

DİNAMİK SQL NEDİR?

13

Veritabanı programlamada gerçekleştirilecek işlemler genel olarak daha önceden belirlenen sorunlar için oluşturulan SQL cümleleriyle gerçekleştirilir. Ancak, bazı durumlarda çözülmesi gereken işlemlerin çalışma anında belirlenmesi gerekebilir. Çalışma anında belirlenecek sorgular için Dinamik SQL kullanılır. Birçok büyük VTYS'de olduğu gibi Dinamik SQL özelliği SQL Server'da da desteklenmektedir.

DİNAMİK SQL YAZMAK

SQL Server, Dinamik SQL sorguları yazabilmek için iki farklı yöntem destekler. Bu tekniklerin kendi arasında farklılıkları vardır. Performans ve kullanım olarak farklı olsa da iki teknik de Dinamik SQL sorguları yazmak için kullanılabilir.

SQL Server, dinamik sorgu işlemini gerçekleştirmek için, aldığı metinsel değeri, bir SQL sorgusu ile birleştirerek dinamik olarak yeni bir sorgu üretecek **EXEC**, **EXECUTE** fonksiyonunu ya da **SP_ExecuteSQL** Stored Procedure'ünü kullanır.

EXEC[UTE]

EXEC fonksiyonu, Stored Procedure gibi nesnelerin çalıştırılması için de kullanılan bir fonksiyondur.

Söz Dizimi:

```
EXEC[UTE] ( { @string_degisken | [N] ' t_sql_ifadesi ' } [ + ...n] )
```

EXEC fonksiyonu, kendi içerisinde farklı bir çalışma ortamına sahiptir. Örneğin; **EXEC** fonksiyonuna `string` olarak verilen bir SQL sorgusunun içerisinde tanımlanmış bir değişkene dışarıdan erişilemez. Aynı şekilde, fonksiyon içerisinden de dışarıdaki bir değişken çağrılmaz. Ancak değeri daha önceden alınabilir.

Dinamik SQL doğru kullanıldığında etkili bir çözüm sunabilir. **EXEC** fonksiyonu, birçok açıdan yararlı olsa da, işlemleri dinamik hale getirmek adına kullanılmaması gerekir. Çünkü SQL Server, **EXEC** fonksiyonu içerisinde çalıştırılan sorgular için çalıştırma planı tutmaz. Bu da sorgular açısından performans kaybı demektir.

EXEC fonksiyonu, sadece yüksek gereklilik duyulan dinamik sorgular için kullanılmalıdır.

EXEC fonksiyonu için örnek uygulamalar oluşturarak, fonksiyonun hangi amaçlar ile kullanılabileceğini inceleyelim.

Herhangi bir basit veri seçme sorgusu dahi **EXEC** ile kullanılabilir.

```
EXEC ('SELECT * FROM Production.Product');
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00
6	317	LL Crankarm	CA-5965	0	0	Black	500	375	0,00	0,00
7	318	ML Crankarm	CA-6738	0	0	Black	500	375	0,00	0,00

`string` ile bir değişkenin **EXEC** fonksiyonuna atanması ile kullanılabilir.

```
DECLARE @TabloAd SYSNAME ='Sales.SalesOrderHeader';
EXECUTE ('SELECT * FROM ' + @TabloAd);
```

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber
1	43659	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43659
2	43660	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43660
3	43661	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43661
4	43662	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43662
5	43663	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43663
6	43664	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43664
7	43665	3	2005-07-01 00:00:00.000	2005-07-13 00:00:00.000	2005-07-08 00:00:00.000	5	0	SO43665

EXEC öncesinde bir değişkene atanan SQL sorgusunun fonksiyona atanması ile de kullanılabilir.

```
DECLARE @SQL VARCHAR(256);
SET @SQL = 'SELECT * FROM Production.Product';
EXEC (@SQL);
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00
6	317	LL Crankarm	CA-5965	0	0	Black	500	375	0,00	0,00
7	318	ML Crankarm	CA-6738	0	0	Black	500	375	0,00	0,00

EXEC fonksiyonu içerisinde oluşturulacak bir sorguda kullanılan sütun için takma ismi dışarıdan alarak dinamik bir sorgu hazırlayalım.

```
DECLARE @TakmaAd VARCHAR(6) = 'ÜrünAd';
EXEC ('SELECT Name AS ' + @TakmaAd + ' FROM Production.Product');
```

	ÜrünAd
1	Adjustable Race
2	All-Purpose Bike Stand
3	AWC Logo Cap
4	BB Ball Bearing
5	Bearing Ball
6	Bike Wash - Dissolver
7	Blade

Şema adı, tablo adı, sütun adı, karşılaştırma operatörü ve karşılaştırılacak değeri, çalışma zamanında alalım ve sonucu listeleyelim.

```
DECLARE @table VARCHAR(128);
DECLARE @schema VARCHAR(128);
DECLARE @column VARCHAR(128);
DECLARE @exp VARCHAR(4);
DECLARE @value VARCHAR(128);

SET @schema = 'Production'
SET @table = 'Product'
SET @column = 'ProductID'
SET @exp = '='
SET @value = '1'
```

```
EXEC('SELECT * FROM ' + @schema + '.' + @table + ' WHERE '
      + @column
      + @exp
      + @value);
```

ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00

Bu sorgu ile istediğimiz tam olarak şu idi:

```
SELECT * FROM Production.Product WHERE ProductID = 1
```

ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00

Tabi ki isterseniz, sorguda şema, tablo, sütun, operatör ve değer olmak üzere 5 parametreyi de değiştirerek farklı tablolardan farklı koşullarda istekler oluşturabilirsiniz.

Aynı sorguyu aşağıdaki **SET** işlemlerini gerçekleştirerek çalıştırmayı deneyin.

```
SET @schema = 'Person'
SET @table = 'Person'
SET @column = 'BusinessEntityID'
SET @exp = '<='
SET @value = '7'
```

Dinamik SQL ile bir tablo üzerinde dinamik sorgu oluşturalım.

İlk olarak tablomuzu oluşturalım.

```
CREATE TABLE DynamicSQL
(
    TableID INT IDENTITY NOT NULL CONSTRAINT PKDynSQL PRIMARY KEY,
    SchemaName VARCHAR(150),
    TableName VARCHAR(150),
    Create_Date SMALLDATETIME
);
```

SELECT ile oluşturduğumuz tabloya kayıt ekleyelim.

```
INSERT INTO DynamicSQL
SELECT S.Name AS SchemaName, T.Name AS TableName, T.Create_date AS OTarih
FROM Sys.Schemas S
JOIN Sys.Tables T
ON S.Schema_ID = T.Schema_ID;
```

Kayıtları eklediğimiz tabloyu listeleyelim.

```
SELECT * FROM DynamicSQL;
```

	TableID	SchemaName	TableName	Create_Date
1	1	Production	ScrapReason	2012-03-14 13:14:00
2	2	HumanResources	Shift	2012-03-14 13:14:00
3	3	Production	ProductCategory	2012-03-14 13:14:00
4	4	Purchasing	ShipMethod	2012-03-14 13:14:00
5	5	Production	ProductCostHistory	2012-03-14 13:14:00
6	6	Production	ProductDescription	2012-03-14 13:14:00
7	7	Sales	ShoppingCartItem	2012-03-14 13:14:00

Artık dinamik SQL sorgumuzu hazırlayabiliriz.

```
DECLARE @SchemaName VARCHAR(128);
DECLARE @TableName VARCHAR(128);

SELECT @SchemaName = SchemaName, @TableName = TableName
FROM DynamicSQL WHERE TableID = 7;

EXEC ('SELECT * FROM ' + @SchemaName + '.' + @TableName);
```

	ShoppingCartItemID	ShoppingCartID	Quantity	ProductID	DateCreated	ModifiedDate
1	2	14951	3	862	2007-12-11 17:54:07.603	2007-12-11 17:54:07.603
2	4	20621	4	881	2007-12-11 17:54:07.603	2007-12-11 17:54:07.603
3	5	20621	7	874	2007-12-11 17:54:07.603	2007-12-11 17:54:07.603

Bu dinamik sorgu ile **DynamicSQL** tablosundan, istediğiniz tablonun **TableID** değerini sorgulayabilirsiniz.

EXEC İÇERİSİNDE FONKSİYONLAR KULLANILABİLİR Mİ?

EXEC mimarisi gereği, içerisindeki verinin daha önceden çözümlenmesi gerekir. Yani direkt olarak **EXEC** içerisinde bir fonksiyon kullanılamaz.

EXEC içerisinde bir fonksiyon kullanılması gerekiyor ise; **EXEC** işleminden önce, fonksiyon ile ilgili işlemler yapılır ve bu değerler bir değişkene atanır. Daha sonra içerisindeki veriyi işlemek için, **EXEC** komutuna verilen değişken, herhangi bir hataya sebebiyet vermeyecektir. Ancak **EXEC** içerisinde bir fonksiyon kullanılırsa, bu durum hataya yol açar.

EXEC İLE STORED PROCEDURE KULLANIMI

Stored Procedure içerisinde dinamik sorgular oluşturulabilir. Bu yöntem performans olarak önerilmese de teknik olarak kullanılabilir.

Dışarıdan parametre olarak Stored Procedure ismini alarak prosedür çağıran bir Sproc oluşturalım.

```
CREATE PROC pr_ProcedureCall(
    @sp_ad VARCHAR(2000)
)
AS
EXEC (@sp_ad);
```

Kilitler ile ilgili bir sistem prosedürü olan sp_lock prosedürünü çağıralım.

```
pr_ProcedureCall 'sp_lock';
```

ya da

```
EXEC pr_ProcedureCall 'sp_lock';
```

	spid	dbid	Objid	Indid	Type	Resource	Mode	Status
1	51	5	0	0	DB		S	GRANT
2	52	7	0	0	MD	1(22:3:0)	Sch-S	GRANT
3	52	7	0	0	MD	1(22:2:0)	Sch-S	GRANT
4	52	7	0	0	MD	1(22:1:0)	Sch-S	GRANT
5	52	7	0	0	MD	1(22:4:0)	Sch-S	GRANT
6	52	32767	0	0	MD	1(f72ef125:1:0)	Sch-S	GRANT
7	52	32767	0	0	MD	1(29:13:0)	Sch-S	GRANT

DİNAMİK SQL GÜVENLİK SORUNSAĞI

Bir dinamik sorgu oluştururken, diğer tüm sorgulara göre daha dikkatli davranılmalıdır. Gerekli güvenlik önlemleri alınmazsa, dinamik sorgular bir para kasasına açılan küçük bir deliğe benzer. Hırsızın birisi, para kasasındaki bu deliği bir gün fark ederek tüm varlığını ele geçirebilir. Dinamik SQL ile oluşturulan sorgu da, eğer sorgu içerisinde bir filtreleme ve gerekli güvenlik koşulları oluşturulmazsa, kötü niyetli kişiler tarafından hiç tahmin etmeyeceğiniz ve istemeyeceğiniz sorgular çalıştırılarak, veri kaybına kadar gidebilecek sorunlar yaşatabilir.

Az önce geliştirdiğimiz ve basit olarak, sadece bir prosedür ismi vererek bir prosedürün çağrılmasını sağladığımız sorguyu güvenlik konusu ile tekrar ele alalım.

```
pr_ProcedureCall 'sp_lock';
```

Yukarıdaki sorgu sadece **sp_lock** prosedürünü çağıracaktır. Peki, kötü niyetli birinin farklı bir sorgu çalıştırması engelleyen bir önlem alındı mı?

pr_ProcedureCall prosedürünü **AdventureWorks** veritabanında oluşturmuştuk. Prosedürde herhangi bir değişiklik yapmadan aşağıdaki şekilde kullanalım.

```
pr_ProcedureCall 'USE DIJIBIL; DROP TABLE Makaleler';
```

AdventureWorks veritabanını kullanması gereken bir prosedüre **USE DIJIBIL** diyerek **DIJIBIL** veritabanını kullanma isteği gönderebildik. Ayrıyeten, noktalı virgülden sonra ikinci sorguyu da çalıştırarak **DIJIBIL** veritabanındaki **Makaleler** tablosunu silme isteğini de rahatlıkla gönderebildik.

Bu sorgu sonucunda, basit bir prosedürdeki güvenlik hatası ile farklı bir veritabanındaki bir tabloyu silebildik.

Bu tür dışarıdan parametre olarak çalışan tüm sorgularda, sadece sizin belirlediğiniz kriterlerde işlem yapabilecek yetki ve esneklik dışında hiç bir işleme müsaade edilmemelidir.

Ürün isimleri arasında arama yapacak bir Stored Procedure geliştirelim. Bu prosedür içerisindeki sorguları **EXEC** fonksiyonu ile çalıştıralım.

```
CREATE PROCEDURE sp_ProductDynamicSP(
    @val VARCHAR(10)
)
AS
EXEC('SELECT Name, ProductNumber
    FROM Production.Product
    WHERE Name LIKE \'%\' + @val + \'%\';
```

Prosedürü çağıralım.

```
EXEC sp_ProductDynamicSP 'jus'
```

	Name	ProductNumber
1	Adjustable Race	AR-5381

```
EXEC sp_ProductDynamicSP 'a'
```

	Name	ProductNumber
1	Adjustable Race	AR-5381
2	Bearing Ball	BA-8327
3	BB Ball Bearing	BE-2349
4	Headset Ball Bearings	BE-2908
5	Blade	BL-2036
6	LL Crankarm	CA-5965
7	ML Crankarm	CA-6738

Dişarıdan bir ürün **ProductID** değerini parametre olarak sonucunda ürünün **Name** ve **ProductNumber** bilgilerini getiren prosedür oluşturalım.

```
CREATE PROCEDURE sp_GetProductByIddynSP(
    @productID VARCHAR(10)
)
AS
EXEC('SELECT Name, ProductNumber
    FROM Production.Product
    WHERE ProductID = \' + @productID + \');
```

Prosedüre parametre değeri vererek çağırılım.

```
EXEC sp_GetProductByIDDynSP 1;
```

	Name	ProductNumber
1	Adjustable Race	AR-5381

EXEC FONKSİYONU İÇERİSİNDE TÜR DÖNÜŞÜMÜ

T-SQL sorgulamalarında veri tipi dönüştürme işlemleri gerçekleştirmek mümkündür. Bu işlem için sınırlı sayıda kısıtlama olsa da filtrelemelerde veri tipi dönüşümü desteklenir. Ancak dinamik sorgu üretirken, **EXEC** fonksiyonunda, fonksiyon içerisinde tür dönüşümü yapılamaz.

Yukarıdaki prosedürü dışarıdan **INT** veri tipinde değer alacak şekilde düzenleyelim.

```
CREATE PROCEDURE sp_GetProductByIDDynSP( @productID INT )
AS
EXEC('SELECT Name, ProductNumber
      FROM Production.Product
      WHERE ProductID = ' + CONVERT(VARCHAR(5), @productID));
```

T-SQL standartlarına göre herhangi bir sorun yok. Ancak dinamik sorgulama fonksiyonu **EXEC** için bu bir sorundur. Yukarıdaki dinamik sorgu kullanımı hata üretecektir.

Tür dönüşümü işlemini **EXEC** fonksiyonunun dışında gerçekleştirmek gerekir.

Aynı işlemi gerçekleştiren aşağıdaki sorgu başarıyla çalışacaktır.

```
CREATE PROCEDURE sp_GetProductByIDDynSP( @productID INT )
AS
DECLARE @val VARCHAR(5) = CONVERT(VARCHAR(5), @productID);
EXEC('SELECT Name, ProductNumber
      FROM Production.Product
      WHERE ProductID = ' + @val);
```

Prosedürü tekrar çağıralım.

```
EXEC sp_GetProductByIDDynSP 1;
```

	Name	ProductNumber
1	Adjustable Race	AR-5381

EXEC fonksiyonunu kullanarak geçerli veritabanları üzerine farklı bir örnek oluşturalım.

```
USE AdventureWorks2012
GO
DECLARE @cmd VARCHAR(4000);
SET @cmd = 'EXEC spCurrDB';
SET @cmd = 'SELECT ''Geçerli Veritabanı: [''+D.NAME+'']'''
+ ' FROM master..sysdatabases d, master..sysprocesses p '
+ ' WHERE p.spid = @@SPID and p.dbid = d.dbid ';
EXEC (@cmd);
EXEC (N'USE master;' + @cmd);
EXEC (@cmd);
```

	(No column name)
1	Geçerli Veritabanı: [AdventureWorks2012]
	(No column name)
1	Geçerli Veritabanı: [master]
	(No column name)
1	Geçerli Veritabanı: [AdventureWorks2012]

Bu örnekte, ilk olarak geçerli veritabanı gösteriliyor. Daha sonra master veritabanı seçilerek geçerli veritabanı tekrar gösteriliyor ve son olarak ilk geçerli veritabanına dönülerek gösteriliyor.

SP_EXECUTESQL İLE DİNAMİK SORGU ÇALIŞTIRMAK

EXEC fonksiyonuna göre bazı durumlarda daha yüksek performanslı bir sistem prosedürüdür. Bu prosedür ile **EXEC** fonksiyonunun aksine, sorgular için bir çalıştırma planı oluşturulur. Bu nedenle, sonraki sorgulama işleminde daha performanslı bir sonuç elde edilir.

sp_executesql prosedürü dışarıdan parametre alabilir. Bu da sorgularda daha esnek ve avantajlı bir kullanım sağlayabilir.

Basit bir **SELECT** cümlesi ile kullanımı aşağıdaki gibidir.

```
EXECUTE sp_executesql N'SELECT * FROM Purchasing.PurchaseOrderHeader';
```

	PurchaseOrderID	RevisionNumber	Status	EmployeeID	VendorID	ShipMethodID	OrderDate	ShipDate	SubTotal
1	1	1	4	258	1580	3	2005-05-17 00:00:00.000	2005-05-26 00:00:00.000	201,04
2	2	1	1	254	1496	5	2005-05-17 00:00:00.000	2005-05-26 00:00:00.000	272,1015
3	3	1	4	257	1494	2	2005-05-17 00:00:00.000	2005-05-26 00:00:00.000	8847,30
4	4	1	3	261	1650	5	2005-05-17 00:00:00.000	2005-05-26 00:00:00.000	171,0765
5	5	1	4	251	1654	4	2005-05-31 00:00:00.000	2005-06-09 00:00:00.000	20397,30
6	6	1	4	253	1664	3	2005-05-31 00:00:00.000	2005-06-09 00:00:00.000	14628,075
7	7	1	4	255	1678	3	2005-05-31 00:00:00.000	2005-06-09 00:00:00.000	58685,55

Tam nesne ismi belirtilmesi gerektiğinde üç parçalı isimlendirme kuralı kullanılabilir. Bu sistem prosedürü ile üç parçalı isimlendirme şu şekilde yapılabilir.

```
EXEC AdventureWorks2012.dbo.sp_executesql N'EXEC sp_help';
```

SP_Executesql prosedürü ile aşağıdaki gibi girdi ve çıktı parametreleri de kullanılabilir.

```
DECLARE @SQL NVARCHAR(MAX),
        @ParmDefinition NVARCHAR(1024)
DECLARE @ListPrice MONEY = 2000.0,
        @LastProduct VARCHAR(64)
SET @SQL = N'SELECT @pLastProduct = MAX(Name)
            FROM AdventureWorks2012.Production.Product
            WHERE ListPrice >= @pListPrice'
SET @ParmDefinition = N'@pListPrice MONEY,
        @pLastProduct VARCHAR(64) OUTPUT'
EXECUTE sp_executesql @SQL, @ParmDefinition, @pListPrice = @
ListPrice,
        @pLastProduct = @LastProduct OUTPUT
SELECT [ListPrice >=] = @ListPrice, LastProduct = @LastProduct;
```

	List Price >=	LastProduct
1	2000,00	Touring-1000 Yellow, 60

Dinamik SQL ile tüm veritabanlarındaki tablo sayısını hesaplayalım.

```
DECLARE @SQL NVARCHAR(MAX), @dbName SYSNAME;
DECLARE DBcursor CURSOR FOR
    SELECT NAME FROM master.dbo.sysdatabases
    WHERE NAME NOT IN ('master','tempdb','model','msdb')
    AND DATABASEPROPERTYEX(NAME,'status') = 'ONLINE' ORDER BY NAME;
OPEN DBcursor; FETCH DBcursor INTO @dbName;
WHILE (@@FETCH_STATUS = 0)
    BEGIN
        DECLARE @dbContext NVARCHAR(256) = @dbName+'.dbo.'+'sp_executeSQL'
        SET @SQL = 'SELECT ''Database: ' + @dbName +
            ' TABLE COUNT'' = COUNT(*) FROM sys.tables';
        PRINT @SQL;
        EXEC @dbContext @SQL;
        FETCH DBcursor INTO @dbName;
    END;
CLOSE DBcursor; DEALLOCATE DBcursor;
```

Database: AdventureWorks2012 TABLE COUNT	
1	78
Database: DIJIBIL TABLE COUNT	
1	7
Database: DijiLabs TABLE COUNT	
1	2
Database: ReportServer TABLE COUNT	
1	34
Database: ReportServerTempDB TABLE COUNT	
1	13

DİNAMİK SQL İLE SIRALAMA İŞLEMİ

Bir dinamik sorgu içerisinde **ORDER BY** ile sıralama özelliği ekleyerek dinamik sıralama gerçekleştirebiliriz.

```
DECLARE @SQL NVARCHAR(MAX) = 'SELECT ProductID, Name, ListPrice, Color
    FROM Production.Product ORDER BY Name '
DECLARE @Collation NVARCHAR(MAX);
SET @Collation = 'COLLATE SQL_Latin1_General_CP1250_CS_AS'
SET @SQL = @SQL + @Collation
```

```
PRINT @SQL
EXEC sp_executeSQL @SQL;
```

	ProductID	Name	ListPrice	Color
1	1	Adjustable Race	0,00	NULL
2	879	All-Purpose Bike Stand	159,00	NULL
3	712	AWC Logo Cap	8,99	Multi
4	3	BB Ball Bearing	0,00	NULL
5	2	Bearing Ball	0,00	NULL
6	877	Bike Wash - Dissolver	7,95	NULL
7	316	Blade	0,00	NULL

SP_EXECUTESQL İLE STORED PROCEDURE KULLANIMI

SP_ExecutesQL ile de Sproc kullanılabilir. Bu işlemin **EXEC** fonksiyonu ile sproc oluşturmaktan bir farkı yoktur.

SP_ExecutesQL ile ürün araması gerçekleştiren bir Sproc oluşturalım.

```
CREATE PROCEDURE pr_UrunAra @ProductName VARCHAR(32) = NULL
AS
BEGIN
    DECLARE @SQL NVARCHAR(MAX)
    SELECT @SQL = ' SELECT ProductID, ProductName = Name,
                    Color, ListPrice ' + CHAR(10) +
                ' FROM Production.Product' + CHAR(10) +
                ' WHERE 1 = 1 ' + CHAR(10)
    IF @ProductName IS NOT NULL
        SELECT @SQL = @SQL + ' AND Name LIKE @pProductName'
    PRINT @SQL

    EXEC sp_executesql @SQL, N'@pProductName VARCHAR(32)', @ProductName
END
GO
```

Prosedürü çağıralım.

```
EXEC pr_UrunAra '%bike%';
```

	ProductID	ProductName	Color	ListPrice
1	879	All-Purpose Bike Stand	NULL	159,00
2	877	Bike Wash - Dissolver	NULL	7,95
3	876	Hitch Rack - 4-Bike	NULL	120,00
4	710	Mountain Bike Socks, L	White	9,50
5	709	Mountain Bike Socks, M	White	9,50

SP_EXECUTESQL İLE INSERT İŞLEMİ

Dinamik sorgudan dönen veriyi gerçek ya da geçici bir tabloda saklayarak, veri üzerinde yeni bir sorgu oluşturulabilir.

Geçici bir tablo oluşturalım ve dinamik sorgudan dönen değeri bu geçici tabloya ekleyelim.

```
CREATE TABLE #Product(ProductID int, ProductName varchar(64));
INSERT #Product
EXEC sp_executesql N'SELECT ProductID, Name FROM Production.Product';
SELECT * FROM #Product ORDER BY ProductName;
GO
DROP TABLE #Product;
```

	ProductID	ProductName
1	1	Adjustable Race
2	879	All-Purpose Bike Stand
3	712	AWC Logo Cap
4	3	BB Ball Bearing
5	2	Bearing Ball
6	877	Bike Wash - Dissolver
7	316	Blade

SP_EXECUTESQL İLE VERİTABANI OLUŞTURMAK

Dinamik sorguların sık kullanıldığı alanlardan biri de toplu program parçalarından oluşan sorgu script'leridir. Örneğin; sütunlarını sizin belirlediğiniz bir tablo oluşturmak ya da bir veritabanı oluşturmak için kullanılabilir.

Dinamik olarak basit bir veritabanı oluşturmak için **SP_Executesql** kullanalım.

```
CREATE PROC pr_CreateDB @DBName SYSNAME
AS
BEGIN
```

```
DECLARE @SQL NVARCHAR(255) = 'CREATE DATABASE ' + @DBName;  
EXEC sp_executesql @SQL;  
END;
```

Prosedürü çalıştırarak örnek bir veritabanı oluşturalım.

```
EXEC pr_CreateDB 'ornek_db';
```
