

# Microsoft SQL Server 2005®

## Programcılar Açısından Öne Çıkan Yenilikler

### İçindekiler

- Transact SQL Gelişmeleri
  - Common Table Expressions (CTE)
  - Try-Catch
- .NET ile SQL Server programlama
  - Assembly Yönetimi
  - In-Proc Managed Provider
    - SQL Server için Assembly Oluşturmak
    - Stored Procedure oluşturulması
    - User Defined Function Oluşturulması
  - In-Proc Managed Provider Nesneleri
    - System.Data.SqlServer Namespace Yapısı

Cengiz HAN  
cengiz@cengizhan.com

Not: Doküman Microsoft SQL Server 2005® Beta 2 Express sürümü ile hazırlanmıştır.

Bu doküman [www.cengizhan.com](http://www.cengizhan.com)'dan indirilmiştir.

## Transact Sql Gelişmeleri

Sql Server 2005 ile beraber Transact SQL (T-SQL) dil yapısına yeni anahtar kelimeler eklenmiştir. Bu eklentiler ile beraber T-SQL çok daha güçlü bir sorgulama dili haline gelmektedir. Bu konu başlığı altında T-SQL gelişmelerinden Common Table Expressions ve TRY..CATCH hata kontrol yapısından bahsedilecektir.

### Common Table Expressions

SQL Server 2005 sürümü ile eklenen özelliklerden birisi olan **Common Table Expressions (CTE) (Genel Tablo İfadeleri)**, veritabanında kalıcı olarak tanımlanmayan sadece aktif kapsam içerisinde geçerli olan kayıt setleridir.

İlk bakışta uygun bir benzetme yapacak olursak CTE veritabanına yazılmamış, geçici View nesnelere benzerler. CTE'ler bir view veya tablo gibi sorguda FROM ifadesinden sonra kullanılırlar. Tanımlanmış olan bir CTE'a aktif kapsam içerisinde istediğiniz sayıda erişim yapabilirsiniz.

CTE Tanımlama Yapısı

```
WITH <ctelsmi>(<sutunlar>)  
AS  
(  
  <cteSorgu>  
)
```

CTE'in Kullanımı

```
SELECT * FROM <ctelsmi>
```

CTE'ların sağladığı en önemli getiri "kendi kendini çağırır" (recursive) sorguların oluşturulabilmesindeki esneklik ve kolaylıktır. CTE'ları kullanarak yapılabilecek diğer (kendini kendini çağırır sorgular dışındaki) işlerin hemen hemen hepsini CTE'ları kullanmadan da yapabilirsiniz. Bu durumlarda CTE'lar sadece kodlamanızın kalitesini ve okunabilirliğini yükseltecektir.

## Try Catch Yapısı

SQL Server'ın önceki sürümü olan SQL Server 2000'de hata kontrolü yapmak için @@ERROR ifadesinin değeri kontrol edilirdi. Bu şekilde yapılan kontrol ile eğer bir hata oluştuğu anlaşılıyor ise transaction iptal edilirdi. Yani otomatik bir transaction iptal mekanizması yoktu.

SQL Server 2005 transaction iptali için kullanılabilecek olan yeni TRY..CATCH yapısını getirmektedir. TRY..CATCH yapısı .NET ile uygulama geliştiren programcılara tanıdık gelecek bir yapıdır. TRY..CATCH yapısı aynı .NET dillerinde olduğu gibi TRY bloğu içerisindeki kodlar çalışırken bir hata olduğunda CATCH bloğunu çalıştırır.

```
SET XACT_ABORT ON
```

```
BEGIN TRY
  BEGIN TRAN
  UPDATE MusteriHesap SET Bakiye=Bakiye-1000 WHERE MusteriID=15
  UPDATE MusteriHesap SET Bakiye=Bakiye+1000 WHERE MusteriID=21
  SELECT 'havale islemi yapildi'
  COMMIT
END TRY
BEGIN CATCH TRAN_ABORT
  ROLLBACK
  SELECT 'havale islemi yapilamadi'
END CATCH
```

Not: Bu örnek kod Beta1 ile çalışmaktadır.

## .NET ile SQL Server Programlama

SQL Server 2005'in getirdiği en önemli özellik olarak söyleyebileceğimiz özelliği .NET ile birlikte çalışabilmesidir. Bu durum özellikle şuan .NET ile uygulama geliştiren yazılım geliştiricileri heyecanlandırmaktadır.

SQL Server 2005, .NET ile yazılmış dll dosyalarını kendi çalıştırma ortamı içerisinde çalıştırabilmekte yani "host" edebilmektedir.

.NET ile SQL Server 2005 için "Managed" (yönetilen) diye tanımlanan nesnelere oluşturabiliriz. Bu nesnelere "Managed" denilmesinin sebebi .NET ile geliştirilen uygulamanın .NET çalışma ortamını sağlayan **Common Language Runtime (CLR)** tarafında yönetilmesidir. Aynı şekilde SQL Server üzerinde çalışan bir .NET Assembly'si de "Managed" yani yönetilen ortamda çalışır.

SQL Server 2005 .NET Assembly'lerini host edebilmesi sayesinde oluşturulabilen veritabanı nesnelere nelerdir?

- Stored Procedure
- Function
- Trigger
- Aggregate Function
- Kullanıcı Tanımlı Özel Veri Türleri

SQL Server 2000 ile ADO.NET kullanarak uygulama geliştirirken SqlClient Veri Sağlayıcısını (Data Provider) kullanıyorduk. Bu durum hala geçerli fakat SQL Server üzerinden çalışacak olan .NET kodlarında kullanılması amacıyla **System.Data.SqlServer** namespace hiyerarşisinde yer alan yeni bir veri sağlayıcısı .NET 2.0 ile beraber SQL Server 2005 için gelmektedir. Bu veri sağlayıcısının adı "**In-Proc Managed Provider**" olarak geçer. In-Proc "In Process" kelimelerinden oluşturulmuş bir kısaltmadır.

**In-Proc Veri Sağlayıcısında** yer alan sınıflardan ve kullanımlarından bahsetmeden önce .NET ile yazılmış metodların nasıl SQL Server 2005 ile çalışabilir hale geleceğini inceleyeceğiz. Ardından In-Proc veri sağlayıcısını örnekler ile inceleyeceğiz.

## Assembly Yönetimi

SQL Server 2005'in gerek duyduğu kurallar çerçevesinde tasarlanmış olan sınıflar ve metodları içeren bir assembly dosyası elde edildiğinde bunun SQL Server üzerinde çalıştırılabilmesi için SQL Server tarafından kataloglanması gerekir. Kısacası assembly dosyasının SQL Server'a tanıtılması gerekir.

Bir assembly'nin SQL Server'a tanıtılması için CREATE ASSEMBLY ifadesi kullanılır.

```
CREATE ASSEMBLY <assembly_tanımlayıcı_ismi>  
FROM <assembly_dosyasının_fiziksel_konumu>
```

Burada CREATE ASSEMBLY deyiminden sonra assembly'nin SQL Server üzerindeki mantıksal ismi tanımlanır. FROM anahtar kelimesinden sonra ise DLL dosyasının bulunduğu fiziksel konum belirtilir. Örneğin;

```
CREATE ASSEMBLY BilgeAdam  
FROM 'C:\cengiz\BilgeAdam.dll'
```

Yukarıdaki örnekte C:\cengiz\BilgeAdam.dll konumundaki assembly dosyası SQL Server'a BilgeAdam ismi ile kayıt edilmiştir.

Kaydedilen dll dosyaları veritabanında bir sistem tablosunda saklanmaktadır. Bir dll dosyasını CREATE ASSEMBLY ile SQL Server'a kaydettikten sonra SQL Server'ın sürücüdeki fiziksel dosya ile hiç bir bağlantısı kalmaz. Dosyanın kopyası veritabanında saklanır. Bu şekilde backup ve restore gibi işlemlerde assembly dosyaları da saklanmış olurlar.

Bu doküman [www.cengizhan.com](http://www.cengizhan.com)'dan indirilmiştir.

Cengiz HAN (cengiz@cengizhan.com)

Bu komut çalıştığında SQL Server sadece bu dll dosyasını değil bu assembly'nin ihtiyaç duyduğu (refere ettiği) diğer assembly dosyalarının da veritabanındaki sistem tablosunda saklar. Burada refere edilen diğer assembly dosyaları .NET Framework assembly dosyaları değil, kullanıcı tanımlı assembly dosyalarıdır. SQL Server .NET Framework'ün assembly dosyalarını kaydetmesine gerek olmadığı için o assembly dosyalarını kaydetmez. Ayrıca refere edilen diğer assembly dosyasının SQL Server tarafından kaydedilebilmesi için ana dosya ile **aynı klasörde olması gerekir**.

CREATE ASSEMBLY ifadesi ile bir assembly dosyası veritabanına kaydedilir. Kaydedilen assembly tüm sunucu bazında değil veritabanı bazında kaydedilir. Ve assembly için verilen mantıksal ismin veritabanı üzerinde tekil olması gerekir.

NOT: Veritabanında kayıtlı olan assemblyler hakkında bilgi almak için **sys.assemblies** view nesnesini sorgulayabilirsiniz. Ayrıca **sys.assembly\_files** view nesnesi ile her bir assembly için veritabanına eklenmiş olan dosya(lar) görüntülenebilir.

```
SELECT * FROM sys.assemblies
SELECT * FROM sys.assembly_files
```

## Code Access Security (CAS)

Bir assembly veritabanına yüklendiğinde assemblynin yapabileceği yada yapamayacağı işler belirlenmelidir. Bazı kaynaklara erişim veya işlemleri yapabilme izni belirlemek gerekir. CLR altında çalışan .net assemblyleri güvenlik ayarlamaları Code Access Security (CAS) ile yapılır.

SQL Server 2005 assemblyler için üç tane güvenlik modeline sahiptir. Bunlar SAFE, EXTERNAL ACCESS, UNSAFE.

CAS Modeli	Açıklama
<b>SAFE</b>	Veriye erişim yetkisi vardır. In-Proc veri sağlayıcısı sınıflarını kullanabilir. Dosya sistemi, ağ bağlantısı gibi dış kaynaklara erişim yetkisi yoktur. <b>Varsayılan güvenlik modelidir.</b>
<b>EXTERNAL_ACCESS</b>	SAFE ile yapılan bilen tüm işlemlerin yapılabileceği gibi dosya sistemi, ağ bağlantısı, event loglar gibi dış kaynaklara erişim de sağlanabilir.
<b>UNSAFE</b>	Bu güvenlik modeli ile çalışan kodlar üzerinde CASın hiç bir kısıtlaması yoktur.

```
CREATE ASSEMBLY BilgeAdam
FROM 'C:\cengiz\BilgeAdam.dll'
WITH PERMISSION_SET=SAFE
```

```
CREATE ASSEMBLY BilgeAdam
FROM 'C:\cengiz\BilgeAdam.dll'
WITH PERMISSION_SET=EXTERNAL_ACCESS
```

```
CREATE ASSEMBLY BilgeAdam
FROM 'C:\cengiz\BilgeAdam.dll'
WITH PERMISSION_SET=UNSAFE
```

CREATE ASSEMBLY ifadesi kullanılırken WITH PERMISSION\_SET eki kullanılmaz ise assembly kaydı **varsayılan** olarak belirlenmiş olan **SAFE** güvenlik modeli ile yapılır.

## Kaydedilen Bir Assembly'nin SQL Server Veritabanından Silinmesi

Kaydedilen bir assemblynin veritabanından silinmesi için DROP ASSEMBLY ifadesi kullanılır.

```
DROP ASSEMBLY <assembly_tanımlayıcı_ismi>
```

Örneğin BilgeAdam ismi ile kaydettiğimiz assembly'i silmek için aşağıdaki komut çalıştırılmalıdır.

```
DROP ASSEMBLY BilgeAdam
```

DROP ASSEMBLY komutu yukarıdaki şekilde kullanıldığında assembly'i silmesi için assembly'i kullanan bir veritabanı nesnesi (stored procedure,function gibi) olmamalıdır. Assembly silinmeden önce onunla ilgili veritabanı nesnelere önceden silinmelidir.

## In-Proc Managed Provider

Oluşturulan assembly dosyaları veritabanına nasıl kaydedilir, kaydedilen bir assembly veritabanında nasıl silinir gibi işlemleri bi önceki bölümde inceledik. Fakat bu DLL dosyaları nasıl oluşturulur. Oluşturulan sınıfların nasıl bir tasarıma sahip olması gerekir. Stored Procedure, Function gibi veritabanı nesnelere için nasıl bir sınıf tasarımı yapılır, bu sınıflardaki metodlar veritabanında nasıl veritabanı nesnesi haline getirilir gibi konuları **In-Proc Managed Provider** başlıklı bu bölümde inceleyeceğiz.

## SQL Server için Assembly Oluşturmak

SQL Server 2005 üzerinde bir veritabanına kaydedilecek ve "Yönetilen Veri Tabanı Nesnelere" olarak kullanılacak sınıfları ve metodları içeren assemblyler herhangi bir .NET dili ile yazılmış olabilirler. Bu diller .NET ile uygulama geliştirme alanında genellikle tercih edilen Visual Basic.NET veya C# dillerinden bir tanesi olabileceği gibi COBOL gibi .NET derleyicisi olan herhangi bir programlama dili olabilir. Bu dokümanda örnek programlama dili olarak Visual Basic.NET kullanılacaktır.

Veritabanı nesnesi (stored procedure, function, trigger gibi) olarak kullanılmak üzere metodlar tanımlanır. .NET ortamında seçilen dil ile oluşturulan metodlar veritabanında stored procedure, function gibi veritabanı nesnelere olarak tanımlanırlar.

Oluşturulacak olan sınıf tasarımında uyulması gereken kurallar :

- **Sınıf public** olmalıdır.
- Tanımlanan **metod public** olmalıdır.
- Tanımlanan **metod shared** olmalıdır. (C#'da static)
- **Yuvalanmış** (bir sınıfın içerisinde yer alan diğer sınıf) sınıf **olmamalıdır**.

## Stored Procedure Oluşturmak

İstenilen kurallara uygun bir sınıf tasarımı yapıp assembly dosyası derlenir. Elde edilen assembly bir .dll dosyası olmalıdır. .exe uzantılı dosyalar veritabanına kaydedilemez. Bir veritabanı nesnesi oluşturmadan önce daha önce bahsettiğimiz CREATE ASSEMBLY ifadesi ile assembly dosyası veritabanına kaydedilmelidir. Ardından CREATE PROCEDURE gibi veritabanı nesnesi oluşturmak için kullandığımız ifadeler kullanılabilir.

```
Public Class test4
    Public Shared Sub Method1()
        'kodlar burada
    End Sub
End Class
```

Yukarıdaki örnek sınıf tasarımı Stored Procedure (SP) olarak kullanılacak bir metod tanımlaması içeriyor. Buradaki Method1 isimindeki metodun bir SP olarak kaydedilmesinden önce bu sınıfın bir dll haline getirilmesi ve SQL Server veritabanına kaydedilmesi gerekir. Bunun için .NET Framework ile beraber gelen **VBC.EXE** derleyici kullanılabilir.

Not: SQL Server 2005 üzerine uygulama geliştirmek için Visual Studio 2005 aracını da kullanabilirsiniz. Bu araç ile CREATE ASSEMBLY, CREATE PROCEDURE gibi işlemler basit bir arabirim ile kolaylık ile yapılabilmektedir.

```
%WINDIR%\Microsoft.NET\Framework\v2.0.40607\vbc.exe test4.vb /t:library
```

Yukarıdaki kod satırında test4.vb isimindeki kod dosyası bir sınıf kütüphanesi (/t:library parametresi bunu belirler.) olarak derleniyor.



Vbc.exe derleyicisi ile .vb kod dosyası derlendikten sonra dll dosyası oluşturulur.

Dll dosyası elde edildikten sonra SP olarak tanımlanması yapmadan önce CREATE ASSEMBLY ile assemblynin veritabanına kaydedilmesi gerekir.

```
CREATE ASSEMBLY test4  
FROM 'C:\test\4\test4.dll'
```

Yukarıdaki T-SQL sorgusu ile test4.dll dosyası veritabanına kaydedilir.

Not: SQL Server'ın C:\test\4\test4.dll konumundaki dosya ile artık bir ilişkisi yoktur, assembly dosyası veritabanında sistem tablolarında saklıdır.

Uygun kurallar ile bir sınıfı tasarımı yaptık, bunu derleyip DLL dosyası haline getirdik, veritabanına kaydedilmesini sağladık. Şimdi veritabanında kataloglanmış olan bu assemblyyi kullanarak bir SP oluşturmalıyız.

```
CREATE PROCEDURE <SPninAdi>  
AS EXTERNAL NAME  
<AssemblyMantıksalAdi>.<AssemblydekiSınıfAdi>.<SınıfdakiMetod>
```

Durumumuz için çalıştırılması gereken T-SQL sorgusu

```
CREATE PROCEDURE Method1  
AS EXTERNAL NAME test4.test4.Method1
```

Yukarıdaki sorgu ile SPmiz oluşturulur ve SP çağrıldığı zaman test4 isimindeki assemblydeki Method1 çalıştırılacaktır.

EXTERNAL NAME ifadesinde sonraki bölümde verilen Method ismi büyük küçük harf duyarlıdır. Burada büyük küçük harf duyarlılığı uygulama geliştirilen dilin büyük küçük harf duyarlı olmasına bağlı değildir. Örneğin VB.NET ile yazılan programlarda büyük küçük harf duyarlılığı yoktur. Fakat SQL Server'da bir assembly kullanılması durumu söz konusu olduğunda metod isimleri büyük küçük harf duyarlılığı göstermektedir.

Oluşturulan Stored Procedure nesnesini kullanmak için aşağıdaki T-SQL sorgusunu yazabilirsiniz.

```
EXEC Method1
```

## Parametre Kullanımı

Stored Procedure'ler genellikle parametre ile kullanılırlar. Parametrelili bir Stored Procedure veri tabanı nesnesi oluşturmak için parametre alan bir metod tanımlanması gerekmektedir. Ardından tanımlanan bu metodun bulunduğu assembly sisteme kaydedilip, CREATE PROCEDURE ifadesi ile SP tanımlanması yapılmalıdır. Bir önceki önceki örnekten farklı olarak .NET metodunun ve SP'nin parametrik olarak tanımlanmış olması gerekmektedir.

.NET ile parametre tanımlaması yaparken kullandığımız ByRef ve ByVal ifadelerinin yönetilen bir Stored Procedure nesnesinde kullanımında desteklenmektedir.

Aşağıdaki örnek .NET kodunu inceleyiniz.

```
Imports System.Data

Public Class test5

    Public Shared Sub SP1(ByVal sayi As SqlTypes.SqlInt32)
        sayi = 34
    End Sub

    Public Shared Sub SP2(ByRef sayi As SqlTypes.SqlInt32)
        sayi = 34
    End Sub

End Class
```

Yukarıdaki metodlarda dikkat edilmesi gereken bir nokta SP1'de parametrenin **ByVal** olarak, SP2'de ise **ByRef** olarak aktarılmasıdır.

Yukarıdaki kodu test5.vb ismi ile kaydettikten sonra aşağıdaki şekilde test5.dll dosyası derlenmelidir. Burada /r ile System.Data.dll' e referans verilmelidir. Çünkü System.Data.SqlTypes namespace'i içerisindeki SqlInt32 sınıfı uygulamamızda kullanılmıştır.

```
%WINDIR%\Microsoft.NET\Framework\v2.0.40607\vbc.exe test5.vb
/r:System.Data.dll /t:library
```

C:\test\5\test5.dll konumunda derlediğim assembly dosyasını Sql Server'a kaydetmek için aşağıdaki T-SQL kodunu çalıştırıyorum. Atık veritabanında **test5** adında bir assembly kayıtlı durumda.

```
CREATE ASSEMBLY test5
FROM 'c:\test\5\test5.dll'
```

Tanımladığımız test5 assemblysi içerisinde yer alan test5 ismindeki sınıfın içerisinde yer alan iki metodu (SP1 ve SP2) sistemde yönetilen birer Stored Procedure olarak tanımlamak için gerekli olan CREATE PROCEDURE deyimlerini aşağıdaki gibi yazıyorum.

```
CREATE PROCEDURE test5sp1
@para1 int
AS
EXTERNAL NAME test5.test5.SP1
```

```
CREATE PROCEDURE test5sp2
@para1 int OUTPUT
AS
EXTERNAL NAME test5.test5.SP2
```

Yukarıdaki iki CREATE PROCEDURE deyimlerinde dikkat edilmesi gereken bir nokta .net metodundaki parametre ismi ile t-sql ile tanımlanan parametre adının aynı olmak zorunda olmaması. Bir .net metodu ile CREATE PROCEDURE deyiminde tanımlanan **parametreler sırasına göre eşleştirilir**. İkinci tanımlamada parametrenin sonunda OUTPUT parametresi vardır bir T-SQL kuralı olan bu **OUTPUT** parametresi ile bu parametreden değer döneceği belirlenir. ByRef olan Bu doküman [www.cengizhan.com](http://www.cengizhan.com)'dan indirilmiştir.  
Gengiz HAN (cengiz@cengizhan.com)

parametrelerde OUTPUT ifadesini parametre tanımlama yapısında kullanamak gerekmektedir. Aynı şekilde ByVal olan parametre tanımlama yapısında OUTPUT kullanmaya çalışmak CREATE PROCEDURE çalışması aşamasında hataya sebep olacak ve procedure oluşturulamayacaktır. OUTPUT ifadesi sadece ByRef tanımlamalarda kullanabilmektedir.

Not: C#'da ByRef ifadesine alternatif olarak ref ve out ifadeleri yer almaktadır. Kullanım yöntemleri ve etkileri VB.NET'deki ByRef gibidir.

Bu kodları çalıştırdıktan ve yönetilen stored procedure nesnelarini oluşturduktan sonra bunların kullanan aşağıdaki kodları yazarak ByVal ve ByRef ifadelerinin yönetilen veritabanı nesnelarine nasıl yansıdığını görebiliriz.

```
DECLARE @x int
SET @x=10
EXEC test5sp1 @x
SELECT @x AS 'Sonuç'
```

Sonuç
10

```
DECLARE @x int
SET @x=10
EXEC test5sp2 @x OUTPUT
SELECT @x AS 'Sonuç'
```

Sonuç
34

Görüldüğü gibi ByRef olan **test5sp2** Stored Procedure nesnesinin .net metodu içerisinde parametre değişkeninde üzerinde yapılan değişiklik (yeni değer ataması) parametre olarak gelen @x değişkenine yansımıştır. Ve **SELECT @x AS 'Sonuç'** olarak yapılan sorgu ile @x değişkeninin değeri .net metodu içerisinde atanan değer olan **34** olarak gözükmektedir.

## Geriye Değer Döndürmek

Stored Procedure nesnelari ile **User Defined Function** nesnelarinde olduğu geriye bir değer döndürülebilir. User Defined Function nesnesinden farklı olarak nesne tanımlaması (CREATE .. ) yapılırken Stored Procedure nesnesinde geri dönüş değeri için veri türü tanımlaması yapılmaz.

```
Imports System.Data
Imports System.Data.SqlTypes

Public Class test6

    Public Shared Function ArtiBir(ByVal sayi As SqlInt32) As SqlInt32
        Return sayi + 1
    End Function

End Class
```

Yukarıdaki sınıf tasarımı ile **ArtiBir** ismiden bir function tanımlanıyor. Bu Function aldığı parametrede ki tam sayı değeri bir arttırıp geriye dönüyor.

```
%WINDIR%\Microsoft.NET\Framework\v2.0.40607\vbc.exe test6.vb
/r:System.Data.dll /t:library
```

Bu kaynak kodu derleyip assembly dosyasına çevirmek için yukarıdaki kodu kullanıyoruz. Bu komut satırı derleyicisi çalıştıktan sonra test6.dll oluşturuluyor.

Bu doküman [www.cengizhan.com](http://www.cengizhan.com)'dan indirilmiştir.  
Cengiz HAN (cengiz@cengizhan.com)

```
CREATE ASSEMBLY test6
FROM 'C:\test\6\test6.dll'
```

Oluşan assembly dosyasını veritabanına kaydetmek için yukarıdaki CREATE ASSEMBLY T-SQL ifadesini kullanıyoruz.

```
CREATE PROCEDURE ArtiBir
@sayi int
AS EXTERNAL NAME test6.test6.ArtiBir
```

test6 isminde assembly veritabanında oluşturulduktan sonra bu assemblydeki test6 sınıfında bulunan ArtiBir isimindeki metodu bir yönetilen Stored Procedure olarak tanımlamak için yukarıdaki kodu çalıştırıyoruz. Burada dikkat edilmesi gereken nokta parametrelerin .net metodu ile aynı sırada olması ve veri türü uygunluğudur. Burada tek parametre olduğu için sıra sorunu yoktur fakat veri türünün uygun olması dikkat edilmesi gereken bir noktadır.

```
DECLARE @x int
EXEC @x=ArtiBir 34
SELECT @x AS 'Sonuç'
```

Yönetilen Stored Procedure oluşturulduktan sonra test etmek için yukarıdaki kodu yazıyoruz. Burada @x isminde int türünde bir değişken tanımlanıyor, EXEC @x=ArtiBir 34 satırı ile @x değişkenine ArtiBir 34 SP çağrısının sonucu (35) atanıyor. SELECT ile görüntülenen @x değerinin sonucu aşağıda 35 olarak görülmektedir.

Sonuç
35

## User Defined Function Oluşturmak

SQL Server için .NET ile Function oluşturmak Stored Procedure oluşturmaktan pek farklı değildir.

```
CREATE FUNCTION <FunctionAdı>
(
<Parametreler>
)
RETURNS <ReturnTürü>
AS EXTERNAL NAME
<AssemblyMantıksalAdı>.<AssemblydekiSınıfAdı>.<SınıfdakiMetod>
```

Yukarıdaki şemada gözüktüğü gibi bir T-SQL ifadesi ile .NET assemblysi içinde yer alan bir metodu SQL Server 2005 Function nesnesi olarak tanımlayabilirsiniz. <FunctionAdı> bölümünde fonksiyona verilecek olan isim yazılmalıdır. <Parametreler> bölümünde ise Function'a verilecek olan parametreler yazılmalıdır.

Örneğin aldığı iki tam sayı parametreyi toplayıp geriye sonucunu dönecek olan bir function yazmak için aşağıdaki adımlar izlenmelidir.

```
Imports System.Data
Imports System.Data.SqlTypes

Public Class test7

    Public Shared Function Topla(ByVal sayi1 As SqlInt32, _
                                ByVal sayi2 As SqlInt32) As SqlInt32
        Return sayi1 + sayi2
    End Function

End Class
```

Bu doküman [www.cengizhan.com](http://www.cengizhan.com)'dan indirilmiştir.  
Gengiz HAN (cengiz@cengizhan.com)

Yukarıdaki kodu test7.vb olarak kaydettikten sonra aşağıdaki şekilde derlenir.

```
%WINDIR%\Microsoft.NET\Framework\v2.0.40607\vbc.exe test7.vb  
/r:System.Data.dll /t:library
```

Derlenen assembly dosyası test7 adıyla veritabanına kaydedilir.

```
CREATE ASSEMBLY test7  
FROM 'C:\test\7\test7.dll'
```

test7 assemblysinde yer alan test7 sınıfının içindeki Topla metodu aşağıdaki CREATE FUNCTION T-SQL ifadesi ile yönetilen bir function olarak tanımlanır.

```
CREATE FUNCTION Topla  
(  
@sayi1 int,  
@sayi2 int  
)  
RETURNS int  
AS EXTERNAL NAME test7.test7.Topla
```

Tanımlandığımız function nesnesini kullanmak için aşağıdaki SELECT ifadesini çalıştırıyoruz. Sonuç resimde gözükmektedir.

```
SELECT dbo.Topla(8,9) AS 'Sonuç'
```

Sonuç
17

## In-Proc Managed Provider Nesneleri

SQL Server 2000 ile ADO.NET kullanarak uygulama geliştirirken SqlClient Veri Sağlayıcısını (Data Provider) kullanıyorduk. Bu durum hala geçerli fakat System.Data.SqlClient namespace hiyerarşisinde yer alan bu sağlayıcı SQL Server'a dışarıdan erişim için tasarlanmıştır. Yani yazdığımız bir Windows uygulaması, ASP.NET uygulaması içerisinde SQL Server'a bağlanmak için kullanılır.

SQL Server üzerinden çalışacak olan .NET kodlarında kullanılması amacıyla **System.Data.SqlServer** namespace hiyerarşisinde yer alan yeni bir veri sağlayıcısı .NET 2.0 ile beraber SQL Server 2005 için gelmektedir. Bu veri sağlayıcısının adı "**In-Proc Managed Provider**" olarak geçer. In-Proc "In Process" kelimelerinden oluşturulmuş bir kısaltmadır.

Not : In-Proc Managed Provider sınıflarının kullanıldığı bir uygulamayı derlemek için sqlaccess.dll dosyasına referans verilmesi gereklidir.

```
%WINDIR%\Microsoft.NET\Framework\v2.0.40607\vbc.exe dosya.vb
/r:System.Data.dll /r:"C:\Program Files\Microsoft Visual Studio
8\Common7\IDE\PublicAssemblies\sqlaccess.dll" /t:library
```

Not: Bu bölümdeki örneklerde kodların derlenmesi ve sql server üzerine kaydedilmesi adımları anlatılmayacak sadece .net kodları verilecektir.

## SqlConnection Sınıfı

Aşağıdaki Sql Server Managed Provider (SqlConnection) kullanılarak veri erişimi örnek kodunu inceleyiniz.

```
Dim baglanti As New System.Data.SqlClient.SqlConnection
baglanti.ConnectionString =
"Server=(local)\YM;Database=Northwind;Trusted_Connection=True;"
Dim komut As New System.Data.SqlClient.SqlCommand
komut.Connection = baglanti
komut.CommandText = "SELECT ProductName FROM Products"
Dim okuyucu As System.Data.SqlClient.SqlDataReader
baglanti.Open()
okuyucu = komut.ExecuteReader

While okuyucu.Read
    'okuyucu(0) ile veriyi kullan
End While

okuyucu.Close()
baglanti.Close()
```

Yukarıdaki örnek Sql Server'a dışarıdan erişmek için kullanılacak olan kodu yapısını örneklemektedir. In-Proc Managed Provider'da hemen hemen her şey aynı buradaki yapı ile aynıdır. Ancak bazı önemli noktalarda farklar ortaya çıkmaktadır. In-Proc Provider ile yapılacak işlemler için bir SqlConnection bağlantı nesnesi oluşturmaya gerek yoktur.

Yazacağımız kodlar zaten Sql Server üzerinde çalışacaktır ve bağlantı nesnesi kendi içinde zaten açık durumdadır. Yazdığımız kod Sql Server üzerinde çalışırken bir bağlantı zaten vardır. Biz yazacağımız T-SQL kodlarını bu açık olan bağlantı üzerinden çalışırız. In-Proc Managed Provider aktif durumdaki bağlantı nesnesine ve diğer nesnelere erişebilmemiz için yeni bir sınıf içermektedir :

## System.Data.SqlServer.SqlContext.

Sql Server üzerinde çalışacak olan ve Products tablosundaki tüm kayıtları dönecek olan yönetilen Stored Procedure şu şekile yazılmalıdır.

```
Dim komut As System.Data.SqlServer.SqlCommand =
System.Data.SqlServer.SqlContext.GetCommand
```

Bu doküman [www.cengizhan.com](http://www.cengizhan.com)'dan indirilmiştir.  
Cengiz HAN (cengiz@cengizhan.com)

```

komut.CommandText = "SELECT ProductName FROM Products"

Dim okuyucu As System.Data.SqlServer.SqlDataReader

okuyucu = komut.ExecuteReader
While okuyucu.Read
    'okuyucu(0) ile veriyi kullan
End While
okuyucu.Close()

```

Yukarıdaki kodu incelersek, Daha önceki dışarıdan erişim için yazdığım koddan farklı olarak burada kullanılan nesnelere `System.Data.SqlClient` namespace'i içerisinde değil `System.Data.SqlServer` namespace'i içerisindeyiz. Çünkü yazdığımız bu kodların Sql Server üzerinde çalışmasını istiyoruz.

İkinci bir nokta ise kod yazarak, connectionstring belirleyerek bir bağlantı açılmamış olmasıdır. **SqlConnection** sınıfını kullanarak aktif olan **System.Data.SqlServer.SqlCommand** nesnesi alınıyor. Bu nesne için bağlantı nesnesi belirlemeye gerek yoktur zaten aktif bağlantı içerisindeki nesne alınmıştır.

Bu örnekte sınıflar namespace hiyerarşileri yazılarak kullanılmıştır, `System.Data.SqlServer` namespace ekini her sınıfa eklemek yerine `Imports System.Data.SqlServer` yazdıktan sonra aynı kodu şu şekilde yazabilirsiniz.

```

Imports System.Data.SqlServer
...
...
...
Dim komut As SqlCommand = SqlConnection.CreateCommand
komut.CommandText = "SELECT ProductName FROM Products"
Dim okuyucu As SqlDataReader
okuyucu = komut.ExecuteReader
While okuyucu.Read
    'okuyucu(0) ile veriyi kullan
End While
okuyucu.Close()

```

## SqlPipe

Yukarıdaki örnekte veriler `SqlDataReader` nesnesi ile okunabilir durumdalar, while döngüsü ile her bir kayıt okunuyor. Fakat sorgu sonucu elde edilen verilerin istemciye aktarılmasını gerektiği durumlarda **SqlPipe** isminde yeni bir nesne kullanılır.

Az önceki örneğimizi verileri istemciye aktaracak şekilde düzenlersek.

```

Imports System.Data.SqlServer
...
...
...
Dim komut As SqlCommand = SqlConnection.CreateCommand
komut.CommandText = "SELECT ProductName FROM Products"
Dim okuyucu As SqlDataReader
okuyucu = komut.ExecuteReader
SqlConnection.GetPipe.Send(okuyucu)

```

Yukarıdaki koddaki koyu bölüm `SqlPipe` türünde bir değişken tanımlayarak aşağıda şekilde de yazılabilir.

```

Dim myPipe As SqlPipe = SqlConnection.GetPipe
myPipe.Send(okuyucu)

```

SqlPipe nesnesinin istemciye veri aktarımı sırasında en sık kullanılan metodları ve özellikleri şunlardır.

**Execute** : İki overload metodu vardır. Verilen SqlCommand veya SqlExecutionContext türünde parametreler alır, bunları çalıştırıp sonucu istemciye aktarır.

**Send** : Dört overload metodu vardır. İki istemciye veri döndürür. Bunlar ISqlReader ve ISqlRecord türünde parametrelerdir. Bir overload metodu istemciye hata mesajı, bir overload metodu ise mesaj döndürür.

**SendResultsRow** : İstemciye ISqlRecord türünde bir kayıt döndürür.

**SendResultStart** : Sonuçların istemciye gönderilmekte olduğunu bildirmek amacıyla SendingResults boolean özelliğini **True** yapar.

**SendResultsEnd** : Sonuçların istemciye gönderilmekte olduğunu bildirmek amacıyla SendingResults boolean özelliğini **False** yapar.

**SendingResults** : Veri aktarımının devam edip etmediği ile ilgili bir veren boolean bir özelliktir.

### SqlPipe ile İstemciye mesaj gönderilmesi

```
Dim myPipe As SqlPipe
myPipe = SqlContext.GetPipe
myPipe.Send("İstemciye giden mesaj")
```

### SqlPipe ile SqlCommand nesnesinin istemciye aktarılması

```
Dim komut As SqlCommand = SqlContext.GetCommand
komut.CommandText = "SELECT ProductName FROM Products"
Dim myPipe As SqlPipe
myPipe = SqlContext.GetPipe
myPipe.Execute(komut)
```

### Parametrelili bir sorgunun istemciye aktarımı

```
Public Shared Sub Urunler(ByVal productID As SqlInt32)
    Dim komut As SqlCommand = SqlContext.GetCommand
    komut.CommandText = "SELECT ProductName FROM Products WHERE
ProductID=@PID"
    komut.Parameters.Add("@PID", SqlDbType.Int).Value = productID
    Dim myPipe As SqlPipe
    myPipe = SqlContext.GetPipe
    myPipe.Execute(komut)
End Sub
```

## SqlDefinition ve SqlExecutionContext

In-Proc veri sağlayıcısı bir sorgunun sabit öğeleri ve değişken öğeleri arasında yeni bir iş bölümü getirir. Bir sorgunun sabit öğeleri sorgu cümlesi, aldığı parametreler ve veri türleri gibi her çalışmasında aynı kalan, değişmeyen öğelerdir. Değişken öğeler ise parametrelerin değerleri ve aktif transaction ile ilgili bilgilerdir.

Sabit öğeler **SqlDefinition**, **SqlExecutionContext** sınıfları ile temsil edilirler.

SqlDefinition sınıfından bir nesne elde etmek için sınıf yapılandırıcısının iki overloadundan birisini kullanabiliriz. Bir tanesi SqlCommand nesnesi isteyen bir overloadur. Diğer ise bir SqlCommand için gerekli olan sql sorgusu, sorgunun türü, ve ayrıntılı parametre bilgilerini istemektedir.

Üçüncü parametre olarak ayrıntılı parametre bilgisi için **SqlMetaData** türünde bir nesne dizisi istemektedir. SqlMetaData sınıfı SqlCommand için tanımlanacak olan parametreler ile ilgili veri türü, adı gibi bir çok bilgiyi yapılandırıcısında istemektedir. Bu sınıftan türemiş nesnelere değerleri readonly propertyler ile sunulur, yani ancak yapılandırma aşamasında verilen değerleri kullanırız, daha sonra bu özelliklerden birisinin değerini değiştiremeyiz.

SqlExecutionContext sınıfından bir nesne doğrudan "new" ile türetilemez. Bir SqlExecutionContext elde etmek için SqlConnection nesnesinin CreateExecutionContext metodu ile bir SqlExecutionContext elde edilebilir. CreateExecutionContext metodu parametre olarak bir SqlDefinition metodu bekler.

Örneğin :

```
Sub Urunler(ByVal productID As SqlInt32)
    Dim komut As New SqlCommand
    komut.CommandText = "SELECT ProductName FROM Products WHERE
ProductID=@PID"
    komut.Parameters.Add("@PID", SqlDbType.Int)
    'sqlcommand oluřturuldu
    Dim def1 As New SqlDefinition(komut)
    'sqldefinition oluřturuldu
    Dim baglanti As SqlConnection
    baglanti = SqlContext.GetConnection
    Dim exc1 As SqlExecutionContext = baglanti.CreateExecutionContext(def1)
    exc1.SetSqlInt32(0, productID) '0.parametreye deęer atandı
    'executioncontext oluřturuldu
    SqlContext.GetPipe.Execute(exc1)
    'exc istemciye gnderildi
End Sub
```