DATABASE

Öğr. Gör. Evgin GÖÇERİ

10 Kasım 2014

Saklı Yordam (SY) Oluşturmak ve Çalıştırmak

- SY'ların performans açısından ne sağladığını anlamak için kodlandıktan sonra çalıştırıncaya kadar hangi aşamalardan geçtiğini ve sonraki çalıştırılmasında nasıl bir yol izleyerek sonuç ürettiğini bilmek gerekir

- Bir SY oluşturulduktan sonraki ilk çalıştırılmasında şu aşamalara tabi tutulur:

  - Ayrıştırma : İfadelerin geçerliliği denetlenir. Sorgu ağacı (query tree) veya sıra ağacı (sequence tree) denilen yapı oluşturulur. Bu aşamada, *sysobjects* tablosuna SY'ın ismi kaydedilir. SY'ın kodları da *syscomments* tablosuna kaydedilir

  - Derleme : Bir önceki aşamada oluşturulan sıra ağacı dikkate alınarak çalıştırma planı çıkarılır. Çalıştırma planı güvenlik, hak ve yetkiler dahilinde denetlenir.

  - Çalıştrma : Bir önceki aşamada oluşturulan çalıştırma planı dikkate alınarak işlemler gerçekleştirilir.

Saklı Yordam (SY)
Oluşturmak ve Çalıştırmak - 2

- Bir SY ikinci defa çalıştırılırsa aynı aşamaları takip etmez.

Çünkü;

- Derleme işleminden sonra elde edilen çalıştırma planı *Prosedür Hafızası* denilen bir hafızada saklanır.
- Takip eden çağırma istekleri, bu hafızadaki çalıştırma planı kullanılarak cevaplanır.

İlk iki aşama atlandığı için daha hızlı olarak sonuca ulaşılır

Bir script bloğu çalıştırmaya göre, SY çalıştırmayı avantajlı kılan bu noktadır

Saklı Yordam (SY)
Oluşturmak ve Çalıştırmak - 3

- *Prosedür Hafizası*, Hafıza (RAM)'da bir alandır ve şu şartlardan biri oluşmadığı sürece, bu hafızada bulunan çalıştırma planları SY tarafından kullanılabilir:

1. SQL Server'ın kurulu olduğu bilgisayar kapatılırsa, RAM'deki ilgili bölge kaybolacağından çalıştırma planı kaybolacaktır

2. SY tarafından erişilen tablo veya view üstünde yapısal değişiklik olması halinde çalıştırma planının verimsiz kalması riski olduğundan, çalıştırma planı yeniden oluşturularak kullanılır

3. Çalıştırma planında kullanılmasına izin verilen bir indeksin silinmesi halinde

4. SY'ın bir sonraki çalışmada yeniden derlenmesi için bir ayarlama yapılması halinde

Saklı Yordam (SY)
Oluşturmak ve Çalıştırmak - 4

SY kullanmak şu faydaları sağlar:

1. Bir kez oluşturulduktan sonra birden çok uygulama tarafından kullanılarak hız ve tutarlı bir veri yönetimi sağlar

2. Güvenlik gerekçesi ile kullanılabilirler (Kullanıcının bir tabloya erişim izni olmasa bile o tablo üstünde işlem yapan bir SY'ı kullanma izni olabilir)

3. SY'lar dışarıdan aldıkları parametreleri değerlendirerek işlem yapabilirler

Saklı Yordam (SY)
Oluşturmak ve Çalıştırmak - 5

SY oluşturmada genel ifadeler:

CREATE PROC[EDURE] sakliyordam_adi
[WITH seçenekler]
AS

Buradaki WITH ifadesi ile SY içindeki kaynak kodu gizlemek gibi ek seçenekler belirtilebilir

SY'ların sonunda GO kullanmak zorunlu değildir

Saklı Yordam (SY) Oluşturmak ve Çalıştırmak - 6

Örnek: Gün içinde yapılan satışların sayısını veren SY;

```
CREATE PROC SP$gunlukSatis
AS
    SELECT COUNT(*)
    FROM tblSiparisDetay SD INNER JOIN tblSiparis S
ON SD.faturaKOD = S.faturaKOD
WHERE S.siparisTarih > GETDATE() – 1
AND S.siparisTarih < GETDATE() +1
GO
```

SY oluşturmak için sysadmin rolüne veya db_owner yada admin rolüne sahip olmak gerekir. Bunlardan birine sahip olmayan kişinin SY oluşturması için CREATE PROCEDURE izninin bu kişiye verilmiş olması gerekir

Saklı Yordam (SY)
Oluşturmak ve Çalıştırmak - 7

- Bir SY çalıştırmak için EXECUTE veya EXEC deyiminden sonra adını yazmak gerekir

Örnek:

```
USE dukkan
SELECT *
FROM tblUrun
EXEC sp_helpdb
GO
```

Saklı Yordam (SY)
Oluşturmak ve Çalıştırmak - 8

- Örnek: Son bir gün içerisinde verilen siparişlerle ilgili genel bilgileri döndüren SY;

```
CREATE PROC SP$bugunkiSiparisler
AS
SELECT K.isim + ‘ ’ + K.Soyad, K.Email, S.*
FROM tblSiparis S JOIN tblKullanici K
ON K.kullaniciKod = S.KullaniciKod
WHERE siparisTarih < GETDATE()-1 AND siparisTarih>GETDATE()+1
GO
```

Örnekte siparişin kime ait olduğunu gösterebilmek için tblSiparis tablosu tblKullanici tablosu ile birleştirilerek sorgulanmıştır. Bu SY şu şekilde çalıştırılır:

```
EXEC SP$bugunkiSiparisler
```

NOCOUNT oturum parametresi

- Bir oturum parametresi olan NOCOUNT'un açılış ve kapanışı için kullanılması gereken genel ifade şu şekildedir:

SET NOCOUNT { ON | OFF}

- NOCOUNT oturum parametresi, SQL Server'da bir performans artış unsuru olarak kullanılabilir

- SQL Server'da her işlemden sonra etkilenen kayıt sayısını mesaj olarak döndürmek üzere fazladan bir döngü çalıştıracak şekilde ayarlı iken, bu oturum parametresi açıldığında kaç kaydın etkilendiğinin sonuç olarak döndürülmesi ile ilgilenmez

- SQL Server, kaç kaydın etkilendiğini özellikle bir istemci yazılımdan gelen sorgunun sonucu olarak, istemci yazılıma geri bildiriyorsa, bu süreç daha fazla kaynak tüketen bir işlem halini alır. Bu nedenle, performans arttırıcı bir yöntem olarak, kaç kaydın etkilendiği ile ilgilenmiyorsanız, sorgulardan önce oturum parametresi şu şekilde açılır:

SET NOCOUNT ON

Bu komut ile sonuçtan kaç satırın etkilendiği ile ilgili sonucu görmek istediğimizi SQL Server'a bildirmiş oluyoruz

NOCOUNT oturum parametresi -2

Örnek:

```
CREATE PROC SP$bugunkiSiparisler
AS
SET NOCOUNT ON

SELECT K.isim + ‘ ’ + K.Soyad, K.Email, S.*
FROM tblSiparis S JOIN tblKullanici K
ON K.kullaniciKod = S.KullaniciKod
WHERE siparisTarih < GETDATE()-1 AND siparisTarih>GETDATE()+1
GO
```

SY Yönetmek

Örnek:

Bir SY'ı önce public rolüne kapatalım. Sonra webuser isimli kullanıcının bu SY'a erişebilmesini sağlayalım;

```
DENY ON SP$bugunkiSiparisler TO public
GRANT ON SP$bugunkiSiparisler TO webuser
```

SY'larda Değişikli Yapmak

SY'larda değişiklik yapmak için ALTER deyimi kullanılır.

Genel kullanımı;

```
ALTER PROC[EDURE] sakliyordam_adi
[WITH seçenekler]
 AS
     SQL ifadeleri…
  GO
```

SY'larda Değişikli Yapmak - 2

Örnek: Aşağıdaki örnekte SP$gunlukSatis isimli SY'ın kaynak kodu gizlenmektedir.

```
ALTER PROC SP$gunlukSatis
WITH ENCRYPTION
AS
SET NOCOUNT ON
     SELECT COUNT(*)
     FROM tblSiparisDetay SD INNER JOIN tblSiparis S
    ON SD.faturaKOD = S.faturaKOD
    WHERE S.siparisTarih > GETDATE() – 1
    AND S.siparisTarih < GETDATE() +1
SET NOCOUNT OFF
GO
```

Yukarıdaki komutlar çalıştırılmadan önce

sp_helptext 'SP$gunlukSatis'

ifadesi bize SY'ın kaynak kodunu gösterirken, yukarıdaki değişiklikten sonra artık kaynak kodlara sp_helptext komutu ile erişilemeyecektir.

SY’ların Yeniden Derlenmesi

Bir SY’ın her çalıştırılmasında yeniden derlenmesi için şu ifadeler kullanılır;

```
CREATE PROC sakliyordam_adi
WITH   RECOMPILE
AS
      SQL ifadeleri…
GO
```

SY'ların Yeniden Derlenmesi - 2

Örnek:

```
ALTER PROC SP$gunlukSatis
WITH  ENCRYPTION,  RECOMPILE
AS
SET NOCOUNT ON
     SELECT COUNT(*)
     FROM tblSiparisDetay SD INNER JOIN tblSiparis S
     ON SD.faturaKOD = S.faturaKOD
     WHERE S.siparisTarih > GETDATE() – 1
     AND S.siparisTarih < GETDATE() +1
SET NOCOUNT OFF
GO
```

Bu durumda, bu SY için prosedür hafızasında çalıştırma planı tutulmayacaktır

SY'ların Yeniden Derlenmesi - 3

Bazen ilgili tabloda çok değişiklik yapıldığı için, SY'ın derlenerek çalıştırılması istenebilir. Bu durumda SY şöyle çağrılır;

EXECUTE sakliyordam_adi WITH RECOMPILE

(bu durumda, hafızadaki çalıştırma planı silinir, tekrar derleme yapılır)

Tekrar derleme işlemi için sistemdeki SY olan sp_recompile da kullanılabilir;

EXECUTE sp_recompile sakliyordam_adi | tablo_adi

SY’ların Yeniden Derlenmesi - 4

- Örnek:

  tblSiparisDetay tablosunda çok fazla değişiklik yapıldı ve üstündeki tüm SY’ların bir sonraki çalıştırılmada yeniden derlenmesi gerekiyor. Bu durumda;

  ```
  EXEC sp_recompile tblSiparisDetay
  ```

SY'larda Parametreler

- SY'ları daha işlevsel hale getirmek için dışarıdan parametre almaları gerekebilir.

- Girdi parametreleri ile bir SY'ın genel yapısı;

```
CREATE PROC sakliyordam_adi (@parametre_ismi veri-tipi [=default deger])
[WITH seçenekler]
AS
   SQL ifadeleri….
GO
```

SY'larda Parametreler - 2

- Örnek: Bir kullanıcı kodu aldığında bu kullanıcıya ait tblSepet'de yer alan ürünlerin kodunu, adını ve adetini döndüren bir SP;

```
CRETAE PROC SP$sepetim ( @kullaniciKod INT )
AS
SET NOCOUNT ON
    SELECT S.UrunKod, U.UrunAd, S.Adet
    FROM tblSepet S INNER JOIN tblUrun U
            ON U.UrunKod = S.UrunKod
    WHERE S.KullaniciKod = @KullaniciKod
SET NOCOUNT OFF
GO
```

Bu SY'ı iki farklı şekilde çağırabiliriz:

```
SP$sepetim @kullaniciKod=2
```

veya

```
EXEC SP$sepetim 2
```

SY'larda Parametreler - 3

- Bazen dışarıdan gelen parametrenin isteğe bağlı olması gerekebilir. Bu durumda DEFAULT değer atama seçeneği kullanılır. Eğer dışarıdan parametreye değer atanmazsa, geçerli değer default atanmış değer olarak alınır ve işlem yapılır (*Default, bir sabit veya NULL olabilir*)

Örnek : Dışarıdan aldığı bir anahtar kelimeyi, ürünlerin adında, markasında, barkodunda, tanıtım ve özet kısmında geçiyorsa bulan ve listeleyen bir SY şöyle yazılabilir:

```
CREATE PROC SP$urunara (@anahtar VARCHAR(50)=NULL )
AS
SET NOCOUNT ON
IF @anahtar IS NOT NULL
    SELECT urunKod, barKod, UrunAd, u.MarkaKod, Marka
    FROM tblUrun U
            JOIN tblMarka M ON U. MarkaKod=M.markaKod
   WHERE  urunDurum<>0 AND barkod + ' '+urunAd+ ' ' + urunTanitim+ ''+urunOzet LIKE
    '%' + @anahtar + '%'
SET NOCOUNT OFF
```

- Bu SY'ı çağırmak için Management Studio'da ; SP$urunara '1.44' demek yeterlidir

Tablo Tipi Parametre Alan SY'lar

- SY'lara tablo değişkenleri parametre olarak aktarmak SQL Server 2008 ile birlikte gelen bir özelliktir
- Eğer bir SY tablo tipi parametre alacaksa, öncelikle bu tablo tipinin alias tip olarak tanımlanması gerekir.

Örnek: Dukkan kullanıcı arayüzünde kullanıcıların sadece belli markaları seçip bu markalar altındaki ürünleri listelemek istediğini düşünelim. Bu senaryo için uygun marka listesi alan ve bu listeye uygun ürünleri seçen SY şöyle yazılır:

1.adım: Girdi parametre olarak kullanılacak kullanıcı tanımlı tablo tipinin oluşturulması;

```
CREATE TYPE dbo.MarkaTip AS TABLE(markaKod INT, marka VARCHAR(50));
```

2.adım: Bu tipi parametre olarak alacak SY'ın kodlanması;

```
CREATE PROC _SP_BelliMarkaUrunlerSec(@markaTablo AS dbo.MarkaTip READONLY)
AS
BEGIN
    SELECT U.urunKod, U.urunAd, M.marka
FROM tblUrun U JOIN @markaTablo M ON
U.markaKod = M.MarkaKod
END
GO
```

Tablo Tipi Parametre Alan SY'lar - 2

- Örnekteki SY'ın test edilmesi :

  İlk 10 marka ile doldurduğumuz bir MarkaTip türündeki tablo tipi değişkeni SY'a göndererek test edelim.

```
DECLARE @marka AS dbo.MarkaTip;

INSERT INTO @marka (markaKod, marka)
SELECT TOP(9) markaKod, Marka
FROM tblMarka;

-- sonra SY'ı seçelim
EXEC _SP_BelliMarkaUrunlerSec @marka
```

Parametre Geçerliliğini Denetlemek

- "SQL Injection" yöntemi Web'de çok yaygın bir saldırı şeklidir.

- Dışarıdan aldığınız parametrelerin değerlerine farklı şeyler karıştırarak sql ifadelerinizin verdiği hatalardan hareketle sistem hakkında bilgiler toplamaya, hatta daha da ötesi, sizin beklediğiniz parametre yerine SQL ifadeleri göndererek programınızın bu zararlı SQL ifadelerini çalıştırmasını sağlama şeklindeki bir saldırı türüdür

➢ Parametre geçerliliğini denetlemek için örnek:

Sepete ürün ekleme işlemini yerine getirecek bir SY yazalım. Bu SY dışarıdan *kullaniciKod*, *urunKod* ve *adet* parametre alması yeterlidir. Bu işlem için, bu üç parametreyi tblSepet tablosuna eklemek yeterli olacaktır.

Parametre Geçerliliğini Denetlemek - 2

```
CREATE PROC SP$sepeteAt (@kullaniciKod INT=NULL, @urunKod INT=NULL, @adet INT=1)
AS
SET NOCOUNT ON

IF @KullaniciKod IS NULL OR @urunKod IS NULL
        -- kullanıcıKod veya urunKod yoksa bizim için bir anlam ifade etmez
RETURN 0
       -- bütün bilgiler düzgün olarak geldiyse
ELSE IF EXISTS (SELECT * FROM tblSepet WHERE kullaniciKod=@kullaniciKod AND
     urunKod=@urunKod)
    UPDATE tblSepet SET adet=adet+@adet
    WHERE kullanicKod=@kullaniciKod AND urunKod=@urunKod
ELSE
     INSERT INTO tblSepet VALUES (@kullaniciKod, @urunKod, @adet)
SET NOCOUNT OFF
GO
```

Bu SY'ı kullanarak, 2 nolu müşterinin 2668 nolu üründen 1 adet almasını istersek;

SP$sepeteAt 2,2668,1

Aynı ürünü bir daha almasını istersek;

SP$sepeteAt 2,2668,1

ifadesi ile bir önceki eklediği ürünün adet kısmı bir arttırılacaktır

Parametre Geçerliliğini Denetlemek - 2

- Örnekteki SP$sepeteAt SY'ını kullanarak, 2 nolu müşternin 2666 nolu ürünü sepete atma işlemini gerçekleştirelim.

Not: Burada parametre isimleri ile değer aktarması yapılacağı için, sıranın bir önemi yok, önce ürün kodu yazılabilir

EXEC SP$sepeteAt @adet=2, @urunKod=2666, @kullaniciKod=2

Eğer parametrelerden ürün ve müşteri numaraları hakkında bilgi gelmezse, RETURN 0 ile hiçbir işlem yapılmadan SY'dan çıkılır.

SY'larda Çıktı Parametreleri

- Output seçeneği dışarıdan gelen bir değişkenin içerisinin doldurularak dışarıya tekrar gönderilmesi esasına dayalı olarak, dışarıya tekrar parametre göndermede kullanılır
- Bu sayede, bir SY kendisini çağıran başka bir SY veya çalıştırıldığı yere, kendi ürettiği bir çıktı değerini yanıt olarak gönderebilir

- Örmek: 3 adet tamsayı geldiğinde ortalamayı hesaplayıp, OUTPUT parametresi ile ortalamayı döndürecek bir SY;

```
ALTER PROC ortalayici(
   @sayi1 smallint, @sayi2 smallint, @sayi3 smallint, @ortalama  decimal(18,2)
    OUTPUT)
AS
SET NOCOUNT ON
SELECT @ortalama=(@sayi1+@sayi2+@sayi3)/3
SET NOCOUNT OFF
GO
```

SY'larda Çıktı Parametreleri - 2

- Çıkış parametrelerini alabilmek için, SY çağrılmadan SY'ın döndüreceği değerleri getirecek uygun değişkenler tanımlanır
- SY'a bu değişkenler OUTPUT parametre olarak verilir
- SY çalıştırıldığında çıktı parametrelerin değerleri bu değişkenlerden okunabilir

Örneğin, bir önceki örnekte SY'dan dönen değeri alıp gösterelim:

```
DECLARE @sonuc DECIMAL(18,2)
EXEC ortalayici 1,2,4, @sonuc OUT
SELECT @sonuc
```

Çıktı parametre adı (@ortalama) ve SY çıktı parametresi olarak gönderilen değişken (@sonuc) aynı adda olmak zorunda değildir.

Aynı SY'ı şöyle de çağırabilirdik;

```
DECLARE @sonuc smallint
EXEC ortalayici @sayi1=1, @sayi2=2,  @sayi3=4, @ortalama=@sonuc OUT
SELECT 'ORTALAMA', @sonuc
```

SY'lar ve RETURN deyimi

- SY'ın bir tek tamsayı değer döndürmek için etkin bir seçenek olarak RETURN ifadesi kullanılabilir. Bu şekilde output ifadelerini hem SY içinde, hem de SY'ın çağrıldığı yerde tanımlamak zorunda kalmadan, doğrudan değer dışarıya iletilmiş olur.
- 0 ile -99 arasında değerler sistem bazlı durumları belirlemek için ayrılmış kodlardır
- Sıfır değeri SY'ın çalışmasında bir hata oluşmadığını belirtir

Örnek: Dışarıdan gelen 3 sayının toplamını ve ortalamasını çıktı parametre olarak dışarıya verecek şekilde önceki SY'ı değiştirelim.

```
ALTER PROC ortalayici(
            @sayi1 smallint,  @sayi2 smallint, @sayi3 smallint,
            @ortalama  decimal(18,2) OUTPUT)
AS
SET NOCOUNT ON
SELECT @ortalama=(@sayi1+@sayi2+@sayi3)/3
return (@sayi1  + @sayi2  + @sayi3)
SET NOCOUNT OFF
GO
```

Burada çıktı parametresi ile ve return parametresi ile gelen değerler farklı yöntemlerle döndürülmüştür

SY’lar ve RETURN deyimi - 2

- Önceki örnekte yer alan SY’ı şöyle çağırabiliriz:

```
DECLARE @sonuc smallint, @toplam smallint
EXEC @toplam=ortalayici  @sayi1=1, @sayi2=2,
        @sayi3=12,
       @ortalama = @sonuc OUTPUT

SELECT ‘ORTALAMA:’,  @sonuc,  ‘TOPLAM:’,  @toplam
GO
```

SY’lar ve RETURN deyimi - 3

Örnek :

Kullanıcı adı ve şifre girildiğinde, kullanıcının giriş işlemlerini yerine getirecek bir SY yazalım.

Kullanıcı giriş yapabilmiş ise giriş seviyesi, son giriş tarihi gibi bilgileri döndürsün.

Kullanıcı giriş yapamadı ise 0 döndürsün

```
CREATE PROC SP$yetkilendir(
      @kullaniciAd VARCHAR(50), @sifre VARCHAR(20), @yetkiSeviye INT=NULL OUT,
     @sonGirisTarih VARCHAR(12)=NULL OUT,  @isim VARCHAR(30)=NULL OUT)
AS
SET NOCOUNT ON
DECLARE @kullaniciKod INT
IF  @kullaniciAd IS NULL OR @sifre IS NULL
RETURN 0
SELECT @kullaniciKod=kullaniciKod, @yetkiSeviye= yetkiSeviye ,
            @sonGirisTarih=CONVERT(VARCHAR(12), sonGirisTarih),
           @isim=isim + ‘  ’ + soyad
FROM tblKullanici
WHERE kullaniciAd=@kullaniciAd AND @sifre=@sifre

IF @kullaniciKod IS NOT NULL  --böyle bir kullanıcı var
BEGIN
  RETURN @kullaniciKod
  UPDATE tblKullanici  --son giriş tarihi ve şifresini değiştirelim
 SET sonGirisTarih=GETDATE(), girisSayisi=girisSayisi+1
WHERE kullaniciAd=@kullaniciAd  AND  @sifre=@sifre
END
ELSE
RETURN 0
SET NOCOUNT OFF
GO
```

SP$yetkilendir isimli SY'ı kullanmak için önce değişkenler tanımlanıp, sonra değişkenlerle bu SY çağrılıp, değişkenlerin değerleri şu şekilde güncellenebilir

```
DECLARE @kullaniciKod INT,
@yetkiSeviye INT,
@sonGirisTarih VARCHAR(12),
@isim VARCHAR(50)

EXEC @kullaniciKod=SP$yetkilendir
@kullaniciAd='aliveli'
@sifre='123',
@yetkiSeviye=@yetkSeviye OUT,
@sonGirisTarih = @sonGirisTarih OUT,
@isim = @isim OUT

SELECT @kullaniciKod,
@yetkiSeviye,
@sonGirisTarih
@isim
```

Bu SY hem output parametreler ile hem de RETURN deyimi ile aynı anda sonuç döndürmektedir.

Burada kullaniciKod değeri RETURN deyimi ile döndürülmektedir

NOT :

- Her SY dışarıdan parametre almayabilir.
- Bazı SY'lar çıkış parametresi ile veri gönderebilir. Bunu belirtmek için OUTPUT deyimi kullanılır
- SY'lar trigger (tetikleyici) gibi otomatik olarak devreye girmezler, bir uygulamadan veya script tarafından çağrılarak çalıştırılmaları gerekir
- SY'ların gönderdiği değerler doğrudan kullanıcıya yöneltilmiştir. Sonuçları veritabanı motorunun içerisinde kalmadığından, SY'ları bir sorgu içinde kullanamayız. Ancak istisna olarak, SY sonuçlarını var olan bir tabloya ekleyebiliriz

SY ile kullanıcı tanımlı fonksiyonlar arasındaki fark; SY'ların bir sorgunun parçası olarak kullanılamamasıdır. Fakat , bu konuda bir istisna; SY'lardan dönen bir sonuç bir INSERT ifadesinin kayıt ekleme işleminde kullanılabilir. (Aksi halde kayıtları her seferinde var olan tabloya ekleyip yeninden sorgulamanın büyük maliyeti vardır.) Bunun için genel ifade şu şekildedir:

```
INSERT INTO tablo_ismi
EXEC sp_ismi
```

Kullanıcı Tanımlı Fonksiyonlar

Kullanıcı Tanımlı Fonksiyonlar

- Kullanıcı tanımlı fonksiyonlar, SQL Server 2000 ile gelen özelliklerdendir
- Fonksiyonlar tek bir değer veya bir tablo döndürmek için kullanılabilir
- Kullanıcı tanımlı fonksiyonlar, SY'lar gibi dışarıdan parametre alabilirler ve IF-ELSE gibi kontrol ifadeleri içerebilirler
- Kullanıcı tamımlı fonksiyonlar VIEW gibi tablo şeklinde kayıt döndürebilirler

Kullanıcı Tanımlı Fonksiyonları
View ve SY'lardan ayıran özellikler

- Dışarıdan parametre alan view tanımlanamaz.
  Fakat kullanıcı tanımlı fonksiyonlar dışarıdan parametre alabilir

- SY'lar bir sorgunun parçası olarak (istisnai bir durum dışında) çalışamazlar.
  Fakat kullanıcı tanımlı fonksiyonlar sorgunun içerisinde kullanılabilirler

Kullanıcı Tanımlı Fonksiyonlar ile neler yapılabilir?

- Bir SELECT ifadesi içeren SY'ı başka bir sorgunun FROM, IN, WHERE gibi kısımlarında kullanmamız gerektiğinde kullanıcı tanımlı fonksiyonlardan yararlanılabilir
- Bir tek SELECT ifadesinin üretemediği bir View ihtiyacı olduğunda kullanıcı tanımlı fonksiyonlardan yararlanılabilir
- SQL Server'da tanımlı olmayan bir fonksiyona gerek duyulduğunda kullanıcı tanımlı fonksiyonlardan yararlanılabilir

Skaler Kullanıcı Tanımlı Fonksiyonlar

- Skaler fonksiyon, bir tek değer döndüren fonksiyondur

```
CREATE FUNCTION [sahip-ismi.] FONKSIYON_ADI
            ( [@parametre-ismi  degisken-tipi [=standart-deger] )
RETURNS skaler-veri-tipi
[WITH seçenekler]
AS
BEGIN
        Fonksiyonda yapılacak işlemler
        RETURN skaler değer veya değişken
END
```

Skaler Kullanıcı Tanımlı Fonksiyonlar - 2

Örnek : Bir müşteri kodu girildiğinde, bu müşterinin sepetinde kaç ürün olduğunu bulan bir fonksiyon

```
CREATE FUNCTION dbo.SEPETTEKI_URUN_SAYISI(@kullaniciKod  INT=NULL)
RETURNS int
WITH ENCRYPTION
AS
BEGIN
    DECLARE @urunSayisi INT
    SELECT @urunSayisi=SUM(adet)
FROM tblSepet
    WHERE kullaniciKod=@KullaniciKod
    RETURN @urunSayisi
END
```

Burada RETURNS deyimi ile int tipi değişken tanımlanıyor. Çünkü, SELECT ifadesi 1x1'lik bir sonuç döndürse bile, skaler fonksiyonlar bu değeri dışarı döndüremez. Bu nedenle bir int değişken kullanılması zorunluluktur.

Burada RETURN ile değişkene sadece tek değerden ibaret bir sonuç aktarılmakta ve skaler değişkenin değeri de dışarı döndürülmektedir

Skaler Kullanıcı Tanımlı Fonksiyonlar - 3

- “SEPETTEKI_URUN_SAYISI” isminde tanımladığımız fonksiyonu bir sorguda kullanmak istersek;

Örneğin, sepetinde 10’dan fazla ürün bulunan müşterilerimizi bu fonksiyonu kullanarak oluşturacağımız bir sorgu ile bulalım :

```
SELECT kullaniciAd, isim, soyad
FROM tblKullanici
WHERE SEPETTEKI_URUN_SAYISI(kullaniciKod)>10
```

Skaler Kullanıcı Tanımlı Fonksiyonlar - 4

Örnek: Öğrencinin adını ve soyadını büyük harfle yazan bir skaler fonksiyon

```
CREATE FUNCTION OGRADSOYAD(@OGRAD Varchar(20), @OGRSOYAD Varchar(20))
RETURNS Varchar(40)
BEGIN
RETURN (UPPER(@OGRAD)+ ‘ ’+ UPPER(@OGRSOYAD))
END
```

Bu fonksiyon iki parametre almaktadır. Ad ve soyadı birleştirerek tek bir değer olarak döndürmektedir