

KULLANICI TANIMLI FONKSİYONLAR

14

Kullanıcı Tanımlı Fonksiyonlar (*User Defined Functions*), sorgu tekrarlarını önlemek amacı ile iş parçacıkları oluşturmak için kullanılır.

Kullanıcı tanımlı fonksiyonlar, dışarıdan parametre alabilir, **IF ELSE** gibi akış kontrol ifadeleri içerebilirler. Parse edilir, derlenir ve tampon hafızadan çağrılabilirler. Stored Procedure ve view nesnelere benzerler. Bir view gibi **SELECT** sorgularında kullanılabilir. Bir view ile parametrelili işlem yapamazsınız, ancak bu KTF ile mümkündür. Bir Stored Procedure'den alınan sonucu **SELECT** sorgunuzda etkin olarak kullanamazsınız, ancak KTF ile mümkündür.

Kullanıcı tanımlı fonksiyonlar, view'lerin esneklik ve kullanılabilirliği ile Stored Procedure'lerin parametre kullanabilme, tampon hafızadan çağrılabilme, parse edilme, derlenme gibi bir çok mimari yeteneklerinin birleşimi olarak düşünülebilir.

Bir KTF, veritabanında veri seçme işlemlerini gerçekleştirmek üzere iş parçacığı olarak geliştirilir. KTF çalışmasını tamamladıktan sonra, kayıtlar üzerinde herhangi bir değişiklik (side-effect / yan etki) yapmamış olması gerekir.

Kullanıcı Tanımlı Fonksiyonlar ile neler yapılabilir?

- Sürekli gerçekleştirilen işlemler fonksiyonel hale getirilebilir.
- SQL Server fonksiyonel hale getirilebilir. SQL Server tarafından desteklenmeyen bir fonksiyon geliştirilebilir. Örneğin, doğum tarihi ve şu anki tarihi vererek yaş hesaplatma işlemi bir fonksiyon geliştirilerek yapılabilir.

Ya da kendi algoritmanıza göre bir metin şifreleme işlemi gerçekleştirmek için fonksiyon geliştirebilirsiniz. Hesap makinesi gibi işlemler yapan bir fonksiyon ya da $\pi()$ sayısı ile işlem yapmaya yarayan bir fonksiyon iyi birer örnek olabilir.

- Bu nesneler T-SQL ile geliştirilebileceği gibi, CLR ile de geliştirilebilirler.

KULLANICI TANIMLI FONKSİYON ÇEŞİTLERİ

KTF nesneleri fonksiyoneldir. Bir KTF, geri dönüş tipi olarak **INT**, **VARCHAR**, **DATETIME** gibi skaler veri tiplerini döndürebileceği gibi, bir tablo tipli değişken de döndürebilir.

KTF, **SELECT** işlemleri için geniş ve işlevsel yeteneklere sahiptir. KTF nesneleri iki ana başlıkta inceleyeceğiz.

- Skaler Kullanıcı Tanımlı Fonksiyonlar
- Tablo Kullanıcı Tanımlı Fonksiyonlar

SKALER KULLANICI TANIMLI FONKSİYONLAR

Skaler fonksiyon, bir tek değer döndüren fonksiyondur. Birden fazla parametre alabilir, ancak sonuç olarak tek bir değer döndürür. SQL Server içerisinde kullanılan **GETDATE()** bir sistem fonksiyonudur. **GETDATE()** fonksiyonu, o anki sistem zaman bilgisini alarak geriye sadece bu bilgiyi döndürür. Bu özellik sistem tarafında sistem fonksiyonlarında kullanılabileceği gibi, T-SQL geliştiricileri tarafından da bu tür örnek fonksiyonlar geliştirilebilir.

π sayısı en kısa haliyle 3,14 olarak kabul edilir. π sayısı ile işlem yapılması gerektiğinde, tanımlayacağınız bir $\pi()$ fonksiyonu içerisine gerekli parametreleri göndererek, fonksiyon içerisinde 3,14 değerini kullanarak işlem yapar ve sonucu tek değer olarak döndürebilirsiniz.

Geliştirilen uygulamada π sayısı kullanmak isteniyor olabilir. Bize π sayısını döndürecek bir fonksiyon geliştirelim.

```
CREATE FUNCTION Pİnedir()
RETURNS NUMERIC(5,2)
AS
BEGIN
    RETURN 3.14;
END;
```

KTF, numeric geri dönüş tipine sahip ve herhangi bir parametre almadan geriye 3.14 değerini döndürecektir. KTF sonucunu, başka bir sorgu içerisinde kullanabileceğiniz gibi tek olarak da kullanabilirsiniz.


PIInedir() fonksiyonunu tek olarak çağırmak için;

```
SELECT dbo.PIInedir();
```

(No column name)	
1	3.14

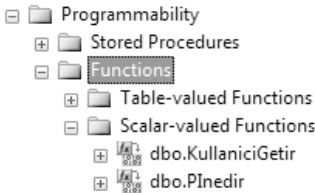
ya da

```
PRINT dbo.PIInedir();
```

 Messages
3.14

İki yöntemle de çağrılabilen fonksiyonun adından önce kullanılan şema adı (dbo) dikkatinizi çekmiş olmalıdır. Şema adı olmadan çalıştırdığınızda sorgunuz hata üretecektir. KTF kullanırken şema adını yazmalısınız.

PIInedir() fonksiyonunu SSMS aracından da görebilir ve yönetebiliriz.



PI sayısına ihtiyacınız olduğunda, sistem fonksiyonları içerisindeki **PI()** fonksiyonunu kullanabilirsiniz.

```
SELECT PI();
```

(No column name)	
1	3.14159265358979

KTF, genel olarak veritabanında bir tablo üzerinden sorgu gerçekleştirmek için kullanılır. Tablo kullanarak bir işlem gerçekleştirelim.

Ürünler tablosunda ne kadar ürün olduğunu hesaplayalım.

```
CREATE FUNCTION dbo.UrunToplamSayi()
RETURNS INT
AS
BEGIN
    DECLARE @toplam INT;
    SELECT @toplam = COUNT(ProductID) FROM Production.Product;
    RETURN @toplam;
END;
```

Oluşturulan KTF nesnesini çağıralım.

```
SELECT dbo.UrunToplamSayi();
```

	(No column name)
1	505

KTF, herhangi bir parametre almadan da çalışabilir. Ancak genel kullanım ve en güçlü olduğu alan tabi ki parametrelili kullanımıdır.

Kullanıcının **BusinessEntityID** değerini alarak, ad ve soyad bilgilerini birleştirip geri döndürelim.

```
CREATE FUNCTION dbo.KullaniciGetir(@KullnıcıKod INT = NULL)
RETURNS VARCHAR(100)
AS
BEGIN
    DECLARE @ad_soyad VARCHAR(100)
    SELECT @ad_soyad = FirstName + ' ' + LastName
    FROM Person.Person WHERE BusinessEntityID = @KullnıcıKod
    RETURN @ad_soyad
END;
```

BusinessEntityID değeri 1 olan kaydı listeleyelim.

```
SELECT dbo.KullaniciGetir(1);
```

	(No column name)
1	Ken Sánchez

Fonksiyonu inceleyelim,

KullaniciGetir() fonksiyonuna dışarıdan **INT** veri tipinde bir **BusinessEntityID** değeri alıyoruz. Fonksiyon içerisinde gerçekleşecek işlemler sonucunda kullanıcının ad ve soyad bilgilerini birleştirerek geri döndüreceğiz. Bu nedenle, fonksiyonun geri dönüş tipi olarak **VARCHAR(100)** belirledik. Geri dönüş tipini belirtmek için **RETURNS** komutunu kullandık. Daha sonra işlemlerimizi gerçekleştirmek için **BEGIN ... END** blokları arasında değişken tanımlama ve **SELECT** sorgumuzu yazıyoruz. Bu sorgu sonucunda dönen değeri **@ad_soyad** değişkenine atayarak **RETURN** ile fonksiyon dışına gönderiyoruz.

TÜRETİLMİŞ SÜTUN OLARAK SKALER FONKSİYON

Skaler fonksiyonlar tablolarda türetilmiş sütun olarak kullanılabilir.

Daha önce oluşturduğumuz **dbo.KullaniciGetir()** fonksiyonunu, bir **SELECT** sorgusu içerisinde türetilmiş sütun olarak kullanalım.

```
SELECT
    BusinessEntityID, PersonType, Title,
    dbo.KullaniciGetir(BusinessEntityID) AS AdSoyad
FROM Person.Person;
```

	BusinessEntityID	PersonType	Title	AdSoyad
1	1	EM	NULL	Ken Sánchez
2	2	EM	NULL	Terri Duffy
3	3	EM	NULL	Roberto Tamburello
4	4	EM	NULL	Rob Walters
5	5	EM	Ms.	Gail Erickson
6	6	EM	Mr.	Jossef Goldberg
7	7	EM	NULL	Dylan Miller

Fonksiyonun **Person.Person** tablosu içerisinde bir sütun olarak yer almasını istersek;

```
ALTER TABLE Person.Person
ADD AdSoyad AS dbo.KullaniciGetir(BusinessEntityID);
```

TABLO DÖNDÜREN

KULLANICI TANIMLI FONKSİYONLAR

Tablo değeri döndüren fonksiyonların skaler fonksiyonlardan farkı, geriye tek bir değer değil, tablo tipinde değer döndürmesidir.

Bu fonksiyonlar parametre alabildiği gibi, içerisinde bir tablo yapısı oluşturularak, oluşturulan bu tabloyu **RETURN** ile geri döndürebilir.

Tablo döndüren fonksiyonlar kendi içerisinde ikiye ayrılır.

- Satırdan Tablo Döndüren Fonksiyonlar (*Inline Table-Valued Functions*)
- Çoklu İfade İle Tablo Döndüren Fonksiyonlar (*Multi-Statement Table-Valued Functions*)

SATIRDAN TABLO DÖNDÜREN FONKSİYONLAR

Tablo döndüren fonksiyonlar, view gibi sorgulanabilen, Stored Procedure gibi parametre alan KTF nesneleridir.

Basit bir ürün arama fonksiyonu geliştirelim.

```
CREATE FUNCTION fnc_UrunAra(
    @ara VARCHAR(10)
) RETURNS TABLE
AS
RETURN SELECT * FROM Production.Product
WHERE Name LIKE '%' + @ara + '%';
```

Fonksiyonu parametre ile çağırılım.

```
SELECT * FROM dbo.fnc_UrunAra('Be');
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	327	Down Tube	DT-2377	1	0	NULL	800	600	0,00	0,00
4	398	Handlebar Tube	HT-2981	1	0	NULL	800	600	0,00	0,00
5	399	Head Tube	HT-8019	1	0	NULL	800	600	0,00	0,00
6	847	Headlights - Dual-Beam	LT-H902	0	1	NULL	4	3	14,4334	36,3896
7	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00

Yukarıdaki fonksiyona gönderilen 'Be' parametre değeri, fonksiyon içerisinde **LIKE** ile aranacak ve bulunan sonuçlar listelenecektir.

ÇOKLU İFADE İLE TABLO DÖNDÜREN FONKSİYONLAR

Bu fonksiyon türünün diğerlerinden farkı, içerisinde geriye değer dönmek için oluşturulan tablo tipindeki değişkene çoklu veri ekleme işlemi gerçekleştirilebilecek olunmasıdır.

En sık kullanılan sorgulardan biri, belirli aralıklardaki değerleri listelemektir. **ProductID** değeri 100 ile 500 arasında olan kayıtları listelemek buna bir örnek olabilir.

```
CREATE FUNCTION dbo.BelirliAraliktakiUrunler(@ilk INT, @son INT)
RETURNS @values TABLE
(
    ProductID INT,
    Name VARCHAR(30),
    ProductNumber VARCHAR(7)
)
```

```

AS
BEGIN
    INSERT @values
        SELECT ProductID, Name, ProductNumber
        FROM Production.Product
        WHERE ProductID >= @ilk AND ProductID <= @son
    RETURN
END;

```

ProductID değeri 1 ile 4 arasında olan ürünleri listeleyelim.

```
SELECT * FROM dbo.BelirliAraliktakiUrunler(1, 4);
```

	ProductID	Name	ProductNumber
1	1	Adjustable Race	AR-5381
2	2	Bearing Ball	BA-8327
3	3	BB Ball Bearing	BE-2349
4	4	Headset Ball Bearings	BE-2908

İki parametre alan bir KTF oluşturalım. İlk parametre de, aralarında virgüller bulunan sayılar, ikinci parametrede ise bu virgüller ile ayrılan sayıları virgüllerden temizleyerek satır satır listelemek için kullanılacak bir ayıraç bulunsun.

```

CREATE FUNCTION dbo.IntegerAyirici(@liste VARCHAR(8000),
                                   @ayirac VARCHAR(10) = ',')
RETURNS @tabloDeger TABLE
(
    [Parça] INT
)
AS
BEGIN
    DECLARE @parca VARCHAR(255)
    WHILE (DATALENGTH(@liste) > 0)
    BEGIN
        IF CHARINDEX(@ayirac, @liste) > 0
        BEGIN
            SELECT @parca = SUBSTRING(@liste,1,(CHARINDEX(@ayirac,
@liste)-1))
            SELECT @liste = SUBSTRING(@liste,(CHARINDEX(@ayirac, @liste)

```

```

+ DATALENGTH(@ayirac)), DATALENGTH(@liste))
    END
ELSE
    BEGIN
        SELECT @parca = @liste
        SELECT @liste = NULL
    END
    INSERT @tabloDeger([Parça])
    SELECT [Parça] = CONVERT(INT, @parca)
END
RETURN
END;

```

Fonksiyonu kullanmak için;

```
SELECT * FROM dbo.Integer_Ayirici('10, 20, 30, 300, 423, 156, 983', ',');
```

KTF içerisinde, tablo geri dönüş veri tipindeki nesneye birden fazla sorgu ile alınan kayıtları ekleyerek listeleyelim.

Person.Person tablosu içerisinde **PersonType** sütununa göre filtrelemeler gerçekleştirerek istediğimiz **PersonType** değerine sahip kayıtları listeleyelim.

Bu fonksiyon içerisinde 5 ayrı filtreden oluşan sonuç listesini birleştirerek tek bir sonuç kümesi haline getireceğiz.

	Parça
1	10
2	20
3	30
4	300
5	423
6	156
7	983

```

CREATE FUNCTION dbo.PersonTypePerson(@pt_sp VARCHAR(2), @pt_sc VARCHAR(2),
@pt_vc VARCHAR(2), @pt_in VARCHAR(2), @pt_gc VARCHAR(2))
RETURNS @PersonTypeData TABLE
(
    BusinessEntityID INT,
    PersonType VARCHAR(2),
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
)
AS
BEGIN
    INSERT @PersonTypeData
        SELECT BusinessEntityID, PersonType, FirstName, LastName

```



```

FROM Person.Person
WHERE PersonType = @pt_sp
INSERT @PersonTypeData
    SELECT BusinessEntityID, PersonType, FirstName, LastName
FROM Person.Person
WHERE PersonType = @pt_sc
INSERT @PersonTypeData
    SELECT BusinessEntityID, PersonType, FirstName, LastName
FROM Person.Person
WHERE PersonType = @pt_vc
INSERT @PersonTypeData
    SELECT BusinessEntityID, PersonType, FirstName, LastName
FROM Person.Person
WHERE PersonType = @pt_in
INSERT @PersonTypeData
    SELECT BusinessEntityID, PersonType, FirstName, LastName
FROM Person.Person
WHERE PersonType = @pt_gc
RETURN
END;

```

Bu sorguyu daha dinamik bir şekilde geliştirerek dışarıdan gelen parametrenin daha esnek olmasını sağlayabilirdik. Ancak, KTF çoklu ifade ile tablo döndürme mantığını kavrayabilmek için, fazlalıklardan arındırılmış olması gerektiğini düşünüyorum.

Oluşturduğumuz fonksiyonu çağıralım.

```
SELECT * FROM dbo.PersonTypePerson('SP','SC','VC','IN','GC');
```

	BusinessEntityID	PersonType	FirstName	LastName
1	274	SP	Stephen	Jiang
2	275	SP	Michael	Blythe
3	276	SP	Linda	Mitchell
4	277	SP	Jillian	Carson
5	278	SP	Garrett	Vargas
6	279	SP	Tsvi	Reiter
7	280	SP	Pamela	Ansman-Wolfe

KULLANICI TANIMLI FONKSİYONLARDA KOD GİZLİLİĞİ: ŞİFRELEMEK

View ve Stored Procedure'ler de kullanılabilen kaynak kod şifreleme özelliği KTF nesneleri için de geçerlidir. Bir KTF şifrelemek de diğer nesneler gibi kolaydır.

Tüm fonksiyonların kaynak kodunun şifrlenmesine gerek yoktur. Şifrelemenin gerekli olması için, işlem yapılacak veri ve sütunların kritik öneme sahip olması ön görülür. Örneğin, ülke bilgilerinin bulunduğu bir tablo ve içerdiği veriler kritik bir öneme sahip değildir. Ancak kullanıcı bilgileri, istatistik, raporlama, uygulama algoritmaları gibi önemli sayılacak veriler kritik bilgilerdir.

Daha önce geliştirdiğimiz, kullanıcı ad ve soyadını getiren fonksiyonun kaynağını şifreleyelim.

```
ALTER FUNCTION dbo.KullaniciGetir(@KullniciKod INT = NULL)
RETURNS VARCHAR(100)
WITH ENCRYPTION
AS
BEGIN
    DECLARE @ad_soyad VARCHAR(100)
    SELECT @ad_soyad = FirstName + ' ' + LastName
    FROM Person.Person WHERE BusinessEntityID = @KullniciKod
    RETURN @ad_soyad
END;
```

Şifreleme işlemi için eklediğimiz tek kod **RETURNS** komutundan sonra kullanılan **WITH ENCRYPTION** ifadesidir.

Şifrelemenin gerçekleştiğini test edelim.

```
EXEC sp_helptext 'dbo.KullaniciGetir';
```



Messages

The text for object 'dbo.KullaniciGetir' is encrypted.

Yeni oluşturulacak (**create**) bir fonksiyon için de aynı yöntem kullanılır.

DETERMİNİZM

Determinizm kavramı, **Deterministic** ve **Nondeterministic** olmak üzere ikiye ayrılır.

Aldığı aynı parametreler için aynı sonucu döndüren fonksiyonlar Deterministic'tir. Örneğin; 3 parametre alan ve aldığı parametreleri toplayarak geri döndüren bir fonksiyon Deterministic'tir. Çünkü aynı 3 parametreyi tekrar aldığı anda aynı sonucu tekrar üretecektir. PI sayısını döndürecek bir fonksiyon da aynı şekilde Deterministic'tir.

Her çalışmasında farklı sonuç üreten fonksiyonlar Nondeterministic'tir. Sistem saatini döndüren `GETDATE()` fonksiyonu buna örnek gösterilebilir. Çünkü her çalışmasında saniye değeri aynı olsa bile, salise değeri farklı bir değer üretecektir. Bu tür fonksiyonlara örnek olarak `GUID` ve `NEWID` fonksiyonları da verilebilir. `GUID` ve `NEWID` her çalıştırılmada farklı değerler üretirler.

Nondeterministic fonksiyonlarda küçük ama önemli bir farklılık vardır. Kimi fonksiyon her sorgu için bir kez çalışarak sonuç üretirken, kimi fonksiyon da sorgu içerisindeki her iş parçası için farklı değerler üretir.

`Production.Product` tablosunu inceleyelim. Bu tablodaki `rowguid` sütunu `GUID`, `ModifiedDate` sütunu ise `GETDATE` isimli Nondeterministic sistem fonksiyonları ile üretilen değerlere sahiptir.

```
SELECT rowguid, ModifiedDate FROM Production.Product;
```

	rowguid	ModifiedDate
1	694215B7-08F7-4C0D-ACB1-D734BA44C0C8	2008-03-11 10:01:36.827
2	58AE3C20-4F3A-4749-A7D4-D568806CC537	2008-03-11 10:01:36.827
3	9C21AED2-5BFA-4F18-BCB8-F11638DC2E4E	2008-03-11 10:01:36.827
4	ECFED6CB-51FF-49B5-B06C-7D8AC834DB8B	2008-03-11 10:01:36.827
5	E73E9750-603B-4131-89F5-3DD15ED5FF80	2008-03-11 10:01:36.827
6	3C9D10B7-A6B2-4774-9963-C19DCEE72FEA	2008-03-11 10:01:36.827
7	EABB9A92-FA07-4EAB-8955-F0517B4A4CA7	2008-03-11 10:01:36.827

Sorgu sonucuna bakıldığında ise, `rowguid` sütun değerlerinin tamamının benzersiz olduğunu, `ModifiedDate` sütunlarının ise aynı değerlere sahip olduğunu görebiliyoruz.

Bu durumun sebebi Nondeterministic fonksiyonlar arasındaki bu farklılıktır.

Bu tablo ve içerisindeki veriler oluşturulurken kullanılan script, tek seferde ve aynı anda çalıştırıldı. Sorgu ilk çalışırken o anın sistem tarihini alan **GETDATE** fonksiyonu tüm **INSERT** işlemleri için aynı zaman değerini kullandı. Ancak rowguid sütunu için kullanılan **GUID** fonksiyonu ise satır bazlı çalıştığından dolayı, her satır için ayrı benzersiz değer üreterek her **INSERT** işleminde yeni bir değer üretti. Bu nedenle tüm rowguid sütun değerleri farklıdır.

Ayrıca bazı Nondeterministic fonksiyonları bir skaler fonksiyon içerisinde doğrudan kullanılamaz. Örneğin, random sayısal değer üretmek için kullanılan ve her defasında farklı değer üreten **RAND** fonksiyonu bir skaler fonksiyon içerisinde doğrudan kullanılamayacaktır.

```
SELECT RAND();
```

	(No column name)
1	0.457724701516271

Fonksiyon oluşturalım.

```
CREATE FUNCTION dbo.fnc_Rand() -- Hatalı Fonksiyon
RETURNS FLOAT
AS
BEGIN
    RETURN RAND()
END;
```

fnc_Rand isimli fonksiyonu oluşturmak isterken hata ile karşılaştık.

Ancak bu fonksiyonu bir view içerisinde kullanarak, fonksiyon içerisinde de bu view'i çağırdığımızda herhangi bir hata ile karşılaşmayız.

RAND fonksiyonunu içerisinde kullanacağımız view'i oluşturalım.

```
CREATE VIEW dbo.vw_Rand
AS
SELECT RAND() AS RANDOM;
```

Oluşturduğumuz view'i fonksiyon içerisinde kullanalım.

```
CREATE FUNCTION dbo.fnc_Rand()
RETURNS FLOAT
AS
BEGIN
    RETURN (SELECT * FROM dbo.vw_Rand)
END;
```

Fonksiyon başarılı bir şekilde oluşturuldu. Şimdi fonksiyonu çağırabiliriz.

```
SELECT dbo.fnc_Rand() AS RANDOM;
```

	RANDOM
1	0,674615548219105

SCHEMA BINDING

KTF nesneleri oluştururken fonksiyon içerisinde kullanılan nesnelerin, ilişkili tablolardan değiştirilmesi ya da silinmesi gibi fonksiyonun işleyişini engelleyecek işlemlerden korumak için **WITH SCHEMA BINDING** kullanılır.

fnc_UrunAra isimli fonksiyonu **ALTER** ile değiştirerek fonksiyon üzerinde **SCHEMABINDING** uygulayalım.

```
ALTER FUNCTION fnc_UrunAra(@ara VARCHAR(10))
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT ProductID, Name FROM Production.Product
WHERE Name LIKE '%' + @ara + '%';
```

SCHEMABINDING ile **ENCRYPTION** özelliğini tek satırda belirtmek için;
WITH SCHEMABINDING, ENCRYPTION

TABLolarla TABLO TİPİ FONKSİYONLARI BİRLEŞTİRMEK

Fonksiyonlar, view ve Stored Procedure'lerin bazı özelliklerini almıştır. View içerisinde **JOIN** kullanmak mümkünse de fonksiyonlar ile **JOIN** kullanımında yöntem biraz farklıdır. Yani fonksiyon ile tablonun ilişkili kullanılabilmesi, birleştirilebilmesi için **CROSS APPLY** ve **OUTER APPLY** kullanılır.

CROSS APPLY ve **OUTER APPLY** operatörlerini örneklemek için iki tablo oluşturalım.

```

Departman bilgilerini tutan Departments;
CREATE TABLE Departments(
    DepartmentID int NOT NULL PRIMARY KEY,
    Name VARCHAR(250) NOT NULL,
) ON [PRIMARY];

```

Çalışan bilgilerini tutan Employees;

```

CREATE TABLE Employees(
    EmployeesID int NOT NULL PRIMARY KEY,
    FirstName VARCHAR(250) NOT NULL,
    LastName VARCHAR(250) NOT NULL,
    DepartmentID int NOT NULL REFERENCES Departments(DepartmentID),
) ON [PRIMARY];

```

Oluşturduğumuz tablolara kayıt girelim.

Departmanlar;

```

INSERT Departments (DepartmentID, Name)
VALUES (1, N'Mühendislik'), (2, N'Yönetim'), (3, N'Satış'),
(4, N'Pazarlama'), (5, N'Finans')

```

Çalışanlar;

```

INSERT Employees (EmployeesID, FirstName, LastName, DepartmentID)
VALUES (1, N'Kerim', N'Fırat', 1 ), (2, N'Cihan', N'Özhan', 2 ),
(3, N'Emre', N'Okumuş', 3 ), (4, N'Barış', N'Özhan', 3 );

```

Eklediğimiz kayıtları listeleterek inceleyelim.

```

SELECT * FROM Employees;

```

	EmployeesID	FirstName	LastName	DepartmentID
1	1	Kerim	Fırat	1
2	2	Cihan	Özhan	2
3	3	Emre	Okumus	3
4	4	Baris	Özhan	3

```
SELECT * FROM Departments;
```

	DepartmentID	Name
1	1	Mühendislik
2	2	Yönetim
3	3	Satis
4	4	Pazarlama
5	5	Finans

CROSS APPLY

Tablo ve fonksiyon birleştirme işleminde, **INNER JOIN** gibi çalışan komut **CROSS APPLY** operatörüdür.

Normal bir **JOIN** ile yapılan işlemi ve **CROSS APPLY** ile nasıl yapılabileceğini inceleyelim.

JOIN ile;

```
SELECT * FROM Departments D
INNER JOIN Employees E ON D.DepartmentID = E.DepartmentID;
```

	DepartmentID	Name	EmployeesID	FirstName	LastName	DepartmentID
1	1	Mühendislik	1	Kerim	Firat	1
2	2	Yönetim	2	Cihan	Özhan	2
3	3	Satis	3	Emre	Okumus	3
4	3	Satis	4	Baris	Özhan	3

CROSS APPLY ile;

```
SELECT * FROM Departments D
CROSS APPLY
(
    SELECT * FROM Employees E WHERE E.DepartmentID = D.DepartmentID
) DIJIBIL;
```

	DepartmentID	Name	EmployeesID	FirstName	LastName	DepartmentID
1	1	Mühendislik	1	Kerim	Firat	1
2	2	Yönetim	2	Cihan	Özhan	2
3	3	Satis	3	Emre	Okumus	3
4	3	Satis	4	Baris	Özhan	3

İki işlemde de aynı sonucun elde edildiği görülmektedir.

OUTER APPLY

OUTER APPLY operatörü ise **LEFT OUTER JOIN** gibi çalışır.

JOIN ile;

```
SELECT * FROM Departments D
LEFT OUTER JOIN Employees E ON D.DepartmentID = E.DepartmentID;
```

	DepartmentID	Name	EmployeesID	FirstName	LastName	DepartmentID
1	1	Mühendislik	1	Kerim	Firat	1
2	2	Yönetim	2	Cihan	Özhan	2
3	3	Satis	3	Emre	Okumus	3
4	3	Satis	4	Baris	Özhan	3
5	4	Pazarlama	NULL	NULL	NULL	NULL
6	5	Finans	NULL	NULL	NULL	NULL

OUTER APPLY ile;

```
SELECT * FROM Departments D
OUTER APPLY
(
    SELECT * FROM Employees E WHERE E.DepartmentID = D.DepartmentID
) KODLAB;
```

	DepartmentID	Name	EmployeesID	FirstName	LastName	DepartmentID
1	1	Mühendislik	1	Kerim	Firat	1
2	2	Yönetim	2	Cihan	Özhan	2
3	3	Satis	3	Emre	Okumus	3
4	3	Satis	4	Baris	Özhan	3
5	4	Pazarlama	NULL	NULL	NULL	NULL
6	5	Finans	NULL	NULL	NULL	NULL

CROSS APPLY VE OUTER APPLY

OPERATÖRLERİNİN FONKSİYONLAR İLE KULLANIMI

CROSS APPLY ve **OUTER APPLY** operatörlerinin **JOIN**'ler ile benzerliklerini örnekler üzerinde inceledik. Bu operatörlerin fonksiyonlar ile ilgili en önemli özelliği ise fonksiyon ile tablonun birleştirilmesi işlemidir.

Bir departmanda çalışan tüm çalışanları listelemek isteyebiliriz.

Bu işlem için bir KTF oluşturalım.

```
CREATE FUNCTION dbo.fnc_GetAllEmployeeOfADepartment(@DeptID AS INT)
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM Employees E WHERE E.DepartmentID = @DeptID
);
```

Fonksiyonun **CROSS APPLY** ile kullanımı;

```
SELECT * FROM Departments D
CROSS APPLY dbo.fnc_GetAllEmployeeOfADepartment(D.DepartmentID);
```

	DepartmentID	Name	EmployeesID	FirstName	LastName	DepartmentID
1	1	Mühendislik	1	Kerim	Firat	1
2	2	Yönetim	2	Cihan	Özhan	2
3	3	Satis	3	Emre	Okumus	3
4	3	Satis	4	Baris	Özhan	3

Fonksiyonun **OUTER APPLY** ile kullanımı;

```
SELECT * FROM Departments D
OUTER APPLY dbo.fnc_GetAllEmployeeOfADepartment(D.DepartmentID);
```

	DepartmentID	Name	EmployeesID	FirstName	LastName	DepartmentID
1	1	Mühendislik	1	Kerim	Firat	1
2	2	Yönetim	2	Cihan	Özhan	2
3	3	Satis	3	Emre	Okumus	3
4	3	Satis	4	Baris	Özhan	3
5	4	Pazarlama	NULL	NULL	NULL	NULL
6	5	Finans	NULL	NULL	NULL	NULL

KULLANICI TANIMLI FONKSİYONLARIN YÖNETİMİ

KTF nesnelerinin yönetimi T-SQL ya da **SSMS** (*SQL Server Management Studio*) ile yapılabilir. KTF nesnesinin içeriğinin değiştirilmesi, yapısal değişiklikler gibi işlemleri T-SQL ile gerçekleştirilebilir.

Aynı işlemler SSMS ile de gerçekleştirilebilir. SSMS editörü ile yapacağınız düzenleme işlemleri için gene sorgu ekranı açılacak ve oluşturma kodları, başında ALTER komutu ile birlikte listelenecektir. Sorgu yapısının hatırlanması için iyi bir yöntemdir.

KULLANICI TANIMLI FONKSİYONLARI DEĞİŞTİRMEK

KTF nesneleri de diğer nesnelerde olduğu gibi **ALTER** komutu ile düzenlenebilir. SSMS editörü de KTF nesnelerini düzenleyebilecek imkan sağlamaktadır.

T-SQL ile düzenlemek için;

```
ALTER FUNCTION dbo.BelirliAraliktakiUrunler(@ilk INT, @son INT)
RETURNS @values TABLE
(
    ProductID INT,
    Name VARCHAR(30),
    ProductNumber VARCHAR(7),
    ListPrice MONEY
)
AS
BEGIN
    INSERT @values
    SELECT ProductID, Name, ProductNumber, ListPrice
    FROM Production.Product
    WHERE ProductID >= @ilk AND ProductID <= @son
    RETURN
END;
```

Yukarıdaki sorgu ile **dbo.BelirliAraliktakiUrunler()** fonksiyonun düzenleme işlemini gerçekleştirdik.

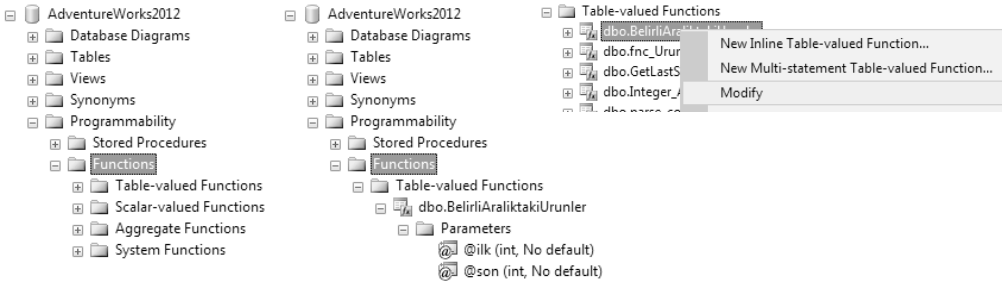
Bu düzenleme işleminde yapılan iki ana işlem var.

- **CREATE FUNCTION** yerine **ALTER FUNCTION** kullanıldı.
- **SELECT** sorgu yapısında değişiklik yapıldı. Bu nedenle, **RETURNS** ile döndürülecek tablo yapısı da düzenlendi.

SELECT sorgusuna **ListPrice** sütununu eklediğimiz için **RETURNS** kısmında da bu sütunu aşağıdaki şekilde tanımladık.

```
...
ListPrice MONEY
...
```

SSMS ile KTF Düzenleme;



SSMS ya da T-SQL ile KTF düzenleyebilmek için, KTF'nin **WITH ENCRYPTION** ile şiflenmemiş olması gerekir. Şiflenmiş bir KTF nenesinin içeriğine ulaşılamayacağı için düzenleme işlemi gerçekleştirilemez.

KULLANICI TANIMLI FONKSİYONLARI SİLMEK

Fonksiyon silmek için **DROP** komutu kullanılır.

```
DROP FUNCTION [dbo].[ BelirliAraliktakiUrunler]
```

T-SQL ile silinebileceği gibi SSMS ile de görsel olarak silinebilir.

SSMS ile silebilmek için;

Programmability > Functions içerisinden ilgili kısımdaki fonksiyona sağ tıklayarak, **Delete** seçeneği ile silebilirsiniz.

