

SQL SERVER İLE XML

15

Teknolojide birçok standart ve yeni platformların gelişmesiyle farklı veri formatları ve platformlar arasında veri uyumluluğu, veri transferi gibi kavramlar önem kazandı. İki farklı nesne yapısının birbiri ile iletişime geçmesi için ara katman oluşturmak gerekir. Veriyi farklı bir formata dönüştürecek bir parser bu işi görebilir. Ancak veri dönüşümünün birden fazla formatlara yapılması gereken durumlarda söz konusu olabilir.

Bir e-ticaret sistemini düşünelim. Bu tür sistemler, hizmet sağladığı firmalara bazı verileri iletmesi gerekir. Anlaşmalı olduğu kargo firmasına sipariş edilen ve kargoya hazır hale gelen ürünleri iletmek için ilgili veriyi belirli bir formata dönüştürerek kargo firmasına ulaştırır. Kargo firması da bu veriyi kendi yazılım mimarisine göre alarak işler ve veritabanlarına kaydeder. Aynı şekilde e-ticaret sistemi de tüm yayın evlerinin yayındaki kitaplarının ya da beyaz eşya firmalarındaki ürünlerin satış fiyatı, ürün adı, ürün açıklaması, ürün resmi gibi tüm bilgileri belirli formatta alarak e-ticaret veritabanına kaydeder. Fark ettiyseniz, benzer işlemler için birbirinden farklı platformların entegrasyonu söz konusu oldu. Bu sistemde entegrasyon yapan firmaların sayısı onlarca olabilir. Ve entegrasyona katılan tüm firmaların yazılımları, veritabanları, işletim sistemleri, kullandıkları teknolojiler tamamen farklı olabilir. Aynı şekilde bu firmaların tamamı farklı veri formatları ile çalışıyor olabilir. Bazıları XML kullanıyor olabileceği gibi farklı formatlarda da veri aktarımı yapıyor olabilir.

XML, bu tür farklı platform ve teknolojilerin bir arada uyumlu olarak veri aktarımı, transferi yapabilmesi için geliştirilen bir standarttır.

XML'i İngilizce diline benzetebiliriz. Siz hangi ülke ve milletten olursanız olun diliniz farklı da olsa, Dünya'da en yaygın dil olarak kullanılan İngilizce'yi bilen tüm insanlarla konuşabilirsiniz. İngilizce lisan dili alanında bir standart ise, XML'de bilgisayar biliminde veri aktarımı, uyumluluk gibi konularda farklı platformları tek bir standart ile iletişim halinde tutmayı sağlayan bir teknolojidir.

Büyük veritabanı yönetim sistemlerinin tamamında desteklendiği gibi SQL Server'da XML veri formatına varsayılan olarak destek vermektedir ve tam uyumlu şekilde çalışmaktadır.

XML

XML (*eXtensible Markup Language*) standardı **W3C** (*World Wide Web Consortium*) tarafından standart haline getirilen ve HTML'in de tasarımcısı olan Tim Berners Lee tarafından tasarlanmış bir işaretleme dilidir. XML'in Türkçe anlamı Genişletilebilir İşaretleme Dili'dir.

XML'de veriler tam anlamıyla genişletilebilir şekilde etiketleme sistemi ile işaretlenir.

Örneğin, KodLab yayınevine ait kitaplar XML formatında şu şekilde tutulabilir.

```
<root>
  <kitap kitapID=123> İleri Seviye SQL Server T-SQL </kitap>
  <kitap kitapID=124> İleri Seviye Android Programlama </kitap>
</root>
```

XML büyük-küçük harf duyarlı bir işaretleme dilidir. <KodLab> ile <kodlab> aynı değildir. Her XML dokümanında bir kök dizin olmak zorundadır. Kök dizin içerisinde ise kök dizine ait elemanlar bulunur. <kitap> bu örnekte bir alt elemandır. <kitap> içerisindeki kitapID ise bir **attribute** (öznitelik)'tür.

XML VERİ TİPİNİ KULLANMAK

XML'in bir standart haline gelmesinden sonra SQL standartları tarafından desteklenir hale gelmesiyle SQL Server'da gelişmiş seviyede XML desteği sunar. XML metodları, veri tipi ve şema gibi birçok özelliğiyle birlikte desteklenen XML ile SQL Server'da şu şekilde faydalanılabilir.

- XML tipinde bir değişken tanımlamak için kullanılabilir.
- Tablo oluştururken XML tipinde sütunlar oluşturulabilir.
- Stored Procedure'lere girdi-çıkı parametreleri olarak kullanılabilir.
- Kullanıcı tanımlı fonksiyonlara girdi parametre ya da geri dönüş tipi olarak kullanılabilir.

XML, yapısal olarak diğer veri tiplerinden farklıdır. Bazı durumlarda farklı veri tiplerine dönüştürülmesi gerekir. Dönüştürme işlemi için **CAST** ya da **CONVERT** fonksiyonları kullanılabilir.

XML, **tip tanımlı** (*typed*) ve **tip tanımsız** (*untyped*) olarak ikiye ayrılır. Bu bölümde bu iki XML tipi için de detaylı örnekler yaparak, normal sorgu ve Stored Procedure ile kullanımlarını inceleyeceğiz.

XML TİPİ İLE DEĞİŞKEN VE PARAMETRE KULLANMAK

SQL Server, XML teknolojisine native olarak destek vermektedir. XML bir veri tipi kullanılabileceği gibi, bir değişken olarak da kullanılabilir. XML'in değişken ve parametre olarak kullanılabilmesi, SQL Server veritabanı programlama esnekliğini ve gücünü tam olarak kullanabilmek anlamına gelir.

Örnek ve uygulamalarda birçok farklı şekilde kullanılacak XML'in değişken olarak nasıl kullanılabileceğine basit bir örnek verelim.

```
DECLARE @xml_veri VARCHAR(MAX);
SET @xml_veri = '
<kitaplar>
  <kitap>İleri Seviye SQL Server T-SQL</kitap>
  <kitap>İleri Seviye Android Programlama</kitap>
  <kitap>Java SE</kitap>
</kitaplar>';
SELECT CAST(@xml_veri AS XML);
SELECT CONVERT(XML, @xml_veri);
PRINT @xml_veri;
```

(No column name)	
1	<kitaplar><kitap>İleri Seviye SQL Server T-SQL</kitap><kitap>İleri Seviye Android Programlama</kitap><kitap>Java SE</kitap></kitaplar>
(No column name)	
1	<kitaplar><kitap>İleri Seviye SQL Server T-SQL</kitap><kitap>İleri Seviye Android Programlama</kitap><kitap>Java SE</kitap></kitaplar>

```
(1 row(s) affected)

(1 row(s) affected)

<kitaplar>
  <kitap>İleri Seviye SQL Server T-SQL</kitap>
  <kitap>İleri Seviye Android Programlama</kitap>
  <kitap>Java SE</kitap>
</kitaplar>
```

Metin olarak oluşturulan XML veri değişkeni, sonrasında **CAST** ve **CONVERT** ile XML veri tipine dönüştürülmektedir.

VARCHAR olarak tanımlanan verinin XML veri tipine dönüştürülmesi bir seçenektir. Değişken, doğrudan XML olarak da aşağıdaki gibi tanımlanabilir.

```
DECLARE @xml_veri XML
SET @xml_veri = '
<kitaplar>
  <kitap>İleri Seviye SQL Server T-SQL</kitap>
  <kitap>İleri Seviye Android Programlama</kitap>
  <kitap>Java SE</kitap>
</kitaplar>'
SELECT CAST(@xml_veri AS VARCHAR(MAX))
SELECT CONVERT(VARCHAR(MAX), @xml_veri)
PRINT CONVERT(VARCHAR(MAX), @xml_veri)
```

	(No column name)
1	<kitaplar><kitap>İleri Seviye SQL Server T-SQL</kitap><kitap>İleri Seviye Android Programlama</kitap><kitap>Java SE</kitap></kitaplar>
	(No column name)
1	<kitaplar><kitap>İleri Seviye SQL Server T-SQL</kitap><kitap>İleri Seviye Android Programlama</kitap><kitap>Java SE</kitap></kitaplar>

```
(1 row(s) affected)
```

```
(1 row(s) affected)
<kitaplar><kitap>İleri Seviye SQL Server T-SQL</kitap><kitap>İleri Seviye Android Programlama</kitap><kitap>Java SE</kitap></kitaplar>
```

TİP TANIMSIZ XML VERİ İLE ÇALIŞMAK (UNTYPED)

XML verinin şeması ya da yapısının denetlenmemesi gereken durumlarda kullanılır. Tip tanımsız kullanım ile XML veri tipine sahip sütun, sadece verinin XML olup olmadığını denetlemek için kullanılır. XSD tanımlaması, XML yapısının SQL Server tarafından tanınmasını sağlar. SQL Server, XML yapıyı ne kadar iyi tanırsa sorgu performansını buna göre optimize ederek daha performanslı XML kullanımı gerçekleştirilmesini sağlar. Tip tanımsız XML kullanımında bu tanımlamaların bulunmaması performansı olumsuz yönde etkileyecektir.

Tip tanımsız XML veri içeren bir tablo oluşturalım.

```
CREATE TABLE OzGecmis
(
    AdayID INT IDENTITY PRIMARY KEY,
    AdayOzGecmis XML
);
```

HumanResources.JobCandidate içerisinde, iş başvurusu yapan adayların kayıtları bulunur. Adayların özgeçmiş bilgilerinin yapısından dolayı, özel bir XML olarak tutmak daha kolay bir kullanımdır. Aday öz geçmiş bilgileri çok önemli ve sık kullanılacak veriler olmadığı için yüksek sorgu performansına da ihtiyacı yoktur. Bu nedenle tip tanımlı ya da tanımsız olması sorun teşkil etmeyecektir.

OzGecmis tablosundaki **AdayOzGecmis** sütununa, **HumanResources.JobCandidate** tablosundaki **Resume** sütunundan tüm kayıtları seçerek ekleyelim.

```
INSERT INTO OzGecmis(AdayOzGecmis)
SELECT Resume FROM HumanResources.JobCandidate;
```

OzGecmis tablosunun içerik eklendikten sonraki halini görüntüleyelim.

```
SELECT * FROM OzGecmis;
```

AdayID	AdayOzGecmis
1	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix>M.</ns:Name.Prefix><...
2	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix>M.</ns:Name.Prefix><...
3	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix>M.</ns:Name.Prefix><...
4	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Pend...
5	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Shen...
6	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Tai</...
7	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix>H.M.</ns:Name.Prefix>...

AdayID değeri 1 olan adayın özgeçmişini görüntüleyelim.

```
DECLARE @Aday XML;
SELECT @Aday = AdayOzGecmis FROM OzGecmis WHERE AdayID = 1;
SELECT @Aday AS Resume;
```

Resume
1 <ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix>M.</ns:Name.Prefix><ns:Name.First>Thiem</...

Görüntülenen aday XML dokümanındaki Name alanı aşağıdaki gibidir.

```
<ns:Name>
  <ns:Name.Prefix>M.</ns:Name.Prefix>
  <ns:Name.First>Thierry</ns:Name.First>
  <ns:Name.Middle />
  <ns:Name.Last>D' Hers</ns:Name.Last>
  <ns:Name.Suffix />
</ns:Name>
```

XML veri tipine sahip bir tablo oluşturup, XML sonuç dönen sorgular ile içerikleri görüntüleyebildik. Genel olarak bu tür işlemler prosedürel olarak gerçekleştirilir. Prosedür içerisinde gerekli düzenleme ve filtrelemeler yapılabildiği ve her seferinde uzun kod satırları yazmaya gerek olmadığı için prosedürler XML gibi büyük ve karmaşık veriler için kullanılabilir nesnelerdir.

Yukarıda yapılan sorguyu Stored Procedure ile gerçekleştirelim.

```
CREATE PROCEDURE AdayEkle( @Aday XML )
AS
INSERT INTO OzGecmis(AdayOzGecmis) VALUES (@Aday);
```

AdayEkle prosedürü, dışarıdan adayların XML bilgilerini alarak **OzGecmis** tablosuna kaydediyor. Şimdi, bir adayın XML bilgilerini **AdayEkle** prosedürü ile **OzGecmis** tablosuna kayıt edelim.

```
DECLARE @AdayProc XML;
SELECT  @AdayProc = Resume FROM HumanResources.JobCandidate
        WHERE JobCandidateID = 8;
EXEC AdayEkle @AdayProc;
```

JobCandidateID değeri 8 olan adayın özgeçmiş bilgilerini **AdayEkle** prosedürünü kullanarak **OzGecmis** tablosuna kayıt ettik.

Son kayıt eklendikten sonra **OzGecmis** tablosunun görünümü aşağıdaki gibi olacaktır.

```
SELECT * FROM OzGecmis ORDER BY AdayID DESC;
```

	AdayID	AdayOzGecmis
1	10	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Peng</ns:Name...
2	9	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>an</ns:Name...
3	8	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix><ns:Name.Prefix><ns:Name...
4	7	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix><ns:Name.Prefix><ns:Name.Fi...
5	6	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Tai</ns:Name.Fir...
6	5	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Shenoda</ns:Na...
7	4	<ns:Resume xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"><ns:Name><ns:Name.Prefix /><ns:Name.First>Peng</ns:Name...

TİP TANIMLI XML VERİ İLE ÇALIŞMAK (TYPED)

Tip tanımlı XML kullanımında, bir XML veri, XML şema tanımları (XSD) ile denetlenir. XML şemaları, verinin yapısını belirtir ve kısıtlamalar, hangi sütuna ne türden veri girileceği gibi denetleme şartlarını sağlayan performans artırıcı özelliklerdir. XML verisini SQL Server'a doğru tanıttacağı için performans olarak etkili bir kullanımdır.

Tip tanımlı XML için veritabanında XML şema koleksiyonu oluşturulması gerekir. XML şema koleksiyonlarını detaylarıyla anlattığımız kısmı inceleyebilirsiniz.

Tip Tanımsız XML konusunu işlerken verdiğimiz **OzGecmis** tablosu örneğini, Tip Tanımlı XML yöntemi ile nasıl oluşturacağımızı ve yöneteceğimizi inceleyelim. Tabloyu yeniden oluşturmak için **DROP TABLE** ile silin.

```
CREATE TABLE OzGecmis
(
    AdayID INT IDENTITY PRIMARY KEY,
    AdayOzGecmis XML (HumanResources.HRResumeSchemaCollection)
);
```

Tip tanımlı XML kullanımında önemli bir farklılık, **AdayOzGecmis** sütunundaki XML veri tipinin yanında, parantez içerisinde bir de XML şema koleksiyonu kullanılmasıdır.

Kullanılan XML şema koleksiyonu **HumanResources** şeması içerisindeki **HRResumeSchemaCollection**'dur.

XML şema koleksiyonlarını işlediğimiz bölümü detaylı inceleyerek bu konuyu daha iyi kavrayabilirsiniz.

OzGecmis tablosuna **SELECT** ile veri girişini gerçekleştirelim.

```
INSERT INTO OzGecmis (AdayOzGecmis)
SELECT Resume FROM HumanResources.JobCandidate;
```

JobCandidateID değeri 8 olan adayın XML özgeçmişini prosedür kullanarak **OzGecmis** tablosuna ekledik.

```
SELECT * FROM OzGecmis;
```

XML VERİ TİPİ İLE ÇOKLU VERİ İŞLEMLERİ

Veri tipi XML olan bir sütun tanımlamak birçok durumda gerekli olabilir. Bazen, SQL Server tablo yapısında genel ve tüm ürünlerde ortak olan ürünlerin bilgilerini tutmak, geri kalan birbirinden farklı ürün özelliklerini de XML şema koleksiyonları denetiminde, veritabanında XML olarak tutmak istenebilir.

XML veri tipi ve diğer SQL Server veri tiplerinden oluşan bir tablo oluşturalım. Bu tabloda, kitap bilgileri yer alsın. Kitapların sadece **KitapID** ve kitap isimlerini tutan iki adet XML olmayan sütun bulunsun. Ek olarak, birden fazla sütun değeri içerecek XML veri tipine sahip sütun bulunsun.

```
CREATE TABLE Kitaplar
(
    KitapID INT IDENTITY(1,1) PRIMARY KEY,
    Ad VARCHAR(60),
    KitapDetay XML
);
```

Kitaplar tablosunda temel bazı özellikler haricinde sadece **KitapDetay** sütunu var. **KitapDetay** sütunu, Kitaplar tablosuna esneklik katmaktadır. Yapısal olarak birçok farklı veri tek bir sütuna XML olarak eklenebilir.

KitapDetay sütunu, XML şema koleksiyonu oluşturularak bu koleksiyonlar ile denetlenebilir.

Çoklu veri ekleme yöntemi ile Kitaplar tablosuna iki kitap kaydı ekleyelim.

```
INSERT INTO Kitaplar(Ad, KitapDetay)
VALUES('İleri Seviye SQL Server T-SQL',
    '<Kitap>
    <Yazar>Cihan Özhan</Yazar>
    <ISBN>978-975-17-2268-7</ISBN>
    <Ozet>İleri seviye SQL Server kitabı.</Ozet>
    <SayfaSayisi>500</SayfaSayisi>
```

```
<BaskiSayisi>3</BaskiSayisi>
</Kitap>'), (
'İleri Seviye Android Programlama',
'<Kitap>
<Yazar>Kerim FIRAT</Yazar>
<ISBN>978-975-17-2243-8</ISBN>
<Ozet>İleri seviye Android programlama kitabı.</Ozet>
<SayfaSayisi>800</SayfaSayisi>
<BaskiSayisi>5</BaskiSayisi>
</Kitap>');
```

Eklenen veriyi görüntüleyelim.

```
SELECT * FROM Kitaplar;
```

	KitapID	Ad	KitapDetay
1	1	İleri Seviye SQL Server T-SQL	<Kitap><Yazar>Cihan Özhan</Yazar><ISBN>978-975-17-2268-7</ISBN><Ozet>İleri seviye SQL Server kitabı.</Ozet><SayfaSayisi>...
2	2	İleri Seviye Android Programlama	<Kitap><Yazar>Kerim FIRAT</Yazar><ISBN>978-975-17-2243-8</ISBN><Ozet>İleri seviye Android programlama kitabı.</Ozet><Say...

Kitap ekleme işlemini prosedürel hale getirelim.

```
ALTER PROC KitapEkle
(
    @Ad VARCHAR(60),
    @KitapDetay XML
)
AS
BEGIN
    INSERT INTO Kitaplar(Ad, KitapDetay)
    VALUES(@Ad, @KitapDetay)
END;
```

KitapEkle isimli prosedürü kullanarak veritabanına bir kayıt ekleyelim.

```
DECLARE @KD XML;
SET @KD = '<Kitap>
<Yazar>Kerim FIRAT</Yazar>
<ISBN>978-975-17-2243-8</ISBN>
<Ozet>Java Standard Edition eğitim kitabı.</Ozet>
<SayfaSayisi>700</SayfaSayisi>
<BaskiSayisi>4</BaskiSayisi>
</Kitap>';
EXEC KitapEkle 'Java SE', @KD;
```

XML ŞEMA KOLEKSİYONLARI

XML standardının ilk geliştiği yıllarda **DTD** (*Document Type Definition*) kullanılıyordu. DTD yapısal olarak zor ve karmaşık bir kullanıma sahipti.

DTD

DTD (*Document Type Definition*), XML dokümanlarının yapısal kurallarını belirleyen bir yapı sağlar. Bir XML içerisinde hangi elementlerin ve her elementin içerisinde kaç attribute'ü olacağını belirlemek için kullanılır.

DOCTYPE BİLDİRİMİ

XML dokümanlarının hangi DTD'ye uyacağını belirtmek için kullanılır. XML'de bildirim işlemi `<! ve >` tag açma-kapama işaretleri ile gerçekleştirilir.

Bir DTD dokümanının uzantısı `dtd` olmalıdır.

Aşağıda, dijibil root elementine sahip bir XML dokümanı için `dijibil.dtd` bildirimi yapılmaktadır.

```
<!DOCTYPE dijibil SYSTEM "http://www.dijibil.com/dtds/dijibil.dtd">
```

Bu işlemde ilk olarak root elementin adı verilir. Eğer daha önce yapılmış bir DTD kullanılmak isteniyorsa, **SYSTEM** yerine **PUBLIC** kullanılır.

DTD dört tip bildirim içerir.

- ELEMENT
- ATTLIST
- ENTITY
- NOTATION

ELEMENT BİLDİRİMİ

XML dokümanlarında kullanılacak elementleri tanımlamak için kullanılır.

```
<!ELEMENT isim CATALOG>
```

ya da

```
<!ELEMENT isim (content)>
```

İçinde hiç bir şey kullanılmayan bir element için;

```
<!ELEMENT br EMPTY>
```

Bir element herhangi bir element içerebilirse **EMPTY** yerine **ANY** kullanılır.

```
<!ELEMENT br ANY>
```

XML içerisinde açıklama satırı gibi yazmış iseniz **PCDATA** komutuna aşına olmalısınız.

Bir elementin içinde sadece yazı olması gerekiyorsa;

```
<!ELEMENT soyisim (#PCDATA)>
```

XML içerisinde birden fazla element kullanmak için;

```
<!ELEMENT dijibil (isim, soyisim)>
```

Çoklu element kullanımında SQL Server sütunlarında olduğu gibi sıralama önemlidir.

XML içerisinde bir element kullanılırsa diğerinin kullanılmaması istendiğinde;

```
<!ELEMENT dijibil (isim|soyisim)>
```

ATTLIST BİLDİRİMİ

Elementlerin attribute tanımlamaları için kullanılır.

Söz Dizimi:

```
<!ATTLIST element_ismi
    attribute_ismi attribute_tipi
    attribute_varsayilan varsayilan_deger>
```

İki attribute'ü olan dijibil root elementini tanımlayalım.

```
<!ATTLIST dijibil
    UserID CDATA #REQUIRED
    UserNO  CDATA #IMPLIED>
```

- **CDATA:** Karakter verisi anlamına gelir.
- **#REQUIRED:** Bu attribute'ün kullanımının zorunlu olduğunu belirtir.
- **#IMPLIED:** Bu attribute'ün kullanımının zorunlu değil, isteğe bağlı olduğunu belirtir.

Attribute'ler istenen değerleri seçenek olarak sunabilir. Bu şekilde kullanılarak attribute'ün sadece belirli değerleri alması sağlanabilir.

Genel olarak yazılımlarda cinsiyet bilgileri tutulur. Bu bilgileri, XML ortamındaki attribute'lerde aşağıdaki gibi seçenekli hale getirilebilir.

```
<!ATTLIST kullanıcı cinsiyet (bay | bayan) #IMPLIED>
```

Yukarıdaki kullanım ile kullanıcının, bu attribute'e sadece 'bay' ya da 'bayan' değerleri vermesi sağlanmış olur.

Attribute'lerde varsayılan değer ataması ise şu şekilde gerçekleşir.

```
<!ATTLIST kullanıcı cinsiyet (bay | bayan) "bay" #IMPLIED>
```

Yukarıdaki varsayılan değer örneğinde, iki seçimli değer için varsayılan olarak 'bay' değeri atanacaktır.

XML ŞEMA KOLEKSİYONLARI HAKKINDA BİLGİ ALMAK

XML şemaları hakkında bilgi almak için `xml_schema_collections` sistem kataloğunu kullanabilirsiniz. Bu katalog ile üstünde çalıştığınız veritabanında var olan şema koleksiyonlarını listeleyebilirsiniz.

Kullandığımız **AdventureWorks** veritabanına ait şema kataloglarını listeleyelim.

```
SELECT * FROM sys.xml_schema_collections;
```

	xml_collection_id	schema_id	principal_id	name	create_date	modify_date
1	1	4	NULL	sys	2009-04-13 12:59:13.390	2011-11-04 21:07:51.970
2	65536	6	NULL	AdditionalContactInfoSchemaCollection	2012-03-14 13:14:18.920	2012-03-14 13:14:18.930
3	65537	6	NULL	IndividualSurveySchemaCollection	2012-03-14 13:14:18.953	2012-03-14 13:14:18.953
4	65538	5	NULL	HRResumeSchemaCollection	2012-03-14 13:14:18.980	2012-03-14 13:14:18.980
5	65539	7	NULL	ProductDescriptionSchemaCollection	2012-03-14 13:14:19.027	2012-03-14 13:14:19.037
6	65540	7	NULL	ManuInstructionsSchemaCollection	2012-03-14 13:14:19.060	2012-03-14 13:14:19.060
7	65541	9	NULL	StoreSurveySchemaCollection	2012-03-14 13:14:19.110	2012-03-14 13:14:19.110

XML yapısında önemli bir yere sahip isim uzayları (*namespace*) hakkında bilgi almak için **xml_schema_namespaces** sistem kataloğunu kullanabilirsiniz.

```
SELECT * FROM sys.xml_schema_namespaces;
```

	xml_collection_id	name	xml_namespace_id
1	1	http://www.w3.org/2001/XMLSchema	1
2	1	http://schemas.microsoft.com/sqlserver/2004/sqltyp...	2
3	1	http://www.w3.org/XML/1998/namespace	3
4	65536	http://schemas.microsoft.com/sqlserver/2004/07/a...	1
5	65536	http://schemas.microsoft.com/sqlserver/2004/07/a...	2
6	65536	http://schemas.microsoft.com/sqlserver/2004/07/a...	3
7	65537	http://schemas.microsoft.com/sqlserver/2004/07/a...	1

XML_SCHEMA_NAMESPACE İLE ŞEMA KOLEKSİYONLARINI LİSTELEMEK

XML şema koleksiyonları karmaşık bir yapıya sahiptir. SQL Server yeteneklerini kullanarak bir XML şema koleksiyona erişebilmek önemli bir gereksinimdir.

Şimdi, bazı örnekler yaparak **AdventureWorks** veritabanı ile ilgili bazı şemaların XML içeriklerini listeleyelim.

Production.ProductDescriptionSchemaCollection şema koleksiyonunu görüntüleyelim.

```
SELECT
xml_schema_namespace(
    N'Production',
    N'ProductDescriptionSchemaCollection');
```

Sonuç ekranındaki XML bağlantısı açıldığında, şema koleksiyonunun XML içeriğini görüntülenecektir.

XQuery konusuna daha sonra değineceğiz. Ancak temel olarak kullanımı görmeniz için bir şema koleksiyonu XQuery ile listeleyelim.

Söz Dizimi:

```
SELECT xml_schema_namespace(
    N'SchemaName',
    N'XmlSchemaCollectionName')
    .query('/xs:schema[@targetNamespace="TargetNameSpace"]');
```

Production şeması içerisindeki **ProductDescriptionSchemaCollection** isimli şema koleksiyonunu görüntüleyelim.

```
SELECT xml_schema_namespace(
    N'Production',
    N'ProductDescriptionSchemaCollection').query('
/xs:schema[@targetNamespace="http://schemas.microsoft.com/
sqlserver/2004/07/adventure-works/ProductModelWarrAndMain"]');
```

Aynı işlemi farklı bir şekilde de gerçekleştirebiliriz.

```
SELECT xml_schema_namespace(
    N'Production',
    N'ProductDescriptionSchemaCollection', N'http://schemas.microsoft.
com/sqlserver/2004/07/adventure-works/ProductModelWarrAndMain');
```

Var olan XML şema koleksiyonları görüntülemeyi öğrenmek, kendi XML şema koleksiyonlarımızı oluşturmayı ve yönetmeyi daha kolay hale getirdi. Artık kendi koleksiyonlarımızı oluşturarak yönetebiliriz.

XML ŞEMA KOLEKSİYONU OLUŞTURMAK

Tüm nesne oluşturma söz dizimleri bu işlem için de geçerlidir. **CREATE** ile bir şema koleksiyonu oluşturulabilir.

Söz Dizimi:

```
CREATE XML SCHEMA COLLECTION [sql_server_sema,] koleksiyon_ismi
AS { sema_metni | sema_metnini_iceren_degisken }
```

Örnek bir Söz Dizimi:

```
CREATE XML SCHEMA COLLECTION EmployeeSchema
AS'<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element >
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element />
        <xsd:element />
        <xsd:element />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>'
```

XML şema koleksiyonu oluşturmak için **AdventureWorks** veritabanında **Production** şeması içerisindeki **ProductDescriptionSchemaCollection** isimli şema koleksiyonunun XML kodunu elde ederek farklı bir isimde tekrar oluşturabiliriz.

XML kaynağı elde etmek için;

```
SELECT xml_schema_namespace(
  N'Production',
  N'ProductDescriptionSchemaCollection', N'http://schemas.microsoft.
com/sqlserver/2004/07/adventure-works/ProductModelWarrAndMain');
```

Bir bağlantı şeklinde gelen sonuca tıklayarak görüntülenen XML kodu kopyalıyoruz. Kopyalanan kodu aşağıdaki şema koleksiyonu oluşturma kodunun sonunda bulunan iki tek tırnak arasına yapıştırıyoruz.

```
CREATE XML SCHEMA COLLECTION yeniXMLSemaKoleksiyon AS ''
```

Kod görünümü aşağıdaki gibi olacaktır. Kodlar seçili hale getirilip çalıştırıldığında **yeniXMLSemaKoleksiyon** isminde yeni bir XML şema koleksiyonuna sahip oluruz.

```
CREATE XML SCHEMA COLLECTION yeniXMLSemaKoleksiyon AS
'\<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:t="http://schemas.microsoft.com/sqlserver/2004/07/
```



```

adventure-works/ProductModelWarrAndMain" targetNamespace="http://
schemas.microsoft.com/sqlserver/2004/07/adventure-works/
ProductModelWarrAndMain" elementFormDefault="qualified">
  <xsd:element name="Maintenance">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
            <xsd:element name="NoOfYears" type="xsd:string" />
            <xsd:element name="Description" type="xsd:string" />
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Warranty">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
            <xsd:element name="WarrantyPeriod" type="xsd:string" />
            <xsd:element name="Description" type="xsd:string" />
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>';

```

Yeni şema koleksiyonumuzu sorgulamak için;

```

SELECT xml_schema_namespace(
  N'dbo',N'yeniXMLSemaKoleksiyon', N'http://schemas.microsoft.com/
sqlserver/2004/07/adventure-works/ProductModelWarrAndMain');

```

Şema koleksiyonumuz varsayılan olarak dbo şeması içerisinde oluşturulduğu için **SELECT** sorgusunda **dbo** şema adını belirttik.

XML ŞEMA KOLEKSİYONU DEĞİŞTİRMEK

XML şema koleksiyonunu değiştirmek komut olarak diğerleriyle aynıdır. **ALTER** ifadesi ile değişiklik gerçekleştirilebilir. Ancak diğer nesnelere göre şema koleksiyonlarında **ALTER** işlemi bir kısıtlamaya sahiptir. Şema koleksiyonlarında **ALTER** sadece yeni kısımlar eklemek için kullanılabilir.

Söz Dizimi:

```
ALTER XML SCHEMA COLLECTION [sql_server_sema,] koleksiyon_ismi
ADD { sema_metni | sema_metnini_iceren_degisken }
```

XML ŞEMA KOLEKSİYONUNU KALDIRMAK

XML şema koleksiyonlarını kaldırmak diğer nesneler gibi gerçekleştirilir.

Söz Dizimi:

```
DROP XML SCHEMA COLLECTION [ sql_server_sema .] koleksiyon_ismi
```

Oluşturduğumuz **yeniXMLSemaKoleksiyon** isimli şema koleksiyonunu kaldıralım.

```
DROP XML SCHEMA COLLECTION yeniXMLSemaKoleksiyon;
```

XML VERİ TİPİ METODLARI

XML veri üzerinde işlemler gerçekleştirilecek bazı metotlara sahiptir. Bu metotlar benzersiz özelliklere sahip olduğu gibi, kendi içlerinde söz dizimleri de farklılık gösterir. XML verinin sorgulanması, işlenmesi ve dönüştürülmesi için fonksiyonel özellikler sağlar.

XML metotları için kullanacağımız tablo ve kayıtları oluşturalım.

```
CREATE TABLE Magazalar
(
    MagazaID INT PRIMARY KEY,
    Anket_UnTyped XML,
    Anket_Typed XML(Sales.StoreSurveySchemaCollection)
);
```

Magazalar ismindeki tablonun XML veri tipindeki **Anket_Typed** sütununda şema koleksiyonu olarak **Sales.StoreSurveySchemaCollection**’u kullanıyoruz.

Magazalar tablosuna, **AdventureWorks** veritabanındaki **Sales.Store** tablosundan veri aktaracağız.

```
INSERT INTO Magazalar
VALUES
(292, '<MagazaAnket>
  <YillikSatis>145879</YillikSatis>
  <YillikGelir>79277</YillikGelir>
  <BankaAd>HIC Bank</BankaAd>
  <IsTuru>CO</IsTuru>
  <AcilisYil>2005</AcilisYil>
  <Uzmanlik>Technology</Uzmanlik>
  <Markalar>2</Markalar>
  <Internet>ISDN</Internet>
  <CalisanSayisi>14</CalisanSayisi>
  <Urunler Tip="Yazilim">
    <Urun>Mobil</Urun>
    <Urun>Masaüstü</Urun>
    <Urun>Sistem</Urun>
    <Urun>Web</Urun>
  </Urunler>
  <Urunler Tip="Eğitim">
    <Urun>Android</Urun>
    <Urun>Oracle</Urun>
    <Urun>Java</Urun>
  </Urunler>
</MagazaAnket>'
```

```
(SELECT Demographics FROM Sales.Store WHERE BusinessEntityID = 292));
```

Veri aktarımı tamamlandı. **Magazalar** tablosundaki veriyi görüntüleyelim.

```
SELECT * FROM Magazalar;
```

MagazaID	Anket_UnTyped	Anket_Typed
1 292	<MagazaAnket><YillikSatis>145879</YillikSatis><YillikGelir>79277</YillikGelir>...	<StoreSurvey xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-...

XML.QUERY

XQuery'nin uygulanmasını sağlar. XML veriye, XQuery sorgularıyla ulaşarak birden fazla parçasının sonuç olarak döndürülmesi için kullanılır. XQuery ile `doc()` ya da `collection()` gibi fonksiyonlarla bir dosyadan ya da bellek üzerindeki bir değişkenden okuma yapılabilir.

Söz Dizimi:

```
nesne.query('XQuery')
```

XQuery ile XML içerisindeki tüm ürünleri listeleyelim.

```
SELECT Anket_UnTyped.query('/MagazaAnket/Urunler/Urun') AS Urun
FROM Magazalar;
```

Sorgu sonucundaki kayıtları görüntüleyebilmek için bağlantı formatındaki XML sonucuna tıklamanız yeterlidir.

```
<Urun>Mobil</Urun>
<Urun>Masaüstü</Urun>
<Urun>Sistem</Urun>
<Urun>Web</Urun>
<Urun>Android</Urun>
<Urun>Oracle</Urun>
<Urun>Java</Urun>
```

XQuery ile XML içerisindeki mağaza anketlerini listeleyelim.

```
SELECT
    Anket_UnTyped.query('/MagazaAnket') AS UnTyped_Info,
    Anket_Typed.query('declare namespace ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/StoreSurvey";/ns:StoreSurvey')
    AS Typed_Info
FROM Magazalar;
```

	UnTyped_Info	Typed_Info
1	<MagazaAnket><Yillik Satış>145879</Yillik Satış><Yillik Gelir>79277</Yillik Gelir><Banka Ad>HİC Bank</Banka Ad>...	<StoreSurvey xmlns="http://schemas.microsoft.com/sqlserve...

Mağaza anketlerini listelerken, query ile namespace kullandığımıza dikkat edin. **AdventureWorks** veritabanındaki **StoreSurvey** isimli namespace'i kullandık.

Şimdi de, aynı yöntemi kullanarak sadece **YillikSatis** bilgisini alalım.

```
SELECT
    Anket_UnTyped.query('/MagazaAnket/YillikSatis') AS UnTyped_Info,
    Anket_Typed.query('declare namespace ns="http://schemas.
microsoft.com/sqlserver/2004/07/adventure-works/StoreSurvey";/
ns:StoreSurvey/ns:AnnualSales') AS Typed_Info
FROM Magazalar;
```

	UnTyped_Info	Typed_Info
1	<YillikSatis>145879</YillikSatis>	ns:AnnualSales xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/StoreSurvey">800000...

Tüm ürünleri, tip kategorili olarak listeleyelim.

```
SELECT
    Anket_UnTyped.query('/MagazaAnket/Urunler') AS Urunler
FROM Magazalar;
```

	Urunler
1	<Urunler Tip="Yazilim"><Urun>Mobil</Urun><Urun>Masaüstü</Urun><Urun>Sistem</Urun><Urun>Web</Urun></Urunler><Urunler Tip="Egitim"><Urun>Android</Urun><Urun>Oracle...

Sadece, Yazılım tipindeki ürünleri listeleyelim.

```
SELECT
    Anket_UnTyped.query('/MagazaAnket/Urunler[@Tip="Yazilim"]') AS Yazilim
FROM Magazalar;
```

	Yazilim
1	<Urunler Tip="Yazilim"><Urun>Mobil</Urun><Urun>Masaüstü</Urun><Urun>Sistem</Urun><Urun>Web</Urun></Urunler>

Bazen XML içerisinde arama yaparak, aranan metni içeren kayıtlar bulunmak istenebilir. XML içerisindeki ürünlerde **'Mob'** metnini içeren kayıtları bulalım.

```
SELECT Anket_UnTyped.query('
    for $b in /MagazaAnket/Urunler/Urun
    where contains($b, "Mob")
    return $b
') AS Urun
FROM Magazalar;
```

Kullandığımız XML'de bu şartları, sadece Mobil kategorisi sağladığı için tek kayıt bulundu.

Urun	
1	<Urun>Mobil</Urun>

XML.EXIST

Özel bir veri tipinin var olup olmadığını test eder. Test edilen XML örneğinde, belirli bir düğüm ya da attribute girişi olup olmadığına bakar. Dışarıdan aldığı ifadenin XML düğümü içerisinde bulunması halinde **true** (1), bulunamaması halinde **false** (0) değerini döndürür.

XML içerisinde, kaç adet Urunler listesinin bulunduğunu sorgulayalım.

```
SELECT Anket_UnTyped.exist('(//MagazaAnket/Urunler)[2]') AS Sales_UnTyped
FROM Magazalar;
```

Sales_UnTyped	
1	1

XML'in yapısını incellerseniz, iki adet farklı tiplere sahip **urunler** listesi bulunur.

```
<Urunler Tip="Yazilim">
  <Urun>Mobil</Urun>
  <Urun>Masaüstü</Urun>
  <Urun>Sistem</Urun>
  <Urun>Web</Urun>
</Urunler>
<Urunler Tip="Egitim">
  <Urun>Android</Urun>
  <Urun>Oracle</Urun>
  <Urun>Java</Urun>
</Urunler>
```

Eğer kare parantez içerisinde 3 değerini parametre olarak gönderseydik, sorgu 0 değerini döndürecekti. Çünkü XML içerisinde 3 farklı **urunler** listesi yoktur.

Bu **urunler** isimli iki listenin de içerisinde, farklı sayıda **urun** bilgisi bulunuyor. Toplamda 7 adet olan **urun** bilgilerini sorgulayarak doğruluğunu test edelim.

```
SELECT
Anket_UnTyped.exist('(//MagazaAnket/Urunler/Urun)[7]') AS Sales_
UnTyped
FROM Magazalar;
```

Sales_UnTyped	
1	1

Bu sorguda da, kare parantez içerisinde, 7 yerine farklı bir değer kullansaydık, 0 geri dönüşü olacaktı.

XML üzerinde yapılan bu işlemler, XML verisini değişken ortamına alınarak da yapılabilir.

IsTuru bilgisi 'CO' olan **kayıtın** var olup olmadığını sorgulayalım.

```
DECLARE @xml XML;
DECLARE @exist BIT;
SET @xml = (SELECT Anket_UnTyped FROM Magazalar);
SET @exist = @xml.exist('/MagazaAnket[IsTuru="CO"]');
SELECT @exist;
```

(No column name)	
1	1

Sonuç olarak dönen 1 değeri, 'CO' isimli bir iş türünün bulunduğunu gösterir. **IsTuru** bilgisi için farklı bir metinsel değer belirtildiğinde, 0 değeri dönecektir.

exist metodu, bir **WHERE** filtresi ile birlikte de kullanılabilir. Örneğin, **Eğitim** ürünleri içerisinde, iş türü 'CO' olan kaydın var olup olmadığını öğrenelim.

```
SELECT
Anket_UnTyped.query('/MagazaAnket/Urunler[@Tip="Egitim"]') AS Egitim
FROM Magazalar
WHERE Anket_UnTyped.exist('/MagazaAnket[IsTuru="CO"]') = 1;
```

Egitim	
1	<Urunler Tip="Egitim"><Urun>Android</Urun><Urun>Oracle</Urun><Urun>Java</Urun></Urunler>

Burada kullanılan 1 değeri, **exist** metodundan dönmesi beklenen değerın filtrelmesi için kullanılır. **exist** metodundan 0 değeri dönerse, **Eğitim** tipinde de olsa, ürün bilgisi iş türü olarak 'CO' koşulunu karşılamayacağı için sorgu sonuç döndürmeyecektir.

Ancak, **IsTuru** olarak XML de karşılığı olmayan bir değer verilip, eşitlik olarak da 0 değeri kullanılırsa, bu şartlara uyacak kayıtlar getirilecektir. Çünkü XML'de olmayan kayıtlar sorgulanmış olacaktır.

```
SELECT
```

```
Anket_UnTyped.query('/MagazaAnket/Urunler[@Tip="Egitim"]') AS Egitim
FROM Magazalar
WHERE Anket_UnTyped.exist('/MagazaAnket[IsTuru="CE"]') = 0;
```

	Egitim
1	<Urunler Tip="Egitim"><Urun>Android</Urun><Urun>Oracle</Urun><Urun>Java</Urun></Urunler>

XML.VALUE

Belirli bir element ya da attribute için sınırlı değere erişime izin verir. XQuery'nin sonucunu, bir SQL Server veri tipine dönüştürerek skaler bir sonuç döndürür. Ancak, XQuery ifadesi ile verilen düğüm, bir tek skaler elemana dönüştürülebilir bir düğüm olmalıdır.

YillikSatis bilgisinin değerini skaler bir türe dönüştürerek görüntüleyelim.

```
SELECT Anket_UnTyped.value('/MagazaAnket/YillikSatis')[1]', 'INT') AS Satis
FROM Magazalar;
```

	Satis
1	145879

Önceki örnek gibi satış bilgilerini skaler bir türe dönüştürerek görüntüleyelim.

```
SELECT
```

```
Anket_UnTyped.value('/MagazaAnket/YillikSatis')[1]', 'INT') AS Satis,
Anket_Typed.value('declare namespace ns="http://schemas.microsoft.
com/sqlserver/2004/07/adventure-works/StoreSurvey";(/ns:StoreSurvey/
ns:AnnualSales)[1]', 'INT') AS Satis
FROM Magazalar;
```

	Satis	Satis
1	145879	800000

Ürün tipi 2 olan ürün tipini görüntüleyelim.

```
SELECT
```

```
Anket_UnTyped.value('/MagazaAnket/Urunler/@Tip')[2]', 'VARCHAR(10)') AS Tip
FROM Magazalar;
```

	Tip
1	Egitim

Ürün tipi olarak 1 değeri gönderildiğinde ise **Yazılım** görüntülenecektir.

XML veri içerisindeki uzmanlık alanındaki bilgileri görüntüleyelim. Bu işlemi yaparken de, XML'den alınan bir değeri, **CONCAT** fonksiyonunu kullanarak bir metin ile birleştirerek görüntüleyelim.

```
SELECT
Anket_UnTyped.value('concat("Uzmanlık: ", (/MagazaAnket/Uzmanlik)[1])',
'VARCHAR(20)') AS Uzmanlık
FROM Magazalar;
```

	Uzmanlık
1	Uzmanlık: Technology

XML.NODES

XML veriyi, daha fazla ilişkisel biçimde satırlara ayırmayı sağlar. Bir XQuery sorgusu ile yeni bir XML düğümü oluşturur.

Oluşturulan bu XML doğrudan değil, XML metotları ya da **CROSS APPLY** ya da **OUTER APPLY** ile kullanılabilir.

```
DECLARE @veritabanlari XML;
SET @veritabanlari =
'<Urunler>
    <Urun>SQL Server</Urun>
    <Urun>Oracle</Urun>
    <Urun>DB2</Urun>
    <Urun>PostgreSQL</Urun>
    <Urun>MySQL</Urun>
</Urunler>'
SELECT Kategori.query('./text()') AS VeritabaniTuru
FROM @veritabanlari.nodes('/Urunler/Urun') AS Urun(Kategori);
```

	VeritabaniTuru
1	SQL Server
2	Oracle
3	DB2
4	PostgreSQL
5	MySQL

XML içerisindeki **Tip** bilgisine göre bir **CROSS APPLY** sorgusu oluşturarak **Yazılım** tipindeki ürünleri listeleyelim.

```
SELECT Kategori.query('\/text()') AS VeritabaniTuru FROM Magazalar
CROSS APPLY Anket_UnTyped.nodes('\/MagazaAnket/Urunler[@
Tip="Yazılım"]\/Urun') AS Urun(Kategori);
```

	VeritabaniTuru
1	Mobil
2	Masaüstü
3	Sistem
4	Web

XML.MODIFY() İLE XML VERİYİ DÜZENLEMEK

XQuery'e veri değişikliği yapabilme yeteneği kazandırır. XML DML olarak da adlandırılabilir. Bir XML veri üzerinde modify ile düzenleme yapılabilir.

Bir XML değişken tanımlayalım. Bu değişkene değer atayalım ve üzerinde değişiklikler yapalım.

```
DECLARE @xmlAraba XML;
SET @xmlAraba = '
<arabalar>
  <araba></araba>
  <araba></araba>
</arabalar>';
SELECT @xmlAraba;

SET @xmlAraba.modify(
'insert attribute renk{"siyah"}
into /arabalar[1]/araba[1]')
SELECT @xmlAraba;
```

	(No column name)
1	<arabalar><araba /><araba /></arabalar>
	(No column name)
1	<arabalar><araba renk="siyah" /><araba /></arabalar>

İlk olarak araba bilgilerinde herhangi bir veri yoktu. Daha sonra, modify ile **INSERT** işlemi gerçekleştirerek renk adında yeni bir alan açtık ve veri olarak da siyah değerini girdik.

MagazaAnket örneğimize geri dönerek, modify işlemini bu veri üzerinde gerçekleştirelim.

```
UPDATE Magazalar
SET Anket_UnTyped.modify('
    insert(<Aciklama>Yazılım ve eğitim çözümleri</Aciklama>)
    after (/MagazaAnket/CalisanSayisi)[1]'),
    Anket_Typed.modify('declare namespace ns="http://schemas.
microsoft.com/sqlserver/2004/07/adventure-works/StoreSurvey";
    insert(<ns:Comments>Yazılım ve eğitim çözümleri</ns:Comments>)
    after (/ns:StoreSurvey/ns:NumberEmployees)[1]')
WHERE MagazaID = 292;
```

(1 row(s) affected)

Bu sorgu ile birlikte, XML verimiz içerisinde **Aciklama** adında yeni bir alan oluştu. Bu alanın namespace içerisindeki karşılığı ise **Comments** alanıdır.

DELETE

modify metodu kullanılarak XML veri üzerinde **DELETE** işlemi de yapılabilir. Bu işlem için delete ifadesi kullanılır.

Az önce oluşturduğumuz açıklama alanı üzerinde silme işlemi yapalım.

```
UPDATE Magazalar
SET Anket_UnTyped.modify('delete (/MagazaAnket/Aciklama)[1]')
WHERE MagazaID = 292;
```

replace value of ifadesi ile düğüm üzerinde düzenleme yapılabilir. Bu ifade, düğümde güncelleme yapmak için kullanılabilir.

REPLACE VALUE OF

```
UPDATE Magazalar
SET Anket_UnTyped.modify('replace value of (/MagazaAnket/Aciklama/
text())[1] with "2. açıklama"')
WHERE MagazaID = 292;
```

XML BİÇİMİNDEKİ İLİŞKİSEL VERİYE ERİŞMEK

SQL Server 2005 ile birlikte, XML entegrasyonu ve veri erişim mimarisine birçok farklı özellik eklendi. Bu özelliklerle birlikte, XML ile çalışmak daha kolay ve işlevsel hale geldi.

FOR XML

Veritabanı tablolarındaki veriyi XML formatı ile istemcilere ulaştırmaya hazır hale getirir. Yani, FOR XML deyimi, ilişkisel veriler üzerinden XML veri elde etmek için kullanılır.

Söz Dizimi:

```
SELECT sutun_isimleri
FROM tablo_isim
FOR XML { RAW | AUTO | EXPLICIT | PATH }
```

FOR XML, çıktı veriyi belirleyecek farklı modlara sahiptir. Bunlar;

- **RAW**: Sonuç kümesindeki her veri satırını, tek bir veri elemanı olarak geri döndürür. Veri elemanı, satır olarak adlandırılır ve her sütun bir satır elemanının attribute'u olarak listelenir. Satır kontrolü kolay olacağı için iç içe **FOR XML** ifadeleri ile iç içe XML yapıları oluşturulabilir.
- **AUTO**: Her bir elemanı ya tablo ismi ile ya da verinin kaynağı olan tablonun takma adı ile sınıflandırılır. **SELECT** ifadesi sonucunda, tablo yapısını XML'de satır bazında göstererek XML çıktı üretir.
- **EXPLICIT**: XML yapısını etkin olarak biçimlendirmek için kullanılır. **PATH** özelliği geniş oranda bu özelliğin yerini alsa da geçmişe dönük uyumluluk açısından desteklenmektedir.
- **PATH**: **EXPLICIT** özelliğine benzer. XML yapısı üzerinde etkin biçimlendirme için kullanılır. Ancak geliştirmesi ve yönetimi daha kolaydır.

XML biçimlendirme özelliklerine ek olarak, XML sonuçlarını düzenlemek için kullanılabilecek bazı parametreler de vardır.

- **ROOT**: Bu özellik, XML'de bir kök düğüm eklemenizi sağlar. Bu kök düğümü

eklemek zorunlu değildir. Belirttiğiniz kök düğüme isim verebileceğiniz gibi varsayılan olarak `root` ismiyle de kullanabilirsiniz.

- **TYPE:** SQL Server'a, sonuçları varsayılan unicode karakter tipinde değil, XML veri tipinde raporlamasını bildirir.
- **ELEMENTS:** Sonuç verisinde, sütunların attribute'ler halinde değil, iç içe yerleştirilmiş elemanlar olarak döndürülmesini sağlar. `AUTO` ve `RAW` biçimlendirme özellikleriyle birlikte kullanılır.
- **BINARY BASE 64:** `Image`, `Binary`, `VarBinary` gibi sütunların base64 biçiminde kodlanmasını sağlar.
- **XML DATA:** Sonuçlara öncelikle bir XML şema uygulamak için kullanılır. Uygulanan bu schema, veriler için, veri yapısı ve kurallar tanımlayacaktır.

RAW

`ProductID` değerine göre koşul ile sorguladığımız kayıtları XML olarak görüntüleyelim.

```
SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML RAW;
```

Sorgu sonucunda görüntülenen XML dosyası şu şekilde olacaktır.

```
<row ProductID="1" Name="Adjustable Race" ProductNumber="AR-5381" />
<row ProductID="2" Name="Bearing Ball" ProductNumber="BA-8327" />
<row ProductID="3" Name="BB Ball Bearing" ProductNumber="BE-2349" />
<row ProductID="4" Name="Headset Ball Bearings" ProductNumber="BE-2908"/>
```

Bu sorguda satır bazlı XML oluşturulması söz konusudur. Ancak XML yapısına uygun bir model değildir. Daha temiz ve kullanışlı bir XML için aşağıdaki sorgu kullanılabilir.

```
SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML RAW, ELEMENTS;
```

ELEMENTS eklendikten sonra oluşturulan sorgunun sonucu şu şekilde olacaktır.

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

<row> ifadesi hoşunuza gitmediyse bu yapıyı değiştirerek kendi element isminizi belirleyebilirsiniz.

```
SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML RAW('kodlab');
```

Yukarıdaki sorgu sonucunda, daha önce **<row>** ile belirtilen sütunların artık **<kodlab>** olarak belirtildiğini görebilirsiniz.

```
<kodlab ProductID="1" Name="Adjustable Race" ProductNumber="AR-5381" />
<kodlab ProductID="2" Name="Bearing Ball" ProductNumber="BA-8327" />
<kodlab ProductID="3" Name="BB Ball Bearing" ProductNumber="BE-2349" />
<kodlab ProductID="4" Name="Headset Ball" ProductNumber="BE-2908" />
```

Bu XML dokümanına bir root element de eklenebilir.

```
SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML RAW('kodlab'), ROOT('DIJIBIL');
```

XML'e root element eklendikten sonra görünüm şu şekilde olacaktır.

```
<DIJIBIL>
  <kodlab ProductID="1" Name="Adjustable Race" ProductNumber="AR-5381" />
  <kodlab ProductID="2" Name="Bearing Ball" ProductNumber="BA-8327" />
  <kodlab ProductID="3" Name="BB Ball Bearing" ProductNumber="BE-2349" />
  <kodlab ProductID="4" Name="Headset Ball" ProductNumber="BE-2908" />
</DIJIBIL>
```

Tüm veri işlemlerinde olduğu gibi XML yapısında da **NULL** veri bazen sorun çıkarabilir. Yukarıda kullanılan yöntemler ile **NULL** değer içeren bir sütun sorgulandığında o sütuna ait herhangi bir kayıt görüntülenemeyecektir.

Ancak yeni özelliklerden biri olan **XSINIL** ifadesi ile **NULL** değer içeren sütunlar da görüntülenebilir.

```
SELECT ProductID, Name, ProductNumber, Color
FROM Production.Product
WHERE ProductID < 5
FOR XML RAW, ELEMENTS XSINIL;
```

Bu işlem **RAW** ya da **AUTO** ile birlikte, **ELEMENTS** deyiminin sonuna ekleyerek kullanılabilir. **Color** sütunundaki **NULL** kayıtları da görüntüleyecek bu sorgunun sonucu aşağıdaki gibidir.

```
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>1</ProductID>
  <Name>Adjustable Race</Name>
  <ProductNumber>AR-5381</ProductNumber>
  <Color xsi:nil="true" />
</row>
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>2</ProductID>
```

```

    <Name>Bearing Ball</Name>
    <ProductNumber>BA-8327</ProductNumber>
    <Color xsi:nil="true" />
</row>
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ProductID>3</ProductID>
    <Name>BB Ball Bearing</Name>
    <ProductNumber>BE-2349</ProductNumber>
    <Color xsi:nil="true" />
</row>
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <ProductID>4</ProductID>
    <Name>Headset Ball Bearings</Name>
    <ProductNumber>BE-2908</ProductNumber>
    <Color xsi:nil="true" />
</row>

```

AUTO

Tablodaki her elementi bir tablo satırı olarak oluşturan **AUTO** komutu, ana element olarak şema adı ve tablo isminden (Örn; **Production.Product**) oluşan bir isimlendirme kullanır.

En temel **FOR XML AUTO** kullanımı;

```

SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML AUTO;

```

Sorgu sonucu olarak şu şekilde bir XML üretilir.

```

<Production.Product ProductID="1" Name="Adjus.." ProductNumber="AR-5381" />
<Production.Product ProductID="2" Name="Beari.." ProductNumber="BA-8327" />
<Production.Product ProductID="3" Name="BB Ba.." ProductNumber="BE-2349" />
<Production.Product ProductID="4" Name="Heads.." ProductNumber="BE-2908" />

```

** Name sütununun uzun olması nedeniyle içerikler ... ile kısaltılmıştır.*

Oluşturulan bu XML'e bir **ROOT** nod'u da eklenebilir.

```
SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML AUTO, ROOT('DIJIBIL');
```

ROOT nod'u eklendikten sonra üretilen XML çıktı şu şekildedir.

```
<DIJIBIL>
<Production.Product ProductID="1" Name="Adjus.." ProductNumber="AR-5381" />
<Production.Product ProductID="2" Name="Beri.." ProductNumber="BA-8327" />
<Production.Product ProductID="3" Name="BB Ball" ProductNumber="BE-2349" />
<Production.Product ProductID="4" Name="Headset" ProductNumber="BE-2908" />
</DIJIBIL>
```

Aynı verileri attribute değil de element olarak görüntülemek istersek, **ELEMENTS** ifadesini eklememiz yeterli olacaktır.

```
SELECT ProductID, Name, ProductNumber
FROM Production.Product
WHERE ProductID < 5
FOR XML AUTO, ELEMENTS, ROOT('DIJIBIL');
```

Sorgu sonucunda şu şekilde bir XML üretilcektir.

```
<DIJIBIL>
  <Production.Product>
    <ProductID>1</ProductID>
    <Name>Adjustable Race</Name>
    <ProductNumber>AR-5381</ProductNumber>
  </Production.Product>
  <Production.Product>
    <ProductID>2</ProductID>
    <Name>Bearing Ball</Name>
    <ProductNumber>BA-8327</ProductNumber>
  </Production.Product>
  <Production.Product>
    <ProductID>3</ProductID>
    <Name>BB Ball Bearing</Name>
```

```

        <ProductNumber>BE-2349</ProductNumber>
    </Production.Product>
</Production.Product>
    <ProductID>4</ProductID>
    <Name>Headset Ball Bearings</Name>
    <ProductNumber>BE-2908</ProductNumber>
</Production.Product>
</DIJIBIL>

```

EXPLICIT

XML'in esnek kullanımı için gerekli olan bazı durumlar olabilir. Bu tür durumlarda farklı XML formatları elde etmek için kullanılır.

EXPLICIT ile XML veri oluşturabilmek için sorgu içerisinde iki temel meta sütun bulundurulmak zorundadır.

Bunlar;

- **TAG**: İlk sütun Integer tipinde bir **TAG** numarası almak ve sütun adı da **TAG** olmak zorundadır.
- **PARENT**: İkinci sütunun adı **PARENT** olmak zorundadır.

TAG ve **PARENT** sütunları veriler üzerinde hiyerarşi belirlemek için kullanılır.

Ürünler tablosu üzerinde hazırlanacak bir XML için format belirleyelim.

```

SELECT
    TOP 5
    1 AS TAG,
    NULL AS PARENT,
    ProductID AS [Product!1!ProductID],
    Name AS [Product!1!Name!element],
    ProductNumber AS [Product!1!ProductNumber!element],
    ListPrice AS [Product!1!ListPrice!element]
FROM Production.Product
ORDER BY ProductID
FOR XML EXPLICIT;

```

Sorgu sonucunun XML görüntüsü:

```
<Product ProductID="1">
  <Name>Adjustable Race</Name>
  <ProductNumber>AR-5381</ProductNumber>
  <ListPrice>0.0000</ListPrice>
</Product>
<Product ProductID="2">
  <Name>Bearing Ball</Name>
  <ProductNumber>BA-8327</ProductNumber>
  <ListPrice>0.0000</ListPrice>
</Product>
<Product ProductID="3">
  <Name>BB Ball Bearing</Name>
  <ProductNumber>BE-2349</ProductNumber>
  <ListPrice>0.0000</ListPrice>
</Product>
```

Sorgunun XML çıktısında **ProductID** bir attribute iken, **Name**, **ProductNumber** ve **ListPrice** sütunları bir element olarak görüntülendi.

Bunun nedeni; **ProductID** sütunu için formatlama yaparken element komutunu eklememiş olmamızdır.

```
ProductID AS [Product!1!ProductID]
```

Element komutu eklenen diğer 3 sütun ise, element formatında görüntülenmiştir.

```
ProductNumber AS [Product!1!ProductNumber!element]
```

Aynı sorguda **ProductID** sütunu için de element tanımlaması yaptıktan sonra, artık XML formatındaki dokümanda **ProductID** sütunu da element formatında görüntülenir.

```
SELECT
  TOP 3
  1 AS TAG,
  NULL AS PARENT,
  ProductID AS [Product!1!ProductID!element],
```

```
Name AS [Product!!Name!element],
ProductNumber AS [Product!!ProductNumber!element],
ListPrice AS [Product!!ListPrice!element]
FROM Production.Product
ORDER BY ProductID
FOR XML EXPLICIT;
```

Sorgunun XML çıktısı:

```
<Product>
  <ProductID>1</ProductID>
  <Name>Adjustable Race</Name>
  <ProductNumber>AR-5381</ProductNumber>
  <ListPrice>0.0000</ListPrice>
</Product>
<Product>
  <ProductID>2</ProductID>
  <Name>Bearing Ball</Name>
  <ProductNumber>BA-8327</ProductNumber>
  <ListPrice>0.0000</ListPrice>
</Product>
<Product>
  <ProductID>3</ProductID>
  <Name>BB Ball Bearing</Name>
  <ProductNumber>BE-2349</ProductNumber>
  <ListPrice>0.0000</ListPrice>
</Product>
```

NULL değer içeren sütunlar üzerinde de **EXPLICIT** kullanılabilir.

ProductID sütununu **DESCENDING** olarak sıralayarak XML çıktısı oluşturalım.

```
SELECT
TOP 3
1 AS TAG,
NULL AS PARENT,
ProductID AS [Product!!ProductID!element],
Name AS [Product!!Name!element],
ProductNumber AS [Product!!ProductNumber!element],
ListPrice AS [Product!!ListPrice!element],
```

```

    Color AS [Product!Color!elementxsinil]
FROM Production.Product
ORDER BY ProductID DESC
FOR XML EXPLICIT;

```

Sorgunun XML çıktısı:

```

<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>1004</ProductID>
  <Name>% 20 indirimli ürün</Name>
  <ProductNumber>SK-9299</ProductNumber>
  <ListPrice>0.0000</ListPrice>
  <Color xsi:nil="true" />
</Product>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>999</ProductID>
  <Name>Road-750 Black, 52</Name>
  <ProductNumber>BK-R19B-52</ProductNumber>
  <ListPrice>619.5637</ListPrice>
  <Color>Black</Color>
</Product>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>998</ProductID>
  <Name>Road-750 Black, 48</Name>
  <ProductNumber>BK-R19B-48</ProductNumber>
  <ListPrice>619.5637</ListPrice>
  <Color>Black</Color>
</Product>

```

Sorgu sonucunda dönen 3 kayıttan ilkinde, **Color** sütunu **NULL** olduğu için şu şekilde belirtildi.

```
<Color xsi:nil="true" />
```

Ancak diğer kayıtlarda bir değer bulunduğu için renk isimleri metinsel olarak döndürüldü.

EXPLICIT İLE SÜTUNLARI GİZLEMEK

Bazı durumlarda kayıtları XML'e dönüştürürken bazı sütunların gizlenmesi istenebilir. Bu gibi durumlarda HIDE direktifi kullanılır.

Daha önce kullandığımız sorguda ürün fiyatlarını gizleyelim.

```
SELECT
TOP 3
1 AS TAG,
NULL AS PARENT,
ProductID AS [Product!1!ProductID!element],
Name AS [Product!1!Name!element],
ProductNumber AS [Product!1!ProductNumber!element],
ListPrice AS [Product!1!ListPrice!hide],
Color AS [Product!1!Color!elementxsinil]
FROM Production.Product
ORDER BY ProductID DESC
FOR XML EXPLICIT
```

Sorgu içerisinde **ListPrice** sütunu olmasına rağmen, XML çıktısında bu sütun yer almayacaktır.

```
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>1004</ProductID>
  <Name>% 20 indirimli ürün</Name>
  <ProductNumber>SK-9299</ProductNumber>
  <Color xsi:nil="true" />
</Product>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>999</ProductID>
  <Name>Road-750 Black, 52</Name>
  <ProductNumber>BK-R19B-52</ProductNumber>
  <Color>Black</Color>
</Product>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ProductID>998</ProductID>
  <Name>Road-750 Black, 48</Name>
  <ProductNumber>BK-R19B-48</ProductNumber>
  <Color>Black</Color>
</Product>
```

EXPLICIT ile kullanılabilecek diğer direktifler:

- ID, IDREF ve IDREFS
- CDATA
- XML
- XMLTEXT

PATH

FOR XML PATH ifadesi yetenekli ve sade bir XML çıktısı üretme komutudur.

Ana kategori ve alt kategoriler üzerinde bir **JOIN** işlemi gerçekleştirerek **FOR XML PATH** ifadesini kullanalım.

```
SELECT pc.ProductCategoryID, pc.Name, psc.ProductSubcategoryID, psc.Name
FROM Production.ProductCategory AS pc
INNER JOIN Production.ProductSubcategory AS psc
ON pc.ProductCategoryID = psc.ProductCategoryID
FOR XML PATH;
```

Sorgu sonucunda temiz ve anlaşılır bir XML çıktısı üretilecektir. Üretilen XML içerisindeki ilk üç kayıt;

```
<row>
  <ProductCategoryID>1</ProductCategoryID>
  <Name>Bikes</Name>
  <ProductSubcategoryID>1</ProductSubcategoryID>
  <Name>Mountain Bikes</Name>
</row>
<row>
  <ProductCategoryID>1</ProductCategoryID>
  <Name>Bikes</Name>
  <ProductSubcategoryID>2</ProductSubcategoryID>
  <Name>Road Bikes</Name>
</row>
<row>
  <ProductCategoryID>1</ProductCategoryID>
  <Name>Bikes</Name>
  <ProductSubcategoryID>3</ProductSubcategoryID>
  <Name>Touring Bikes</Name>
</row>
```

Bu XML yapısına element isimleri verilerek daha anlaşılır hale getirilebilir.

```
SELECT pc.ProductCategoryID, pc.Name,
       psc.ProductSubcategoryID, psc.Name
FROM Production.ProductCategory AS pc
INNER JOIN Production.ProductSubcategory AS psc
ON pc.ProductCategoryID = psc.ProductCategoryID
FOR XML PATH('Kategori'), ROOT('Kategoriler');
```

Bazı bilgiler element olarak değil, bir elemente öznitelik olarak eklenmek istenebilir. Hazırladığımız örnekte **ProductCategoryID** değerini **Kategori** elementine öznitelik olarak ekleyelim.

```
SELECT pc.ProductCategoryID "@KategoriID", pc.Name,
       psc.ProductSubcategoryID, psc.Name
FROM Production.ProductCategory AS pc
INNER JOIN Production.ProductSubcategory AS psc
ON pc.ProductCategoryID = psc.ProductCategoryID
FOR XML PATH('Kategori'), ROOT('Kategoriler');
```

Bu sorgunun XML çıktısı şu şekilde olacaktır.

```
<Kategoriler>
  <Kategori KategoriID="1">
    <Name>Bikes</Name>
    <ProductSubcategoryID>1</ProductSubcategoryID>
    <Name>Mountain Bikes</Name>
  </Kategori>
  <Kategori KategoriID="1">
    <Name>Bikes</Name>
    <ProductSubcategoryID>2</ProductSubcategoryID>
    <Name>Road Bikes</Name>
  </Kategori>
  <Kategori KategoriID="1">
    <Name>Bikes</Name>
    <ProductSubcategoryID>3</ProductSubcategoryID>
    <Name>Touring Bikes</Name>
  </Kategori>
  <Kategori KategoriID="2">
    <Name>Components</Name>
```



```

        <ProductSubcategoryID>4</ProductSubcategoryID>
        <Name>Handlebars</Name>
    </Kategori>
</Kategoriler>

```

** Sorgu sonucunun ilk dört kayıdır.*

OPEN XML

Dışarıdan alınan veriyi ilişkisel veritabanı ortamında saklamayı sağlar.

Bu işlemi sistem prosedürleri ile gerçekleştirir. Bunlar;

- `sp_xml_preparedocument`
- `sp_xml_removedocument`

İşlem adımları;

- `sp_xml_preparedocument` ile XML veri parse edilir.
- DOM'a yerleşen veri **OPENXML** ile parçalanır ve ilişkisel ortama aktarılır.
- XML işlemleri için kullanılan hafızanın boşaltılmasını sağlar. Hafıza SQL Server tarafından scope bitiminde yapılır. Ancak hafıza yönetimini SQL Server'a bırakmadan yapabilmek gerekir. Bunun için `sp_xml_removedocument` kullanılır.

Books isminde bir tablo oluşturalım ve bir XML veriyi, **OpenXML** ile bu tabloya ekleyelim.

```

CREATE TABLE Books
(
    BookID INT,
    Name VARCHAR(50),
    Author VARCHAR(50),
    Technology VARCHAR(30)
);

```

XML veriyi tabloya ekleyelim.

```

DECLARE @xml_data INT;
DECLARE @xmldoc VARCHAR(1000);

SET @xmldoc =

```

```
'<root>
  <book BookID="1" Name="İleri Seviye SQL Server T-SQL"
    Technology="SQL Server" Author="Cihan Ozhan" />
  <book BookID="2" Name="İleri Seviye Android Programlama"
    Technology="Android" Author="Kerim Fırat" />
</root>'
```

```
EXEC sp_xml_preparedocument @xml_data OUTPUT, @xmldoc;
```

```
INSERT INTO Books
```

```
SELECT * FROM OpenXML(@xml_data, '/root/book') WITH Books;
```

```
EXEC sp_xml_removedocument @xml_data;
```

XML veriyi, **Books** tablosuna başarıyla ekledik. Tablodaki veriyi listeleterek görelim.

```
SELECT * FROM Books;
```

	BookID	Name	Author	Technology
1	1	İleri Seviye SQL Server T-SQL	Cihan Ozhan	SQL Server
2	2	İleri Seviye Android Programlama	Kerim Fırat	Android

Şimdi de, tabloya veri eklemekten, sadece **SELECT** ile bir XML veriyi okuma işlemi gerçekleştirelim.

```
DECLARE @DocHandle INT;
```

```
DECLARE @XmlDocument NVARCHAR(1000);
```

```
SET @XmlDocument =
```

```
N'<root>
```

```
<Musteri MusteriID="VINET" İletisimAd="Hakan Aydın">
```

```
  <Urun UrunID="10248" MusID="VINET" CalID="8" SipTarih="2013-02-02">
```

```
    <UrunDetay UrunID="19" Miktar="7"/>
```

```
    <UrunDetay UrunID="25" Miktar="5"/>
```

```
  </Urun>
```

```
</Musteri>
```

```
<Musteri MusteriID="LILAS" İletisimAd="Cansu Aycan">
```

```
<Urun UrunID="10283" MusID="LILAS" CalID="6" SipTarih="2013-02-13">
```

```
  <UrunDetay UrunID="13" Miktar="2"/>
```

```
</Urun>
```

```
</Musteri>
```

```
</root>';
```

```
EXEC sp_xml_preparedocument @DocHandle OUTPUT, @XmlDocument;

SELECT *
FROM OPENXML(@DocHandle, '/root/Musteri',1)
    WITH (MusteriID varchar(10),IletisimAd varchar(20));
EXEC sp_xml_removedocument @DocHandle;
```

	MusteriID	IletisimAd
1	VINET	Hakan Aydın
2	LILAS	Cansu Aycan

AdventureWorks veritabanı modelinde olan bir XML'i tüm sütunlarıyla birlikte görüntüleyelim.

```
DECLARE @XmlDokumanIslem INT;
DECLARE @XmlDokuman NVARCHAR(1000);
SET @XmlDokuman = N'
<root>
<Musteri MusteriID="VINET" Iletisim="Hakan Aydın">
    <Siparis SiparisID="10248" MusteriID="VINET" CalisanID="5"
        SiparisTarih="2013-02-02">
        <SiparisDetay UrunID="11" Miktar="12"/>
        <SiparisDetay UrunID="42" Miktar="10"/>
    </Siparis>
</Musteri>
<Musteri MusteriID="LILAS" Iletisim="Cansu Aycan">
    <Siparis SiparisID="10283" MusteriID="LILAS" CalisanID="3"
        SiparisTarih="2013-02-04">
        <SiparisDetay UrunID="72" Miktar="3"/>
    </Siparis>
</Musteri>
</root>';

EXEC sp_xml_preparedocument @XmlDokumanIslem OUTPUT, @XmlDokuman;

SELECT *
FROM OPENXML(@XmlDokumanIslem, '/root/Musteri/Siparis/SiparisDetay', 2)
WITH
(
    SiparisID INT        '../@SiparisID',
    MusteriID VARCHAR(10) '../@MusteriID',
    SiparisTarih DATETIME '../@SiparisTarih',
```

```

        UrunID          INT          '@UrunID',
        Miktar          INT          '@Miktar'
    );
EXEC sp_xml_removedocument @XmlDokumanIslem;

```

	SiparisID	MusteriID	SiparisTarih	UrunID	Miktar
1	10248	VINET	2013-02-02 00:00:00.000	11	12
2	10248	VINET	2013-02-02 00:00:00.000	42	10
3	10283	LILAS	2013-02-04 00:00:00.000	72	3

SELECT işleminden önce, XML yapı değişkene **SET** edilir. Bu XML, **sp_xml_preparedocument** sistem prosedürü ile parse edildikten sonra, **SELECT** sorgusu çalıştırılır. Son olarak, XML'in bulunduğu belleğin boşaltılması için **sp_xml_removedocument** sistem prosedürü çalıştırılır ve bellek boşaltılır.

Sorguda kullanılan XML verinin yapısı farklı formatlarda görünecek şekilde değiştirilebilir. XML verinin tamamı okunsa da, **SELECT** sorgusu içerisinde, sadece görünmesi istenen sütunlar seçilerek, belirli sütunların görüntülenmesi sağlanabilir.

HTTP ENDPOINT'LERİ

Web üzerinden istemcilere ulaşmak, uzaktaki servislere erişmek bilişim dünyasının farklı çözümler ürettiği sorunlardan biridir. Web üzerinden bir servislere erişmek istendiğinde güvenlik nedeniyle Firewall gibi bazı sorunlarla karşılaşılır.

Servis mimarili teknolojilerde bu iletişim ilk olarak port açma yöntemi ile gerçekleştirilirdi. Açılan port ile web ve servis arasında binary veri alış verişi gerçekleştirilir ve teorik anlamda çözüm sağlanmış olurdu. Ancak bir sistem yöneticisi açısından bu durum asla böyle olmamalıdır. Çünkü açılan her port aslında bir güvenlik riskidir. İş tecrübeleriniz arasında profesyonel bir IT departmanı ile birlikte çalışma tecrübesi varsa, bu tür port açma isteklerine IT departmanının vereceği olumsuz yanıtı yaşamış olmalısınız.

Port üzerinden binary alış verişi işleminin güvenlik sorunları oluşturması nedeniyle SOAP geliştirildi. **SOAP (Simple Object Access Protocol)**, HTTP üzerinden istekte bulunduğu için ekstra bir port açma işleminin önüne geçmiştir. HTTP portu her sistemde aynıdır ve 80. portu kullanır.

Yani zaten erişime açık olan bir port üzerinden servis iletişimi sağlanmaktadır. SOAP paketi binary değil, metin demeti içerir.

HTTP Endpoint'ler veritabanına değil, sunucu üzerine kurularak birçok özelliğe sahip olduğu için güvenlik açısından risklidir. Yani, Endpoint kurulu sunucular güvenlik riski altındadır.

HTTP ENDPOINT VE GÜVENLİK

HTTP Endpoint kullanımının bazı güvenlik riskleri taşıdığını belirttik. Bu riskleri en aza indirmek ve önlemler almak için bazı özelliklere sahiptir.

- **Connect:** Kullanıcının endpoint üzerindeki veriyi görmesine izin verir. Bir kullanıcıya, endpoint üzerindeki veriyi görme izni vermeden, endpoint'i yönetme izni verilebilir.
- **View Definition:** Kullanıcının bir endpoint ile ilgili metadata bilgisini görmesine izin verir.
- **Control:** Kullanıcının belirli bir endpoint'i değiştirmesi ya da kaldırmasına izin verir.

HTTP ENDPOINT İLE KULLANILACAK VERİ NESNELERİNİN OLUŞTURULMASI

Endpoint ile Stored Procedure'leri kullanarak bir servis oluşturacağız. Bu servisi oluşturmadan önce iki tane Stored Procedure oluşturacağız.

Tüm ürünleri listeleyen sproc;

```
CREATE PROC sp_Products
AS
SELECT ProductID, Name FROM Production.Product;
```

Tüm personelleri listeleyen sproc;

```
CREATE PROC sp_Persons
AS
SELECT BusinessEntityID, PersonType, FirstName, LastName
FROM Person.Person;
```

Oluşturulan Stored Procedure'ler Endpoint ile servis olarak kullanılabilir hale getirilecektir.

HTTP ENDPOINT OLUŞTURULMASI VE YÖNETİLMESİ

Endpoint'ler için oluşturulan Stored Procedure'leri kullanarak bir HTTP Endpoint oluşturalım.

```
CREATE ENDPOINT EP_AdventureWorks
STATE = STARTED
AS HTTP
(
    PATH = '/AWorks',
    AUTHENTICATION = (INTEGRATED),
    PORTS = (CLEAR),
    SITE = 'localhost'
)
FOR SOAP
(
    WEBMETHOD 'Products'
    (NAME = 'AdventureWorks2012.dbo.sp_Products'),
    WEBMETHOD 'Persons'
    (NAME = 'AdventureWorks2012.dbo.sp_Persons'),
    BATCHES = DISABLED,
    WSDL = DEFAULT,
    DATABASE = 'AdventureWorks2012',
    NAMESPACE = 'http://AdventureWorks/'
)

```

HTTP Endpoint'lerin oluşturulabilmesi için **CREATE ENDPOINT** nesne oluşturma komutunun çalışıyor olması gerekir. Ancak SQL Server Express Edition sürümünde bu komut çalışmaz. Lisanslı sürümlerde bu tür sorunlar yaşamazsınız.

SQL Server Express Edition ile alacağınız hata bildirimi;

This "CREATE ENDPOINT" statement is not supported on this edition of SQL Server.

Eğer sorgunuz başarılı bir şekilde çalıştı ise artık web servisiniz oluşturulmuş demektir. Daha önce web servis geliştirme ya da kullanma tecrübesine sahipseniz aşağıdaki bağlantı ile ilgili stored procedure'lerin dışarıya açılmış servis modellerini inceleyebilirsiniz.

`http://localhost/AWorks?wsdl`

Yukarıdaki servis ile iki prosedüre de ulaşabilirsiniz. Servis isimlerinden herhangi birini çağırdığınızda, SQL Server'da tanımlı ve çağırdığınız servis için oluşturulan Stored Procedure çalışacak ve sonucu döndürecektir.

