @sayi int -- tamsayı tipinde değişken
declare @arananMetin nvarchar(max)--metin tipinde değişken
declare @tarih date -- tarih tipinde değişken
```

Değişken tanımlanırken "@" işareti değişkenin isminin önüne koyulmalıdır. Bir değişkene değer atamak için aşağıdaki komut kullanılabilir.

```
declare @sayi int

select @sayi=count(*)
from Siparis
```

@sayi isimli bir değişken tanımlanmıştır. @sayi değişkenine siparis tablosundaki kayıt sayısı hesaplanıp aktarılmıştır. Değişkene doğrudan değer atanbilir.

```
select @sayi=45 -- değişkene select ile değer atama
set @sayi=105--değişkene set anahtar kelimesi ile değer atama
```

Temel veri tipleri türünde değişken tanımlanabildiği gibi tablo tipinde değişkenler de tanımlanabilir. Tablo tipindeki değişkenlere tabloları gibi veri ekleme ve sorgulama yapılabilir. Tanımlanan tablo tipinde değişken tanımlandıktan sonra prosedüre parametre olarak verilir. Aşağıda liste isimli bir tablo tipinde değişken tanımı görülmektedir.

```
declare @liste table
(
	No int,
```

```
	Ad nvarchar(max),
	KayitZamani datetime
)
```

Veritabanında mevcut bulunan tablolar tipinde bir tablo değişkeni de kolayca tanımlanabilir. Mesela urun tablosunun tüm alanlarını barındıran tablo tipinde bir değişken tanımlamak istersek aşağıdaki kod yapısı kullanılabilir.

```
declare @urunListe  as dbo.Urun;

Tablo tipinde parametre alan prosedür tanımlanırken parametre yukarıdaki şekilde tanımlanır.
```

Yukarıdaki tanımlama @urunListe isimli bir tablo tipli değişken oluşturur. Değişkenimiz ürün tablosunun alanlarını barındırır. Tanımlanan tablo tipli değişkeni prosedüre parametre olarak tanımlamak için aşağıdaki kod bloğu kullanılır. Prosedürde kullanılacak tablo tipindeki parametre yeni bir tip olarak tanımlanmalıdır. Aşağıdaki örnekte MarkaListesi isimli tablo tipinde bir tür tanımlanmıştır.

```
create type dbo.MarkaListesi as table
(
	No int,
	Ad nvarchar(max)
)

create proc MarkaUrunGetir
(@urunListe as dbo.MarkaListesi)
As
Select m.* from Marka M
inner join @urunListe u on M.No=u.MarkaNo
```

Prosedürlere parametre olarak sabit bir sayı göndermek yerine bir veri listesini parametre olarak göndermek istediğimizde tablo tipinde parametre alan prosedür tanımlarız.
Yukarıdaki prosedür tanımyla parametre olarak gönderilen tablo prosedür içindeki sorguda normal bir tabloymuş gibi sorguya eklenebilmektedir.

4.5.6. Çıktı Parametreli Prosedür Kullanmak

Prosedürün ürettiği bir sonucu prosedür dışındaki bir değişkene aktarmak istiyorsak çıktı parametreli prosedürler kodlanabilir. Prosedür içinde hesaplanan bir veri çıktı parametresi sayesinde prosedür dışındaki bir değişkene alınabilir. Bu işlem için OUTPUT parametre tanımlanır. Output olarak işaretlenen parametre hesaplanan sonucun aktarılacağı değişkeni belirtir.

```
create proc OrtalamaHesapla
(
@vize int,
@final int,
@ortalama decimal(6,2) output
)
as
select @ortalama=(@vize+@final)/2
```

Çıktı parametreli prosedür tanımlarken çıktı parametresi output anahtar kelimesiyle işaretlenir. Prosedür kullanılırken de çıktı parametresi olacak değişken out anahtar kelimesiyle belirtilir.

```
declare @sonuc decimal(6,2)
exec OrtalamaHesapla 45,76,@sonuc out
```

Prosedür eğer bir dizi sorgu ile birlikte kullanılıyorsa exec parametresi ile kullanılmalıdır. Çıktı parametreli prosedürler kullanılmadan önce çıktı olarak kullanılacak değişken tanımlanmalıdır. Yukarıdaki örnekte @sonuc isimli çıktı değişkeni kullanılarak prosedürün ürettiği sonuç dışardaki @sonuc değişkenine aktarılmıştır. Çıktı paramtetresinin türü hangi tipte ise prosedür kullanılırken aktarılan değişkenin türü de aynı olmalıdır.

Örnek:

Marka numarasını parametre olarak alan bir prosedür tanımlayalım. Prosedür ilgili markalı satılan ürünlerin toplam fiyatını hesaplayıp çıktı parametresiyle dışardaki bir değişkene aktaracaktır.

```
create proc SatisToplami
@MarkaNumarasi int,@sonuc decimal(15,2) output
As
select @sonuc=Sum(U.Fiyat)
from Siparis S
inner join Urun u on s.UrunNo=U.No
where U.MarkaNo=@MarkaNumarasi
```

Prosedür parametrelerinden birisi output anahtar kelimesiyle işaretlenmiştir. Bu parametre ile prosedüre, hesapladığı sonucu hangi değişkene aktaracağı söylenecektir. Prosedür aşağıdaki gibi kullanılarak hesaplanan sonuç prosedür dışındaki bir değişkene aktarılabilir.

```
declare @toplamSatis decimal(15,2)
select @toplamSatis --prosedür çalıştırılmadan önce değişkenin değeri boştur

exec SatisToplami 5,@toplamSatis out
select @toplamSatis --prosedür çalıştırıldığında değişkenin değeri dolar
```

Çıktı parametreli prosedür çalıştırıldığında hesapladığı değeri @toplamSatis isimli değişkene aktaracaktır. Böylece prosedür içinde hesaplanmış olan bir değer dışarda bir değişkene alınarak kullanılabilecektir. Prosedürler bir sorgu dizisinin parçası olarak çalıştırılacaksa `exec` anahtar kelimesi ile çağrılması gerekir. Yukarıdaki örnekte prosedür çağrılmadan önce bir değişken tanımlanmış ve ardından bu değişken ile bir proesedür çağırılmıştır. Bu sebeple prosedür exec anahtar kelimesi ile çalıştırılmıştır. Prosedürü tek başına çağırmak istersek exec zorunlu değildir.
Çıktı parametreli prosedürler genellikle bir dizi sorgu ile birlikte kullanılır. Aşağıda böyle bir örnek kullanım görülmektedir.

```
declare @toplamSatis decimal(15,2)

exec SatisToplami 3,@toplamSatis out

select
       m.ad,
       @toplamSatis
from Marka M where m.No=3
```

Yukarıdaki komut ile prosedürümüzün hesapladığı sonuç değişkene aktarılmış daha sonra bu değişken başka bir sorgu içinde kullanılmıştır.

5. Fonksiyon Tanımlama

Fonksiyonlar prosedürlere benzer yapılardır. Ancak bir takım farklı özellikler barındırırlar. Fonksiyonlar bir sorgu içinde kullabilirler. Prosedürler ise tek başına kullanılırlar, bir sorgunun parçası olarak kullanılamazlar. Tablo döndüren fonksiyonlar bir sorguda sanki bir tabloymuş gibi sorguya ilave edilebilirler. Fonksiyonların kullanım amaçlarını aşağıdaki gibi sıralayabiliriz.

4. Bir sorgunun ürettiği sonucu başka bir sorguda birleştirerek kullanma ihtiyacı
5. Bir takım programcının ihtiyacı olan hesaplamaları yapmak üzere fonksiyon tanımlanabilir. Bir ortalama hesaplama, sistemdeki tablolar üzerinde bir takım işlemler yapma vb.
6. Karmaşık sorguların daha sade şekilde yazılmasına imkan tanır.

Sistem fonksiyonları sql serverda tanımlı olan fonksiyonlardır. Bu fonksiyonları değiştiremeyiz ancak bir sogu içinde bu fonksiyonları kullanabiliriz. Fonksiyonları sorgu içinde kullanırken kaç argümanı tanımlı ise o kadar argüman yazılarak kullanılmalıdır. Parametresi olan bir fonksiyonu parametresinin değerini yazmadan kullanamayız. Fonksiyonun parametresi olmasa bile parametre parantezleri yazılmak zorundadır. Aşağıda birkaç sistem fonksiyonunun sorgu içinde kullanımı görülmektedir.

```
select getdate()

select
       m.Ad,
       getdate() as SuAnkiTarih,
       len(m.ad) as AdKarakterSayis,
       SUBSTRING(m.ad,2,4)

from Marka m
```

Yukarıda görüldüğü gibi ***getdate()*** fonksiyonu parametresiz olarak kullanılırken ***len(m.ad)*** fonksiyonu parametre verilerek çağırılmıştır. Her fonksiyonun çağrıldığı yere döndürdüğü bir sonuç değeri bulunur. Fonsiyon çalıştırıldığında tanımına uygun sonuç değerini çağrıldığı yere döndürür. Yukarıdaki örnekte ***len*** fonksiyonu tamsayı sonuç döndürürken, ***getdate*** fonksiyonu ***date*** tipinde birsonuç, ***substring*** fonksiyonu ise ***nvarchar*** tipinde bir sonuç döndürür.

Tanımlanan bir fonksiyon içinde başka bir fonksiyon çağrılabilir.

Fonksiyonlar veriler üzerinde düzenleme yapmaktan ziyade verileri sorgulamak için kullanılır. Veri ekleme, silme, güncelleme işlemleri için stored procedure tanımlanması gerekir.

Fonksiyon bloğu içinde çeşitli sorgu ifadeleri, değişken tanımlamaları, if - else blokları kullanılabilir. Fonksiyon içinde pek çok sorgu çalıştırılabilir. Peş peşe birden fazla sorgu çalıştarılabilir.

5.1. Skaler Değer Döndüren Fonksiyonlar (Tek bir değer döndüren fonksiyon)

Tekil değer döndüren fonksiyonlar çalıştıklarında tek bir sonuç değeri döndürür. Tek bir tam sayı, tarih veya metin tipinde bir sonuç döndüren fonksiyonlardır. Tekil değer döndüren fonksiyonlar management studioda ***programability*** sekmesi altında ***function*** klasöründe **skaler valued function** seçeneği altında görülebilir.

5.1.1. Skaler Fonksiyon Tanımlama

Fonksiyon tanımlanırken fonksiyonun **argüman parantezleri mutlaka koyulmalıdır**. Herhangi bir parametre tanımı olmasa bile fonksiyon tanımlarken argüman parantezleri boş olarak yazılır.

Fonksiyon tanımlanırken argüman parantezlerinden sonra fonksiyonun geri dönüş tipi tanımlanır. Fonksiyon çalıştığnda ne türden bir veri döndüreceği burada ***RETURNS*** anahtar kelimesi ile belirtilmelidir.

Fonksiyonun işlem kodları begin end bloğu arasına yazılmalıdır. ***Begin*** ve ***end*** anahtar kelimelerinin rası fonksiyon bloğu olarak adlandırılabilir. Begin end bloğu arasında fonksiyon hesaplamalarını yaptıktan sonra mutlara bir değer döndürmelidir. Geri dönüş tipinde belirtilen türde bir değişken veya bir değer mutlaka fonksiyon bloğu içinde ***RETURN*** anahtar kelimesiyle döndürülmelidir.

Aşağıda bir müşterinin kaç tane ürün sipariş verdiğini döndüren bir fonksiyon tanımlanmıştır. Kullanıcı numarasını parametre olarak alan ve kullanıcının kaç sipariş verdiğini döndüren fonksiyonu tanımlayalım.

```
create function KullaniciSiparisSayi
(@kullaniciNumarasi int)
Returns int
Begin
	declare @siparisSayisi int
	select @siparisSayisi = count(*)
	from Siparis S
	inner join Musteri M on S.MusteriNo=M.No
	where M.No=@kullaniciNumarasi

	return @siparisSayisi
End
```

Fonksiyon tanımlanırken geri dönüş tipinin Returns anahtar kelimesi ile belirtildiğine dikkat ediniz. Fonksiyon için tanımlanan bir değişkenin değer doldurularak döndürüldüğüne dikkat ediniz. Yukarıda tanımlanmış olan fonksiyon sorgu içinde aşağıdaki şekillerde kullanılabilir. Fonksiyonun parametrisini yazmak zorunludur. Parametre tanımlanmış bir fonksiyon parametre olmadan çağrılamaz.

5.1.1.1. Skaler Fonksiyonun Doğrudan Çalıştırılması

Fonksiyon doğrudan adı yazılarak bir parametre ile çağrılabilir. Parametre değeri sabit bir tam sayı değer olara verildiğine dikkat ediniz. Aşağıdaki fonksiyon çalıştırıldığında bir tamsayı değer döndürür.

```
select  dbo.KullaniciSiparisSayi(7)
```

5.1.1.2. Skaler Fonksiyonun Sorgu İçinde Yeni Bir Kolon Üretmek için Kullanılması

Fonksiyon bir sorgu içinde çalıştıralibilir. Fonksiyon her bir kayıt için ayrı ayrı çalışın ve ürettiği sonuç bir sütunda gösterilir.

```
select
	K.ad,
	k.Soyad,
	dbo.kullanicisiparissayi(k.no) as SiparisSayisi
from Kullanici K
```

Yukarıdaki sorguda fonksiyon her bir kullanıcı için ayrı ayrı çalıştırılır ve hesaplanan sonuç yeni bir sütun olarak sorgu sonucuna ilave edilir. Böyle bir kullanımda fonksiyon kullanıcı sayısı kadar tekrar tekrar çalıştırılır.

5.1.1.3. Skaler Fonksiyonun Filtreleme Şartı Olarak Kullanılması

Fonksiyon bir sorgu içinde bir şartı belirtmek için de kullanılabilir. Where şartında fonksiyonun sonuçu filtreleme amaçlı kullanılabilir.

```
select
       K.ad,
       k.Soyad,
       dbo.kullanicisiparissayi(k.no) as SiparisSayisi
from Kullanici K
where dbo.kullanicisiparissayi(k.no)>2
```

NOT: Fonksiyon ismi ile çağrılırken şema adı belirtilmelidir. Varsayılan olarak tüm fonksiyonlar dbo ismli şema altında bulunur. Fonksiyon çağrılırken şema ismi belirtilmelidir. **dbo.kullanicisiparissayi(k.no)>2**

5.1.2. Tablo Döndüren Fonksiyonlar

Fonksiyon bir sorgu sonucunda elde edilen verileri bir liste olarak döndürebilir. Fonksiyondan gelen sonuç bir sorgu içine aktarılabilir ve tablo gibi davranabilir. Tablo döndüren fonksiyonlar sorguların sadeleşmesini sağlar.

Tablo döndüren fonksiyonlar viewlere benzerler. Ancak fonksiyonlar dışardan parametre alarak çalışabilir. Viewler ise dışardan parametre alarak çalışamaz.

Tablo döndüren bir fonksiyon bir sorgunun sonucun return anahtar kelimesi ile doğrudan döndürülmesi şeklinde tanımlanır. Aşağıda örnek bir fonksiyon tanımı görülmektedir.

```
create function SiparisBilgilListesi
(@markaNumarasi int)
Returns table
as

RETURN
select
       U.ad,
       U.Fiyat,
       U.MarkaNo,
       U.ReyonNo
from Siparis S
inner join Urun U on s.UrunNo=u.No
where U.MarkaNo=@markaNumarasi
```

Oluşturulan fonksiyon sanki bir tablo gibi sorgulanabilir.

```
select a.Ad,a.fiyat,a.reyonno,
       r.Ad
from dbo.SiparisBilgilListesi(5) a
inner join Reyon R on a.reyonno=R.No
```

5.1.2.1. Tablo Döndüren Fonksiyon Tanımlama ve Kullanma

Örnek:

Siparişlere ait tüm detay bilgilerini döndüren bir fonksiyon yazalım. Fonksiyon herhangi bir parametre almadan çalışacak ve her bir siparişin, marka bilgilerini, müşteri bilgilerini, personel bilgilerini, ürün bilgilerini, reyon bilgilerini listeleyecektir.

Çözüm:

```
create function SiparisDetayBilgi
()
Returns table
as
Return
select
       S.Tarih,
       K.ad,
       K.Soyad,
       U.ad as UrunAdı,
       Marka.Ad as Marka,
       R.Ad as ReyonAdı,
       U.Fiyat,
       R.No as ReyonNo,
       Marka.No as MarkaNo,
       U.no UrunNo,
       S.MusteriNo,
       k.No as MusteriKullaniciNo,
       kp.No as PersonelKullaniciNo,
       kp.Ad as PersonelAdı,
       kp.Soyad as PersonelSoyadı

from Siparis S
inner join Urun U on S.UrunNo=U.No
inner join Musteri M on S.MusteriNo=M.No
inner join Kullanici k on m.KullaniciNo=K.No
inner join Marka on U.MarkaNo=Marka.no
inner join Reyon R on U.ReyonNo=R.No
inner join Personel P on S.PersonelNo=P.No
inner join Kullanici KP on P.KullaniciNo=kp.No
```

5.1.2.1.1. Tablo Döndüren Fonksiyonun Sorguda Kullanılması

Yukarıdaki fonksiyon siparişe ait tüm bilgileri tablo gibi döndürecektir. Fonksiyon bir tablo gibi davranır ve tablo gibi sorgulanabilir. Tablo döndüren fonksiyonun sonucu seleck ile sorgulanabilir. Fonksiyon sorgulanırken aynen tablolar gibi bir takma ad verilerek sorguda temsil edilebilir. Aşağıdaki örnekte fonksiyonun döndürdüğü sonuç ***bilgi*** olarak adlandırılmıştır.

```
select
       bilgi.UrunAdı,
       bilgi.Tarih,
       bilgi.Ad,
       bilgi.Soyad,
       bilgi.Marka
from dbo.SiparisDetayBilgi() as bilgi
```

Yukarıdaki sorgu siparişlere ait istenen bilgi alanlarının listelenmesini sağlar. Görüldüğü gibi siparişelere ait marka ve müşteri bilgileri lisetelenebilmesine rağmen sorguda bu tablolar

görülmemektedir. Sorgu fonksiyonu çağırıp çalıştırmış ve fonksiyonu döndürdüğü sonuç içinden istenen sütünların görüntülenmesini sağlamıştır.

Yukarıdaki fonksiyon kullanılarak filtreli sorgular da yazılabilir. Fonksiyon bir tablo gibi kullanılarak şart cümleciği yazılabilir.

```
select
	bilgi.UrunAdı,
	bilgi.Tarih,
	bilgi.Ad,
	bilgi.Soyad,
	bilgi.Marka
from dbo.SiparisDetayBilgi() as bilgi
where bilgi.MarkaNo=5
```

Tanımlanan bir fonksiyonu başka tablolar ile birleştirerek de sorgulayabiliriz. Fonksiyon sanki bir tablo gibi inner join ile kullanılarak sorguya eklenebilir.

```
select
	y.YorumMetni,
	bilgi.UrunAdı,
	bilgi.Fiyat,
	bilgi.Ad,
	bilgi.Soyad
from Yorum Y
inner join dbo.SiparisDetayBilgi() as bilgi on Y.UrunNo=bilgi.UrunNo
```

Fonksiyon başka tablolarla birleştirilerek sorgulanırken fonksiyona takma ad verilmelidir. Tekil değer döndüren fonksiyonlar tablo gibi sorgulanmaz. Tablo gibi kullanılabilen fonksiyonlar tablo döndüren fonksiyonlardır.

Haftaya buraya kadar olan konularla ilgili soru sorulacak ve çözüm yapılacaktır. Verilen cevaplar incelenecek artı eksi verilecektir.

Örnek:

Kullanıcı no parametresi ile çalışan bir fonksiyon tanımlayınız. Fonksiyon numarası belirtilen kullanıcının kaç sipariş verdiğini listeleyecektir. Fonksiyon tanımlandıktan sonra örnek bir sorgu içinde nasıl kullanılabilir?

Çözüm:

```
create function KullaniciUrunSayisi
(@kullaniciNumrasi int)
returns int
Begin

	declare @urunSayisi int

	select @urunSayisi=Count(*)
	from Siparis S
	inner join Musteri M on S.MusteriNo=M.No
	inner join Kullanici K on m.KullaniciNo=K.No
	where K.No=@kullaniciNumrasi

	return @urunSayisi
```

```
End
```

Tanımlanan fonksiyonun bir sorgu içinde kullanımı aşağıdaki gibidir.

```
select
       Kullanici.Ad,
       Kullanici.Soyad ,
       dbo.KullaniciUrunSayisi(Kullanici.No)

from Kullanici
where dbo.KullaniciUrunSayisi(Kullanici.No) > 1
```

NOT: Skaler değer döndüren fonksiyon (tek bir sonuç döndüren fonksiyon) bir sorgu içinde kolon oluşturacak şekilde kullanılabilir. Skaler fonksiyon bir sorgu içinde filtreleme amaçlı kullanılabilir.

Örnek:

Her bir ürünün adını, fiyatını, markasını ve reyon bilgisini döndüren bir fonksiyon tanımlayınız.

Çözüm:

Bu fonksiyon tablo döndüren bir fonksiyon olacaktır. Herhangi bir sorgunun sonucunu tabloymuş gibi döndüren bir fonksiyon tanımlanabilir.

```
create function UrunBilgileri
()
returns table
AS
RETURN
Select
       U.Ad as UrunAdi,
       U.Fiyat as UrunFiyat,
       M.ad as Marka,
       R.Ad as Reyon

from Urun U
inner join Reyon R on U.ReyonNo=R.No
inner join Marka M on U.MarkaNo=M.No
```

Yukarıda tanımlanan tablo döndüren fonksiyon aşağıdaki gibi kullanılabilir. Fonksiyon tablo döndüren bir fonksiyon olduğu için aynen bir tablo gibi sorgulanabildiğine dikkat ediniz.

```
select
       ub.*,
       s.Tarih,
       s.No as SiparisNo
from dbo.UrunBilgileri() ub
inner join Siparis s on s.UrunNo=ub.UrunNo
```

Örnek:

Numarası verilen bir urune hangi kullanıcıların hangi yorumu yazdığını listeleyen bir fonksiyon yazınız. Fonksiyon çalıştığında yorum yazan kullanıcının adı, soyadı ve yazdığı yorum metni, yorum tarihi listelenecektir. İlgili ürüne yapılan tüm yorumlar listelenecektir.

Çözüm:

```
create function UrunYorum
(@urunNumara int)
returns table
as
RETURN
select k.Ad,k.Soyad,y.YorumMetni,y.Tarih
from Yorum y
inner join Kullanici k on y.KullaniciNo=k.No
where y.UrunNo=@urunNumara
```

5.2.Tablo Tipinde Sonuç Döndüren Fonksiyonlar - 2

Bir önceki konuda incelenen tablo döndüren fonksiyonlar bir sorgunun sonucudu doğrudan döndüren fonksiyonlardı. Bazı durumlarda öyle bir ihtiyaç olur ki fonksiyon veri listesini döndürmeden önce üzerinde işlemler yapmak gerekebilir. Fonksiyon içinde tanımlı bir tablo değişkene ekleme çıkarma yapmak gerekebilir. Bu durumlarda tablo tipinde bir değişken tanımlanır ve fonksiyon bu tabloyu döndürür. Tanımlanan tablo üzerinde veri ekleme ve çıkarma işlemleri yapılabilir.

Sonuçta oluşturulan tablo üzerinde fonksiyon çalışmasını bitirmeden önce veri ekleme ve silme işlemleri yapılabilir.

Örnek:

Ürün numarasını parametre olarak çalışan bir fonksiyon tanımlayınız. Fonksiyon ilgili ürünü sipariş veren kullanıcıların ad ve soyadlarını, sipariş tarihini ve ürünün reyon bilgisini listeleyecektir.

Çözüm:

Bu soruyu tablo döndüren bir fonksiyon olarak tanımlamak için tablo tipinde bir geri dönüş tablosu tanımlanır. Bu tabloda bulunması gereken kolonlar aynen bir tablo tanımında olduğu gibi belirtilir.

```
create function UrunSiparisBilgi
(@urunNumarasi int)
returns @sonucListesi table
(
	ReyonAd nvarchar(max),
	Tarih datetime,
	Ad nvarchar(max),
	Soyad nvarchar(max)
)
as
Begin
	insert into @sonucListesi(ReyonAd,Tarih,Ad,Soyad)
	select R.Ad,S.Tarih,K.Ad,K.Soyad
	from Siparis S
	inner join Urun U on S.UrunNo=U.No
	inner join Reyon R on U.ReyonNo=R.No
	inner join Musteri M on S.MusteriNo=M.No
	inner join Kullanici K on M.KullaniciNo=K.No
	where S.UrunNo=@urunNumarasi --sadece argümanda belirtilen numaralı ürünü getir

	--boş olan tarih alanları şu anki tarih olarak ayarlansın
	update @sonucListesi set Tarih=getdate()
	where Tarih is null

	return
End
```

Örnek:

Numarası verilen bir ürüne kaç yorum yazıldığını döndüren skaler bir fonksiyon tanımlayınız. Fonksiyon ürün numarasını parametre olarak alacaktır.

Çözüm:

```
create function UrunYorumSayisi
(@urunNumarasi int)
returns int
as
Begin
	declare @yorumSayisi int
	select @yorumSayisi=count(*)
	from Yorum Y
	where y.UrunNo=@urunNumarasi

	return @yorumSayisi
End
```

Örnek:

Kullanıcının favorilerine eklediği ürünlerin adını ve fiyatını, kullanıcının ad, soyad bilgilerini listeleyen tablo döndüren bir fonksiyon tanımlayınız. Fonksiyon kullanıcı numarasını parametre olarak alacaktır.

Çözüm:

Kullanıcının favori ürünleri Favori isimli tabloda tutulmaktadır. Bu yüzden Favori tablosu ile kullanıcı ve ürün tablolarını uygun şekilde birleştirerek sorgulamalıyız.

```
alter function KullaniciFavoriUrun
(@kullaniciNumarasi int)
returns @favoriListe table
(
	Ad nvarchar(max),
	Soyad nvarchar(max),
	UrunAd nvarchar(max),
	Fiyat money,
	Tarih datetime
)
as
Begin

	insert into @favoriListe(Ad,Soyad,UrunAd,Fiyat,Tarih)
	select K.Ad,K.Soyad,U.Ad,U.Fiyat,Tarih
	from Favori F
	inner join Kullanici K on F.KullaniciNo=K.No
	inner join Urun U on F.UrunNo=U.No
	where K.no=@kullaniciNumarasi
	--Tarih verisi boş olan kayıtlara şu anki tarihin atanması
	update @favoriListe set Tarih=GETDATE() where tarih is null

	return
End
```

NOT: Fonksiyon içinde @fovoriListe isimli bir tablo, fonksiyon çalıştığında geri döndürülecek tablo olarak tanımlanmıştır. Bu tabloya bir sorgu sonucunda elde edilen veriler aktarılmıştır. Bu işlemden sonra bir sql kodu daha çalıştırılmıştır. @fovoriListe tablosunda, tarih alanı boş olan kayıtların tarih alanına, o anın tarihi gelecek şekilde bir güncelleme yapılmıştır.

6. View Tanımlama

Viewler sorguları basitleştirmek, bir tablonun sütunlarına erişim yetkisi düzenlemek, sorgu süresini kısaltmak için sql sorgusu ile tanımlanan, gerçekte olmayan sanal tablolardır. View bir sql sorgusunun sonucunun bir tablo gibi saklanması amacıyla kullanılır. Bir view tablolarla birleştirilerek sorgulanabilir. Veiwler üzerinde güncelleme, ekleme, silme yapılabilir. View üzerinde indeks tanımlanabilir.

Viewler tablo döndüren fonksiyonlar gibi bir sorgu içinde kullanılabilirler. Fonksiyonlar bir parametre alarak çalışabilirken viewler herhangi bir parametre almadan tablo gibi sorguya dahil edilir.

6.1.View Oluşturma

View tanımı çok kolay bir şekilde yapılabilir. Yazılan bir sorgu sonucu doğrudan bir view olarak tanımlanabilir. Bu sayede bu sorgu sonucunu sanki sanal bir tablo gibi kayıt altına almış oluruz. Viewi oluşturan sorgu yazılırken sorgu sonucunda gelen tüm sütun adları farklı olmalıdır. Viewde aynı isimli iki sütun bulanamaz. Nasıl ki tablodaki tüm sütun adları farklıdır, viewde de sorgu sonucunda gelen sütün isimleri farklı olmak zorundadır.

Örnek:

Kullanıcının favori ürün bilgilerini getirecek bir view aşağıdaki gibi tanımlanabilir.

```
create View KullaniciFavoriView
as
Select
       K.Ad,
       K.Soyad,
       U.Ad as UrunAdi,
       U.Fiyat,
       U.No as UrunNo,
       K.No as KullaniciNo
from Kullanici K
left join Favori F on k.No=F.KullaniciNo
left join Urun U on F.UrunNo=U.No
```

Yukarıdaki örnekte görüldüğü gibi bir view farklı tablolardan veri getirecek şekilde tanımlanabilir. Tanımlanan bir view tablo gibi sorgulanabilir.

```
select * from kullanicifavoriview
```

Örnek:

Her bir kullanıcının adını, soyadını, sipariş verdiği ürün sayısını, siparişlerinin toplam fiyatını barındıran bir view tanımlayınız. Hiç sipariş vermeyen kullanıcılar da viewde listelenecektir. Sipariş vermeyen kullanıcıların ürün sayısı ve ücret bilgileri 0 olarak listelenecektir.

Çözüm:

```
create view SiparisBilgiView
as
select
              K.No as KullaniciNo,
              K.Ad,K.Soyad,
              count(s.no) as SiparisSayisi,
              sum(U.Fiyat) as ToplamFiyat
from Siparis S
inner join Urun U on S.UrunNo=U.No
inner join Musteri M on S.MusteriNo=M.No
right join Kullanici K on m.KullaniciNo=K.No
group by  K.No,K.Ad,K.Soyad
```

6.2.Viewler Üzerinde Veri Ekleme, Güncelleme

Viewler üzerinde veri ekleme, silme ve güncelleme operasyonları yapılabilir. Bu işlemler yapılırken dikkat edilmesi gereken şey eklenen ve silinen verilerin tek bir tabloyu etkileyecek şekilde yazılmasıdır.

```
select * from kullanicifavoriview

insert into kullanicifavoriview(Ad,Soyad)
values('Muhammet','Kale')

update kullanicifavoriview set Ad='Muhammet Ali'
where KullaniciNo=20

delete from kullanicifavoriview where kullanicino=20
```

Yukarıdaki delete sql kodu view üzerinde çalıştırlamak istenirse hata verir. Çünkü view birden çok tablodan oluşur. Aynı anda birden fazla tablodan veri silinmesi işlemi viewde yapılamaz. Sadece tek tablodan veri çekerer oluşturulan bir view varsa o zaman silme işlemi kullanılabilir.

View üzerinde güncelleme yapılırken filtrelemenin ve hangi sütunun güncelleneceği dikkatli kontrol edilmelidir. Aşağıdaki kod 19 numaralı kullanıcının satın aldığı tüm ürünlerin adını değiştirecektir.

```
update FavoriBilgiView set UrunAd='Buhar Makinesi'
where KullaniciNo='19'
```

6.3.View Tanımını Gizleme

Yazılan bir view içeriğini gizlemek isteyebiliriz. View içeriğini gizlemek için aşağıdaki gibi `ENCRYPTION` anahtar kelimesi kullanılır. Kodu görünmemesi için şifrelenen bir viewi onu tanımlayan kullanıcı da açamayacaktır. Bu sebeple şifrelenen viewi oluşturan sql komutları ayrı bir yerde saklanmalıdır. `with ENCRYPTION` seçeneği kullanılmaz ise view şifrelenmeden saklanmış olur.

```
ALTER view SiparisBilgiView
with ENCRYPTION
as
select
```

```
            K.No as KullaniciNo,
            K.Ad,K.Soyad,
            count(s.no) as SiparisSayisi,
            sum(U.Fiyat) as ToplamFiyat
from Siparis S
inner join Urun U on S.UrunNo=U.No
inner join Musteri M on S.MusteriNo=M.No
right join Kullanici K on m.KullaniciNo=K.No
group by  K.No,K.Ad,K.Soyad
```

Örnek:

Hangi markadan kaç ürün siparisi verildiğini listeleyen sql sorgusunu yazınız. Sorgu sonucunda her bir markanın adı, sipariş verilen ürün sayısı, toplam fiyat bilgileri listelenecektir. Hiç sipariş verilmeyen markalar da sorgu sonucunda listelenecektir.

Çözüm:

```
create view SiparisBilgiView
as
select
       M.No as MarkaNo,
       M.Ad,
       count(s.No) as UrunSayisi,
       sum(U.Fiyat) as ToplamFiyat
from Siparis S
right join urun u on s.UrunNo=u.No
inner join Marka M on U.MarkaNo=M.No
group by M.Ad,M.No
```

7. İndex Tanımlama

İndeksler kitapların içindekiler tabloları gibidirler. Bir kitaptan arama yapılırken önce içindekiler tablosuna bakar sonra doğrudan ilgili sayfaya yöneliriz. Böylece aradığımız veriye çok daha hızlı bir şekilde erişebiliriz.

İndeks kavramını anlamak için bir kütüphaneci örneğini inceleyebiliriz. Kütüphanelerde aranana kitaplara hızlı erişebilmek için kitapların fiziksel olarak raflara yerleştirildiği bir düzen bulunur. Kitapların fiziksel olarak doğrudan yerleştirildiği raflardaki düzen en hızlı erişim düzenidir. Mesela kitaplar ada göre sıralı yerleştirilirse ada göre bir clustered indeks tanımlanmış olur. Aranan bir kitaba en hızlı erişim yöntemi clustered indekstir. Kitaplar yazarlara göre de ulaşılacak şekilde düzenlenmek istenirse bunun için yazar adlarına göre kitap isimlerinin bulunduğu bir liste hazırlanır. Kitap yazara göre aranırken önce bu listeye bakılarak ilgili yazarın kitaplarının bulunduğu raflar tespit edilir. Sonra kitaplara doğrudan erişilir. Bu indeks türüne de nonclustered indeks diyoruz.

Bir tabloda indeks tanımlanırken en çok sorgulama ve filtreleme yapılan alanlar üzerinde indeks tanımlanmalıdır. Misafir anahtar alanlar bu iş için potansiyel alanlardır. Genellikle filtreleme işlemleri misafir anahtar alanlar üzerinden yapılır. Ancak bazı durumlarda misafir anahtar olmayan alanlar üzerinde de yoğun şekilde filtreleme yapılabilir. Bu durumda ilgili alan üzerinde bir indeks tanımlanması gerekir. TckimlikNo alanına göre sürekli sorgu çalıştırılıyorsa bu alan üzerinde bir indeks tanımlanabilir.

Birincil anahtar alanlar ve unique olarak tanımlanmış alanlar için otomatik olarak indeks oluşturulur.

İndeksler çok sayıda veri barındıran tablolardaki sorgular için performans sağlarlar. Bu sebeple çok az sayıda veri bulunan tablolar üzerinde indeks tanımlanması faydalı olmayacaktır.

Sql serverda indeksler tablolar üzerinde tanımlanan yapılardır. Tablonun belirtilen alanları için indeks tanımı yapılır. Aşağıda bir indeksin sql komutuyla nasıl tanımlandığını görebilirsiniz.

```
create index Siparis_UrunNo_Index ON Siparis(UrunNo)
```

İndeks tanımlanırken herhangi bir indeks türü belirtilmezse nonclustered indeks tanımlanmış olur. Bir tablo üzerinde yalnız bir tane clustered indeks bulunabilir.

Bir sütun üzerinde birden fazla indeks tanımlanmamalıdır. Sık sorgulanan alanlar için indeks tanını yapılabilir. Birden fazla sütun için aynı anda indeks tanımlanabilir. Bu indekse covering indeks diyoruz. Aşağıda birden fazla sütun üzerinde indeks tanımlama kodunu görebilirsiniz.

```
create index Kullanici_TcKimlik_Indeks
ON Kullanici(TcKimlikNo,Ad)
```

İndeksler tanımlanırken pek çok seçenek tanımlanabilir. FILLFACTOR bu seçeneklerden biridir. Yüzde olarak ayarlanır. İndeks sayfalarında başlangıçta ne kadar boşluk bırakılacağı bu özellik ile ayarlanır. FillFactor belirtilmediğinde varsayılan ayarlar etkin olur. Varsayılan değer 0'dır. Mesela bir indeks tanımlanırken fillfactor değeri verilmek isteniyorsa aşağıdaki gibi tanımlama yapılır.

```
create index Siparis_MusteriNo_Index
on Siparis(MusteriNo)
with (FILLFACTOR=10)
```

Viewler de tablolar gibi davaranan elemanlar olduğundan bunlar üzerinde de indeks tanımlanabilir.

İndeksler tablolara eklenen veri miktarına göre takip edilmesi ve gerekirse yeniden düzenlenmesi sağlanmalıdır. Bu işleme rebuild ve reorganize denir. Management stduio arayüzünden ilgili indekse sağ tıklanarak indek özelliklerinde bu işlem yapılabildiği gibi sql komutuyla da bu işlem yapılabilir. Hatta bu sql komutu zamanlanmış görev olarak tanımlanarak indekslerin belli zaman aralıklarında yenilenmesi sağlanabilir. İndeksler fragmetasyon değeri %15'in üzerine çıktığında yeniden oluşturulması gerekir.

Aşağıda bir indeksin sql komutu kullanılarak nasıl reorganize edildiğini görebilirsiniz.

```
alter index Urun_Marka_Indeks
ON Urun
REORGANIZE


alter index Urun_MarkaNo_Index
on Urun
rebuild
```

Var olan bir indeksi silmek istiyorsak drop komutunu kullanabiliriz.

```
DROP INDEX Urun_Marka_Indeks
on Urun
```

İndekslerle ilgili bili veren birtakım sistem fonksiyonları vardır. Sql komutuyla indeksler hakkında bilgi almak için bu sistem fonksiyonları kullanılabilir. Aşağıda indekslerle ilgili bir fonksiyonu sorgulayan bir komut görülmektedir. `sys.dm_db_index_physical_stats` isimli fonksiyon indekslerin dağınıklık durumunu gösterir.

```
select  * from sys.dm_db_index_physical_stats(DB_ID(),null,null,null,'LIMITED') A
```

```
inner join sys.indexes i on a.index_id=i.index_id
```

8. Vize Sınavı

Vize sınavına buraya kadar incelenen tüm konular dahildir. Verileri gruplayarak sorgulama, prosedür tanımlama, view oluşturma, fonksiyon tanımlama, indeks oluşturma konularında sorular karşınıza çıkacaktır. Bu sorguları yazabilmek için VTYS I dersinden aşağıdaki konulara hakim olmanız gerekmektedir. Veritabanı tasarımı sorusu da sorulacaktır. Yazılan bir metinden verileri çıkarıp ilişkili tablolar olarak tasarlamanız istenecektir. Bu sebeple veritabanı tasarımı konusunda çalışma yapmanız tavsiye edilir.

Tabloları birlikte sorgulama, filtreleme, tabloları iliştkilendirme, veritabanı tasarımı, verileri gruplayarak sorgulama, tabloya veri ekleme, silme, güncelleme, karar yapılarının kullanımı, hesaplanmış sütun oluşturma, tablolara takma da verme, sütunlara takma ad verme.

9. Örnek Veritabanı Tasarım Uygulaması

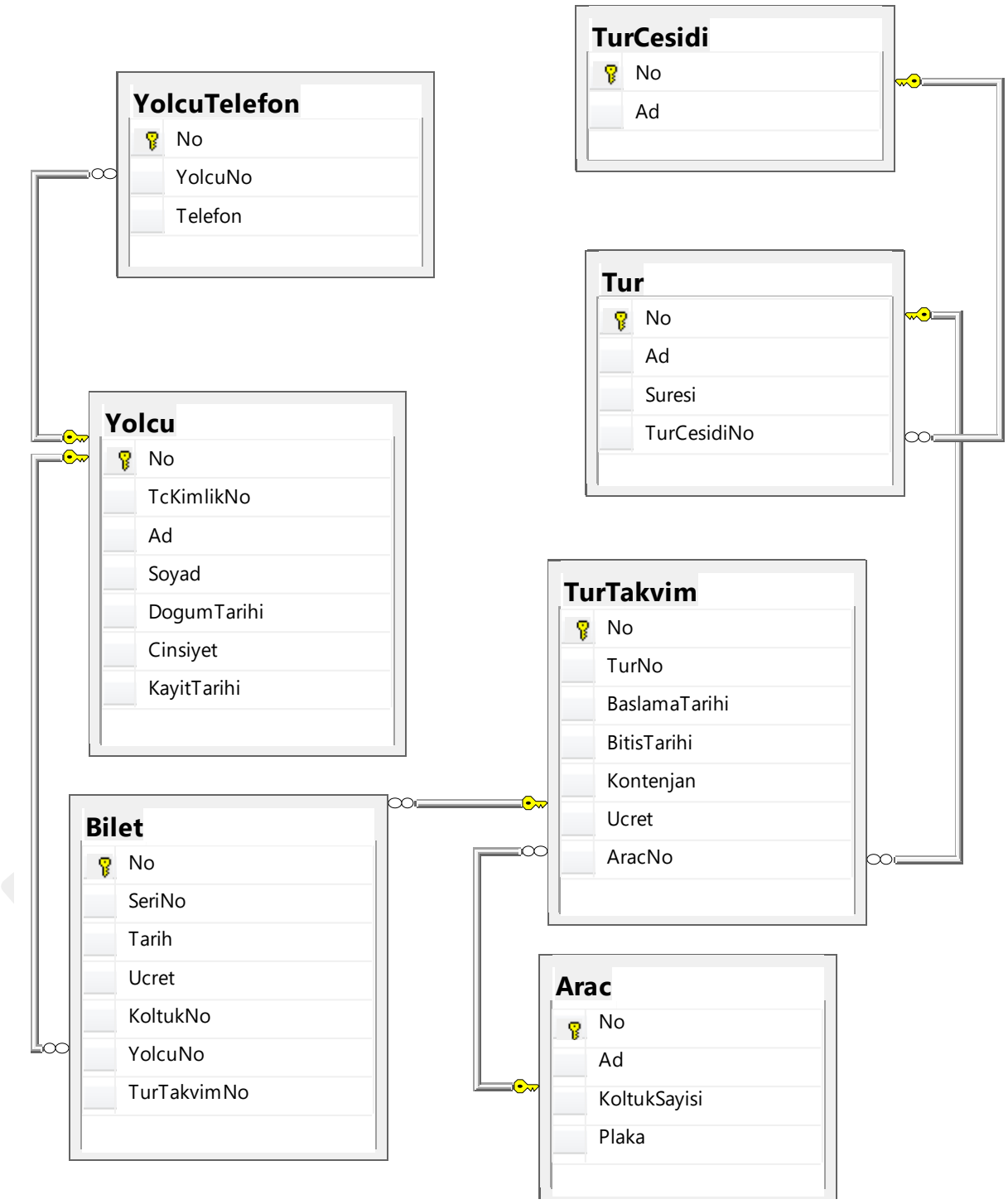
Bir tur firması için veritabanı tasarımı yapılacaktır. Tur firmasının ihtiyaçları aşağıda açıklanmıştır. Belirtilen ihtiyaçlara cevap verebilecek bir veritabanı tasarımı yapınız.

1. Firma birçok farklı tur düzenlemektedir. Bir tur yılın farklı zamanlarında yapılabilmektedir.
2. Yolcular ilgili tarihteki turlara bilet almaktadır. Sistemde yolcu bilgileri ve bilet bilgileri tutulacaktır. Bir yolcunun birden fazla telefon numarası olabilecektir.
3. Turların hangi araçlarla yapıldığı bilgisi sistemde tutulacaktır.
4. Araçlar sisteme tanımlanacaktır. Her bir aracın plakası, koltuk sayısı bilgileri sistemde tanımlanacaktır.
5. Her bir turda araçta görevli şoför bilgisi de sistemde tutulacaktır.
6. Turların fiyatları yılın farklı dönemlerinde farklı olabilmektedir. Her bir tur yılda birçok kez tekrar düzenlenebilmektedir.
7. Yolcular satın aldıkları tur biletlerini hangi numaralı koltuk için aldılar, ne zaman satın aldılar, hangi tarihteki tur için aldılar bilgileri sistemde tutulacaktır.
8. Her bir tur için bir tür belirlenebilecektir. Tarih kültür gezisi, doğa gezisi, yurt dışı tarihi turistik yerler, deniz turizmi, yayla turizmi gibi türler sisteme kayıt edilecektir.

Çözüm:

Problemde anlatılan sistemde bulunan nesne listesini çıkaralım. Nesne dediğimiz şey veritabanında hakkında veri tutulması istenen her şeydir. Fiziki varlığı olabilir veya olmayabilir hakkında veri sakladığımız bir kavram nesne olarak adlandırılır. Nesne listemiz aşağıdaki gibi çıkartılabilir.

Tur, Araç, Şoför, Bilet, Takvim, Yolcu,Gezi Türü nenselerine ihtiyaç olduğu yukarıdaki maddelerde görülmektedir.



Turizm veritabanı üzerinde örnek sorgular.

```
--**Tur çeşitlerinin tanımlandığ tabloya yeni bir kayıt
--ekleyen stored prosedür yazalım
create proc TurCesidiEkle
@Ad nvarchar(max)
as
insert into TurCesidi(Ad)
values (@Ad)

--Tur çeşit ekleme prosedürünün kullanımı
TurCesidiEkle 'İş Seyehati'
```

```
--Tur tablosuna yeni kayıt ekleyen bir prosedür yazınız
create proc TurEkle
@Ad nvarchar(max),@Suresi int,@TurCesidiNo int
as
insert into Tur(Ad,Suresi,TurCesidiNo)
values(@Ad,@Suresi,@TurCesidiNo)

--TurEkle prosedürünün kullanımı
TurEkle 'Aselsan Gezisi',5,3

---Yolcu numarası belirtilen kişinin aldığı bilet bilgilerini
--listeleyen bir fonksiyon yazalım

alter function YolcuBiletBilgi
(@YolcuNo int)
returns table
as
return
Select B.* from Yolcu Y
inner join Bilet B on Y.No=B.YolcuNo
where Y.No=@YolcuNo

--tanımlananan fonksiyonun kullanımı
select * from dbo.YolcuBiletBilgi(2)
```

İkinci öğretim veritabanı örnek sorgular

```
--gezitürü tablosuna yeni kayıt elyen bir prosedür yazınız

create proc GeziTuruEkle
@Ad nvarchar(max)
as
insert into GeziTuru(Ad)
values(@Ad)

--Tur tablosuna yeni kayıt ekleyen bir prosedür yazınız
create proc TurEkle
@Ad nvarchar(max),@KacGun int, @GeziTurNo int
as
insert into Tur(Ad,GeziTurNo,KacGun)
values(@Ad,@GeziTurNo,@KacGun)

--Her bir turun adını ve gezi türünü listeleyen bir fonksiyon yazınız
create function TurBilgiListele()
returns table
return
select
       T.Ad as TurAdı,
       G.Ad as GeziTürü
from Tur T
inner join GeziTuru G on T.GeziTurNo=G.No

---yolcu numarası parametresiyle çalışan bir fonksiyon tanımlayınız
---numarası verilen yolcunun bilet bilgisini ve tur adını listeleyecek
alter function YolcuBiletBilgi
(@YolcuNo int)
returns table
return
select
       Y.Ad as YolcuAdı,
       Y.Soyad as YolcuSoyad,
```

```
	B.*,
	T.Ad as [Tur Adı],
	G.Ad as GeziTürü
from Bilet B
inner join Takvim Tk on B.TakvimNo=Tk.No
inner join Tur T on Tk.TurNo=T.No
inner join GeziTuru G on T.GeziTurNo=G.No
inner join Yolcu Y on B.YolcuNo=Y.No
where B.YolcuNo=@YolcuNo


--Takvim tablosuna yeni kayıt ekleyen bir prosedür yazınız
create proc TakvimEkle
@BaslamaTarihi date,@BitisTarihi date,@Ucret money,
@TurNo int,@AracNo int, @SoforNo int
as
insert into Takvim(BaslangicTarihi,BitisTarihi,
Ucret,SoforNo,AracNo,TurNo)
values(@BaslamaTarihi,@BitisTarihi,@Ucret,@SoforNo,@AracNo,@TurNo)

--Her bir tur için kaç bilet satıldığın listeleyen bir prosedür yazınız
--Hiç bilet satılmayan turlar da 0 olarak listede bulunacaktır
alter proc TurBiletSayisi
as
select
	Tur.Ad,
	COUNT(B.No) as BiletSayisi
from Bilet B
inner join Takvim T on B.TakvimNo=T.No
right join Tur on T.TurNo=Tur.No
group by Tur.Ad

--Bilet tablosu üzerinde indeks tanımlanmak istenirse hangi alanlar üzerinde indeks
tanımlanmalıdır
--Alanları belirtiniz, bir alan üzerindeki indeksi tanımlayınız

create index BiletYolcuIndeks
on Bilet(YolcuNo)

--Hangi aracın kaç kez kullanıldığını listeleyen bir prosedür yazınız.
--Hiç kullanılmayan araçlar da 0 olarak listelenecektir.
alter proc AracKullanimSayisi
as
select
	A.Plaka,
	Count(T.No) as KullanimSayisi
from Arac A
left join Takvim T on A.No=T.AracNo
group by A.Plaka
```

Turizm veritabanı üzerinde çalışma soruları

1. Bilet tablosü üzerinde indeks tanımlanmak istendiğinde hangi alanlar üzerinde indeks tanımlanabilir. İlgili alanları belirtiniz. Bir tanesi üzerinde bir indeks tanımlayınız.
2. Bir kullanıcının adı soyadı, tckimlik numarası, bilet seri numarası, bilet tarihi, biletin satın alındığı tur adını gösterecek bir view tanımlayınız.
3. Yaşı 30'dan küçük olan yolcuların hangi tur için kaç bilte satın aldığını listeleyen sql sorgusunu yazınız.

4. Yolcu no parametresi ile çalışan, yolcunun kaç bilet aldığını ve toplam ödediği ücreti listeleyen bir sql prosedürü yazınız. Prosedür çağırıldığında yolcunun adı soyadı, satın aldığı bilet sayısı, ödediği toplam ücret bilgileri listenecektir.

10. Triggerlar

Triggerlar veritabanında tablolar üzerinde yapılan değişikliklerde otomatik devreye giren yapılardır. Bir trigger bir tablo üzerinde yapılan update, insert veya delete işlemleriyle tetiklenebilir. Triggerlar aşağıdaki amaçlar için kullanılabilir.

1. Tablolar üzerinde yapılan değişikliklerin takip edilmesi gerektiğinde triggerlar kullanılabilir. Kritik verilerin bulunduğu tablolarda yapılan değişikliklerin takip edilmesi istendiğinde bu tablo üzerinde bir trigger tanımlanarak değişikliklerin loglanması sağlanabilir.
2. Birincil anahtar alanların değerinin otomatik artan alan değil de kendi ürettiğimiz bir değer olması istendiğinde bu işlem için trigger tanımlanabilir. Tabloya kayıt eklenmesi sırasında devreye giren trigger birincil anahtarı oluşturur ve kayıt bu birincil anahtarla tabloya eklenir.
3. Bazı veriler kayıt edilirken karmaşık iş kuralları işletilmek istenebilir. Eklenen verilerin bir takım kurallara bağlı olması gibi bir takım kurallar trigger olarak tanınmlanabilir.
4. Veritabanı erişimlerini takip altına almak için kullanılabilir.
5. Veritabanı nesneleri üzerinde yapılan değişiklikleri takip etmek için de triggerlar tanımlanabilir.

10.1. Triggerlar Ne Zaman Kullanılmaz

Aynı veritabanı içinde birincil anahtar ve misafir anahtar uyumluluğunu kontrol etmek için trigger kullanılmaz. Bu özellik tablolar üzerinde tanımlanan misafir anahtarlar vasıtasıyla elde edilir.

Bir sorgudan kaç kayıt etkilendiğini bulmak için trigger kullanmak yerine @@ROWCOUNT fonksiyonu kullanmak daha performanslı bir sonuç verecektir.

10.2. Klasik Triggerlar

Bir trigger birden fazla işlemi gerçekleştirmek üzere yazılabilir. Mesela bir tabloya veri ekleme yapıldığında eklenen verilerin başka bir tabloya loglanması için bir trigger yazılabilir. Bir trigger hen uptade, hem de insert işlemlerinde tetiklenecek şekilde tanımlanabilir.

Temel olarak iki farklı trigger çeşidi vardır. After trigger ve Instead of trigger. After triggeri sadece tablolar üzerinde tanımlanabilir. Instead of trigger ise hem tablolar hem de viewlar üzerinde tanımlanabilir.

10.3. Trigger Tanımlama

```
create trigger Trigger_Adı
ON Tablo_Adı
With encription (kullanımı zorunlu değildir. Trigger kodlarını gizlemeyi sağlar)
FOR AFTER / INSTEAD OF INSERT(triggeri tetikleyecek olay)
As
----Buraya trigger tetiklendiğinde çalışması istenen
--sql komutları yazılır.
```

Örnek:

Yolcu tablosu üzerinde bir veri ekleme işlemi yapıldığında devreye girecek bir trigger yazalım.

```
create trigger YolcuEkleTrigger
ON Yolcu
AFTER  INSERT
as
declare @yeniDeger nvarchar(max)
declare @IslemTuru nvarchar(max)
declare @TabloAdi nvarchar(max)
set @TabloAdi='Yolcu'
set @IslemTuru='Ekleme'
select @yeniDeger=Ad+Soyad from Inserted
insert into Log(YeniDeger,TabloAdi,IslemTuru,Tarih)
values(@yeniDeger, @TabloAdi, @IslemTuru,GETDATE())
```

Yukarıdaki komut incelendiğinde inserted isimli bir tablonun varlığı dikkat çekmektedir. Sql server bir tabloya veri eklemeden önce eklencek verileri inserted isimli bellek üzerinde oluşturduğu bir tabloya yazar. Bu işlemden sonra ana tabloya değişiklik aktarılır. Silme işleminde de önce silinecek veri deleted isimli bellek tablosuna yazılır. Sonra veri ana tablodan silinir. Bir silme işlemi yapılıyorsa inserted tablosu oluşturulmaz sadece deleted tablosu oluşturulur. Bir veri ekleme işi yapılıyorsa deleted tablosu oluşturulmaz sadece inserted tablosu oluşturulur. Güncelleme işlemi yapılırken her iki tablo da oluşturlur. Verinin eski hali deleted tablosunda yeni tali inserted tablosunda tutulur. İşlemler ana tablo üzerinde gerçekleştirildiğinde bu tablolar bellekten silinir.

Yukarıdaki trigger ad soyad verilerinin karakter sayısını kontrol edecek şekilde yeniden düzenlenirse veriler kayıt edilirken belli şartlara bağlı olup olmadıkları kontrol edilebilir. Örneğimizde ad veya soyad alanına girilen verinin karakter sayısı 3 karakterden az ise trigger rollback işlemi yapacaktır. Yani girilen veriyi iptal edecek ve tabloya eklenmesine izin vermeyecektir. Bu esnada RAISERROR sistem fonksiyonu ile de bir hata mesafi fırlatabiliriz. Bu sayedce kullanıcıları veritabanı seviyeli olarak bilgilendirmiş oluruz.

```
alter trigger YolcuEkleTrigger
ON Yolcu
AFTER  INSERT
as
declare @yeniDeger nvarchar(max)
declare @IslemTuru nvarchar(max)
declare @TabloAdi nvarchar(max)
declare @Ad nvarchar(max)
declare @soyad nvarchar(max)

set @TabloAdi='Yolcu'
set @IslemTuru='Ekleme'
select @yeniDeger=Ad+Soyad,@Ad=Ad,@soyad=Soyad from Inserted

if Len(@Ad)<3 or Len(@soyad)<3
	Begin
		RAISERROR('Geçersiz ad soyad verisi kayıt edilemez.',10,1)
		rollback
	End
else
	Begin
		insert into Log(YeniDeger,TabloAdi,IslemTuru,Tarih)
		values(@yeniDeger, @TabloAdi, @IslemTuru,GETDATE())
	End
```

Örnek:

Yolcu tablosu üzerinde update işlemin yapıldığında devreye girecek bir trigger yazalım. Trigger güncellenen yolcunun eski ve yeni bilgilerini log tablosuna kayıt etsin. Hangi kullanıcı güncelleniyorsa birincil anahtarını da log tablosundaki anahtardeger alanına kayıt etsin.

```
create trigger YolcuGuncelleTrigger
ON Yolcu
after update
as
declare @yeniDeger nvarchar(max)
declare @eskiDeger nvarchar(max)
declare @anahtar int

select
       @yeniDeger=ad+Soyad+Convert(nvarchar(max),dogumTarihi),
       @anahtar=inserted.No
from inserted

select @eskiDeger=Ad+Soyad+Convert(nvarchar(max),dogumTarihi)
from deleted

insert into Log(EskiDeger,YeniDeger,Tarih,TabloAdi,IslemTuru,AnahtarDeger)
values(@eskiDeger,@yeniDeger,GETDATE(),'Yolcu','Guncelleme',@anahtar)
```

Örnek:

Yolcu tablosundan bir kayıt silinirken devreye girecek bir trigger tanımlayınız. Trigger silinecek kullanıcıya ait bilet var ise bilet bilgilerini bilet tablosundan silecek ve silinen verileri log tablosuna loglayacaktır.

Çözüm:

Bu tür bir trigger ana tabloda işlem yapılmadan önce devreye girmelidir. Örneğimizde yolcu tablosundan kayıt silinmeden önce silinmek istene kayda ait bilet tablosundaki bağlı kayıtların silinmesi gerekmektedir. Bu türlü işlemler için instead of triggerı tanımlanmalıdır. Aftre trigger tanımlandığında kayıt silinemeyecek ve hata mesajı verecektir.

```
alter trigger YolcuSilTrigger
ON Yolcu
instead of delete
as
declare @eskiDegerler nvarchar(max)
declare @anahtar int
select @eskiDegerler=ad+soyad+Convert(nvarchar(max),kayittarihi)+
              convert(nvarchar(max),no)+Convert(nvarchar(max),dogumtarihi)+
              TcKimlikNo+CONVERT(nvarchar(max),cinsiyet)
from deleted

delete from Bilet where YolcuNo=@anahtar

insert into Log(EskiDeger,tarih,TabloAdi,IslemTuru)
values(@eskiDegerler,getdate(),'Yolcu','Silme')
```

Yukarıdaki gibi bir trigger tanımladığımızda yolcu ile ilişikili bilet kayıtları olsa bile yolcu kaydı ve bağlı bilet kayıtları sistemden silinecektir. Bu sebeple bu türlü bir trigger yazılırken bu işlemin nelere sebep olacağının net şekilde bilinmesi önemlidir. Yoksa istenmeyen kayıtların yanlışlıkla silinmesine sebep olabilirsiniz.

10.4. After Trigger

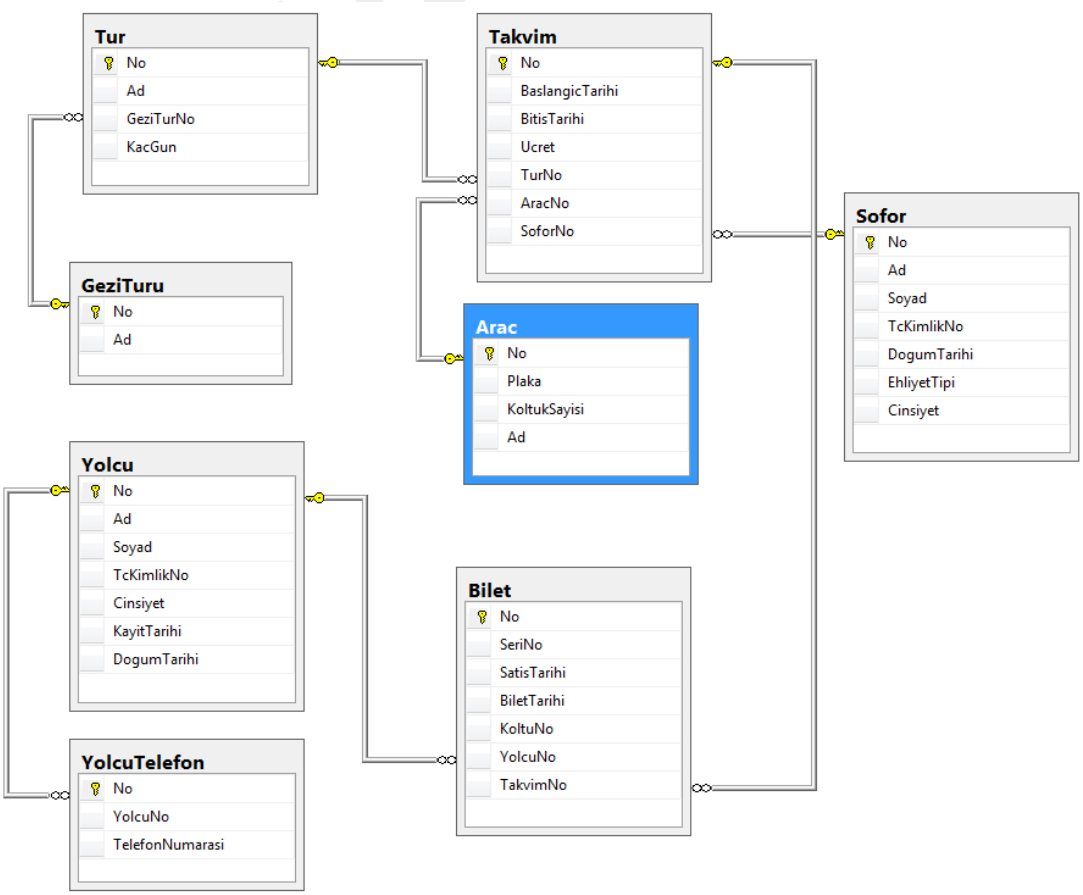
After trigger ana tabloya değişiklikler aktarıldıktan sonra tetiklenen trigger çeşididir. Sql server tablo üzerinde değişiklikleri yapar sonra bu trigger tetiklenir. Bu trigger ana tabloya veri eklendikten sonra tetiklendiği için ana tabloya yeni veri ekleme durumundan yeni bir birincil anahtar oluşturulmuş olur. İşlem triggerda iptal edilirse kayıt tablodan silinir.

10.5. Instead Of Trigger

Ana tabloya veriler yazılmadan önce tetiklenen trigger çeşididir. Veri inserted veya deleted tablolarına yazıldıktan sonra bu trigger tetiklenir. Ana tabloya verinin yazılması işlemi trgger tetiklendikten sonra yapılır. Bu sayedece eğer iptal edilecek bir işlem varsa ana tablo yen eklenen veriden hiç etkilenmeden işlem geri alınabilir.

11. Örnek Çalışmalar

Bu bölümde şimdiye kadar incelenen konularla ilgili örnek çalışmalar sunulmuştur. Aşağıdaki şemaya uygun olarak örnek sorgular yazılacaktır.



Misal 1:

Araç tablosuna yeni kayıt ekleyen bir prosedür yazınız.

Çözüm:

```
create proc AracEkle
(
@Ad nvarchar(max),
@KoltukSayisi nvarchar(max),@Plaka nvarchar(max)
)
as
insert into Arac(Ad,Plaka,KoltukSayisi)
values(@Ad,@Plaka,@KoltukSayisi)
```

Yukarıdaki prosedürde görüldüğü üzere arç tablosunun alanlarını girilecek veriler prosedüre parametre olarak tanımlanmıştır.

Misal 2:

Toplam kaç liralık bilet satıldığını hesaplayan ***ToplamBiletSatis*** isimli bir fonksiyon yazınız.

Çözüm:

```
create function ToplamBiletSatis()
returns money
as
Begin
declare @toplam money
select @toplam= SUM(Ucret) from Bilet
return @toplam
End
```

Yukarıdaki fonksiyon tek bir değer döndüren bir fonksiyondur. Yani bize bir liste değil yalnızca bir değişkeni sonuç olarak döndürmektedir. Bu fonksiyonu aşağıdaki gibi çağırarak kullanabiliriz.

```
select  dbo.toplambiletsatis()

select *,dbo.toplambiletsatis()
from Arac
```

Sorguda görüldüğü gibi tekil değer döndüren bir fonksiyonu bir sorgu içinde, sonuç kümesinde bir kolon oluşturacak şekilde kullanabiliriz. İlk kullanımda fonksiyon tek başına kendisi çalıştırılmış ve sonuç görüntülenmiştir. Bu fonksiyon bir sorgu içinde kolon oluşturmak için, sorguda filtre oluşturmak için kullanılabilir.

Misal 3:

Numarası belirtilen bir tur için toplam kaç liralık bilet satıldığını hesaplayan bir fonksiyon yazınız.

Çözüm:

```
create function TurToplamBiletSatis
(@turNo int)
returns money
as
Begin
declare @toplam money
select @toplam = Sum(Bilet.Ucret) from Bilet
inner join TurTakvim on Bilet.TurTakvimNo=TurTakvim.No
where TurTakvim.TurNo=@turNo
```

```
return @toplam
End
```

Fonksiyon tanımlanırken int türünden bir parametre tanımlandığına dikkat ediniz. Bir önceki örnekten farklı olarak bu fonksiyon tam sayı türünde bir parametre ile çalışacaktır. Bu fonksiyon her bir tur için kaç bilet satıldığını gösteren bir sorgu için kullanabiliriz. Böylece kolaylıkla her bir tur için toplam kaç liralık bilet satıldığını elde edebiliriz.

```
select
tur.Ad,
dbo.turToplamBiletSatis(tur.No) as ToplamBiletUcreti
from Tur
```

Sorgu içinde fonksiyon kullanırken parametresinin uygun bir alandan alındığına dikkat ediniz. Bu sorguda her bir tur numarası bu fonksiyona verilecek ve fonksiyon her bir tur için tekrar tekrar çalışarak satırlara verileri getirecektir.

Misal 4:

Her bir yolcu için bilet bilgilerini listeleyen bir fonksiyon yazınız. Fonksiyon yolcunun adını, soyadını, bilet tarihini, bilet ücretini, koltuk numarasını listeleyecektir. Hiç bilet almayan yolcular da listede görünecektir.

Çözüm:

Soruda fonksiyonun bize bir liste döndürmesi istenmektedir. Bu yüzden fonksiyonun sorgusu yazılırken tablo döndüren bir fonksiyon kullanılması gerekir. Bir sorgunun sonucunu da doğrudan döndürebiliriz. Hiç bilet almayan yolcuların da listelenmesi istenmektedir. Bu sebeple left join kullanılması gerekmektedir.

```
create function BiletDetay()
returns table
as
return
select
Yolcu.Ad,Yolcu.Soyad,Yolcu.No as YolcuNo,
Bilet.KoltukNo,bilet.Tarih,Bilet.No as BiletNo
from Yolcu
left join Bilet on Yolcu.No=Bilet.YolcuNo
```

Sorguda inner join kullanılması Bilet ve Yolcu tablolalarının sadece eşleşen kayıtlarının listelenmesini sağlar. Bilet tablosunda karşılık gelen verisi olmayan yolcular listede yer almayacaktır. Bu sebeple tablolar birleştirilirken inner join yerine left join kullanıldığına dikkat ediniz. Left join kullanıldığında yolcu tablosundaki tüm kayıtlar listelenir. Hiç bilet olmayan yolcular da sonuç kümesinde bulunur.

Misal 5:

Araç tablosu üzerinde bir trigger tanımlayınız. Trigeer tabloya veri eklenirken devreye girecek ve araç ad ve koltuk sayısı alanlarını kontrol edecektir. Ad alanı en az 3 karakter ve koltuk sayısı da en az 2 ise kayıt eklenmesine izin verecek değilse tabloya kayıt eklenmesine izin vermeyecek ve işlemi iptal edecektir. Trigger veri tabloya eklenmeden önce devreye girecektir.

Çözüm:

```
create trigger trg_AracEkle
on Arac
instead of insert
as
```

```
declare @aracAdi nvarchar(max)
declare @koltukSayisi int
select @aracAdi=Ad, @koltukSayisi=KoltukSayisi
from inserted
if @koltukSayisi<2 or Len(@aracAdi)<3
Begin
	raiserror('Koltuk sayısı geçersiz veya ad geçersiz',10,1)
	rollback
End
```

Tanımlaman trigger şartalara uygun olmayan bir kayıt geldiğinde rollback komutunu çalıştırır ve işlemi iptal eder. Kullanıcı özel olarak sql serverın bir hata mesajı dönmesini sağlayabilir. Bu işlem için raiserror sistem fonksiyon kullanılır. Bu fonksiyonun ilk parametresine metin olarak gösterilmek istenen hata mesajı yazılır. Diğer iki parametresi hatanın seviyesini ve durumunu gösterir. Tabloya geçersiz bir kayıt eklenmek istendiğinde kayıt eklenmez ve hata mesajı döndürülür.

Ana tabloya veri eklenmeden önce devreye giren trigger instead of triggerdır. Bu trigger çeşidi kullanıldığında veri tabloya eklenmeden önce kontrol edilir ve işlem geri alınır.

Misal 6:

Tur isimleri sütunlarda görünecek şekilde bir sorgu oluşturunuz. Her bir tur isminin altında kaç bilet satıldığı bilgisi görüntülenmek istenmektedir. Hiç bilet satılmayanlar 0 olarak listelenecektir.

Çözüm:

Aşağıdaki sorgu her bir tur için hangi yolcunun kaç bilet aldığını gösterir.

```
with turBilet(Ad,Soyad,TurAd,BiletNo)
as
(
	select Yolcu.Ad,Yolcu.Soyad, Tur.Ad,Bilet.No from Bilet
	inner join TurTakvim on Bilet.TurTakvimNo=TurTakvim.No
	right join Tur on TurTakvim.TurNo=Tur.No
	right join Yolcu on bilet.YolcuNo=yolcu.No

)
select
p.*
from turBilet
pivot
(
	count(BiletNo)
	For TurAd in([Karadeniz Turu],[Doğu Anadolu Turu],[Akdeniz Turu],['Aselsan
Gezisi],[Frig Vadisi Turu],[Balkanlar Turu])
)as p
```

Sorgu sonucu aşağıdaki gibidir.

Ad	Soyad	Karadeniz Turu	Doğu Anadolu Turu	Akdeniz Turu	'Aselsan Gezisi	Frig Vadisi Turu	Balkanlar Turu
Arzu	Akbalık	0	0	1	0	0	2
Mustafa+	Aladağ	0	0	0	0	0	0
Songül	Alkan	0	0	1	0	0	0
Aygül	Balaban	0	0	0	0	0	0
Halit	Değirmenci	0	0	0	0	0	0
Nazike	Değirmenci	0	0	0	0	0	0
Beril	Doğruoğlu	0	0	1	0	0	0
Oktay	Esin	0	0	1	0	1	0
Mehmet Ali	Pehlivan	0	0	1	0	0	0
Melike	Polat	0	0	1	0	0	0

Yolcu isimleri olmadan yalnızca her bir tur için toplam kaç tane bilet satıldığını gösteren sorgu ise aşağıdaki gibi olacaktır. Yeni sorguda yolcu bilgilerini sorgu sonucundan kaldırıldığını görebilirsiniz.

```
with turBilet(TurAd,BiletNo)
as
(
	select  Tur.Ad,Bilet.No from Bilet
	inner join TurTakvim on Bilet.TurTakvimNo=TurTakvim.No
	right join Tur on TurTakvim.TurNo=Tur.No
	right join Yolcu on bilet.YolcuNo=yolcu.No

)
select
p.*
from turBilet
pivot
(
	count(BiletNo)
	For TurAd in([Karadeniz Turu],[Doğu Anadolu Turu],[Akdeniz Turu],['Aselsan
Gezisi],[Frig Vadisi Turu],[Balkanlar Turu])
)as p
```

Yukarıdaki sorgunun sonucu aşağıdaki gibi olacaktır.

Karadeniz Turu	Doğu Anadolu Turu	Akdeniz Turu	'Aselsan Gezisi	Frig Vadisi Turu	Balkanlar Turu
0	0	6	0	1	2

Çalışılacak konu başlıkları

1. Prosedür tanımlama ,fonksiyon tanımlama, view tanımlama
2. Group by kullanımı
3. Rollup deyimi kullanımı
4. Pivot operatörü
5. Veritabanı tasarımı
6. Trigger tanımlama
7. Tabloları birleştirerek sorgulama

8. Rekürsif Sorgu

Kategori tablosundaki bir kategoriyi tüm alt kategoriyle birlikte listeleyen sorgu aşağıdaki gibidir.

```
--rekürsif sorgu oluşturma
```

```
with KategoriListe(KategoriNo,Ad,UstKategoriNo)
as
(       select * from Kategori where No='1'
        union all
        select Kategori.No,Kategori.Ad,kategori.ustkategoriNo
        from KategoriListe
        inner join Kategori on KategoriListe.KategoriNo=Kategori.ustKategoriNo
)
select KategoriListe.*,Kategori.ad ustKategori from KategoriListe
left join Kategori on KategoriListe.UstKategoriNo=Kategori.No
```