VERİTABANI PROGRAMLAMA

3

Amaçlarımız

Bu üniteyi tamamladıktan sonra;

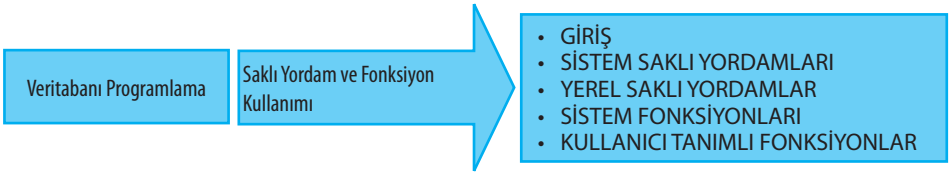
- Saklı yordamların ve fonksiyonların kullanımlarını açıklayabilecek,
- T-SQL'de uygulamaya yönelik yerel saklı yordam hazırlayabilecek veya kullanabilecek,
- T-SQL'de uygulamaya yönelik kullanıcı tanımlı fonksiyon hazırlayabilecek veya kullanabilecek

bilgi ve becerilere sahip olabileceksiniz.

Anahtar Kavramlar

- Yerel Saklı Yordam
- Sistem Fonksiyonu
- Kullanıcı Tanımlı Fonksiyonlar

İçindekiler



Saklı Yordam ve Fonksiyon Kullanımı

GİRİŞ

Saklı yordam; belirli bir görevi yerine getirmek için tasarlanmış, sunucu üzerinde tutulan, birden fazla tablo üzerinde işlem yapabilen, program içinden farklı parametreler ile çağrılarak kullanılabilen SQL tabanlı komut kümesidir. Önceki bölümde anlatılan tüm T-SQL komutlarını saklı yordam içinde programlama amaçlı kullanmak mümkündür.

Veritabanında yer alan veriler ile yapılan seçim, ekleme, güncelleme, silme ve benzeri işlemler için SQL tabanlı komutlar her defasında öncelikle yazım kuralları açısından denetlenir, sonra ilgili tablo ve alanların olup olmadığının kontrolü yapılır. En kısa ve en hızlı sorgunun nasıl yapılacağı belirlendikten sonra da ilgili sorgu derlenerek komut çalıştırılır. Saklı yordamlar sayesinde, sorgunun derlenmesine kadar olan kısımlar bir kez yapılır ve saklı yordamın her çağrılışında sadece ilgili sorgu çalıştırılır. Daha sonraki çalıştırma işlemlerinde derlenmediklerinden saklı yordamlar oldukça hızlı çalışırlar.

Saklı yordamlar programlama dillerindeki gibi parametre içerirler. Bu parametrelere göre çalışıp farklı sonuçlar listeleyebilir. Dışarıdan değer alıp geriye değer döndürebilirler. Bir saklı yordam diğer saklı yordamlar tarafından çağırılıp kullanabilir. Saklı yordamların giriş çıkış parametrelerinde değişiklik olmadığı sürece, kullanıcı arayüzünde program değişikliği yapmadan saklı yordamların kodlanmasında değişiklik yapılabilir. Bu ise veritabanı programlamayı kullanıcı arayüzü programlamadan bağımsız hâle getirir. Modüler yazılım mimarisi oluşturmaya da önemli bir katkı sağlar. Bu sayede farklı kullanıcıların da aynı modülü kullanma imkânı olur.

Kullanıcıların **saklı yordam**a erişimi yetki iznine bağlanabilir. İzne sahip değilse doğrudan saklı yordamı çalıştıramazlar. Veritabanı yöneticileri, kullanıcılarına saklı yordam bazında kullanıcı hakları tanımlayabildiğinden aynı zamanda güvenliğe katkı sağlarlar. Diğer bir yönüyle istemci tarafından birçok satıra sahip SQL komutunun sunucuya gitmesindense, sadece saklı yordamın adının sunucuya gitmesi, ağı daha az meşgul eder. Örneğin, bir sorgu içinde yüzlerce satır varsa ve aynı sorguyu birçok kullanıcı kullanıyorsa ağın trafik yoğunluğu artar. Bu tarz sorguyu saklı yordam olarak tutmak hem ağ yoğunluğunu azaltır hem de yukarıdaki avantajları sağlar. Saklı yordamlar; *sistem saklı yordamları* (system stored procedures), *genişletilmiş saklı yordamlar* (extended stored procedures) ve *yerel saklı yordamlar* (local stored procedures) olmak üzere üçe ayrılmaktadır.

Saklı Yordamlar (Stored Procedures) veritabanlarında tekrarlı işlemler için oluşturulan komut kümeleridir.

Genişletilmiş saklı yordamlar Microsoft SQL Sunucunun 2016 sonrası sürümlerde desteklenmeyecek olup yeni geliştirmelerde kullanılmaması önerilir. Geliştirmelerde alternatif olarak Common Language Runtime (CLR) entegrasyonunun kullanılması önerilir.

VTYS yazılımlarında **fonksiyon**lar (functions) kullanıcı tanımlı ve sistem fonksiyonlu olmak üzere iki türdür.

Veritabanı programlamada önemli bir esneklik sağlayan yapılardan birisi de **fonksiyon**'dur. Fonksiyonlar da kullanıcı tarafından tanımlanabilen fonksiyonlar ve sistem fonksiyonları olmak üzere ikiye ayrılmaktadır. İlk değerlendirmede kullanıcı fonksiyonlarının genel yapısı saklı yordamlara benzemekle beraber aralarında aşağıdaki farklar vardır:

- Kullanıcı tanımlı fonksiyonların içerisinde herhangi bir yerde `WHERE/HAVING/SELECT` vb. kullanılabilirken saklı yordam kullanılamaz. Ayrıca saklı yordamda fonksiyonlar çağrılabilirken, fonksiyon içinde saklı yordam çağrılamaz.
- Kullanıcı tanımlı fonksiyon mutlaka bir tablo veya sayılı değer döndürmelidir. Döndürdüğü tablo değeri, diğer tablolar ile `JOIN` vb. işlemler yapmak için kullanılabilir. Saklı yordamların mutlaka değer döndürmeleri gerekmez.
- Saklı yordamlarda veri işleme (`INSERT`/`UPDATE`/`DELETE`) komutları çalıştırılabilirken kullanıcı fonksiyonlarında bu işlemler yapılamaz.
- Saklı yordam içinde bölüm ikide anlatılan `TRY CATCH` yapısı ile hata ayıklama opsiyonu varken, fonksiyonlar içinde bu opsiyon yoktur.
- Saklı yordam içinde ileriki bölümlerde anlatılan hareket yönetimi varken, fonksiyon içinde bu mümkün değildir.
- Saklı yordamlar derlenmiş olarak veritabanında tutulurken, fonksiyonlar çalışma zamanında derlenir ve çalıştırılır.

Takip eden alt bölümlerde farklı özelliklerdeki saklı yordamlar ve fonksiyonları sırası ile verilmektedir.

SİSTEM SAKLI YORDAMLARI

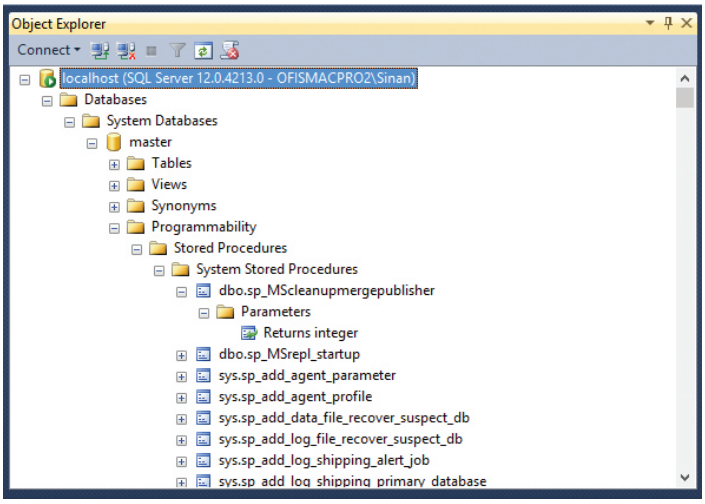
SQL Server sunucusunda ön tanımlı olarak bulunan ve "sp_" ön eki ile adlandırılmış saklı yordamlar sistem saklı yordamları olarak adlandırılır. Herhangi bir veritabanı üzerinde sistemle ilgili işlem yapmak veya bilgi alma için kullanılırlar. Tablo 3.1'de sistem saklı yordamlarının kullanım amacına göre bazı sınıflandırılmaları vardır.

Tablo 3.1 *Sistem Saklı Yordamlarının Bazı Genel Sınıflandırılmaları (Internet: https://msdn.microsoft.com/en-us/library/ms187961.aspx)*

Sınıflandırma	Tanım
Katalog Saklı Yordamları (Catalog Stored Procedures)	Açık Veritabanı Bağlantısı (ODBC) uygulamalarını sistem tablolarındaki değişimlerinden izole etmek için kullanılmaktadır.
Değişen Veri Yakalama Saklı Yordamları (Change Data Capture Stored Procedures)	Değişen veri yakalama nesnelerini aktif-pasif hâle getirme veya raporlama için kullanılmaktadır.
İmleç Saklı Yordamları (Cursor Stored Procedures)	İmleç uygulamaları için kullanılmaktadır.
Veritabanı Motoru Saklı Yordamları (Database Engine Stored Procedures)	SQL sunucu veritabanı motorunun bakımı için kullanılmaktadır.
Veritabanı E-mail Saklı Yordamı (Database Mail Stored Procedures)(T-SQL)	SQL sunucu oluşumlarından e-mail atmak için kullanılmaktadır.
XML Saklı Yordamları (XML Stored Procedures)	XML dosya yönetimi için kullanılmaktadır.

Sistem saklı yordamlarının tüm listesine Nesne tarayıcı (Object Explorer) penceresinde Resim 3.1'de de görüldüğü gibi "Databases → System Databases → master → Programmability → Stored Procedures → System Stored Procedures" hiyerarşisi takip edilerek ulaşılabilir.

Resim 3.1



Sistem saklı yordamlarına erişim

VTYS ile ilgili değişik bilgiler aşağıdaki örneklerde olduğu gibi alınabilir. SQL içinde saklı yordamlar EXEC komutu ile çalıştırılır.

Örnek Uygulama 3.1: Sunucu üzerinde bulunan veritabanlarını ve boyutlarını öğrenmek için "sp_databases" sistem saklı yordamı kullanılabilir. Aşağıda SQL Managment Studio'da çalıştırılan saklı yordam ve sonucu bulunmaktadır.

```
EXEC sp_databases
```

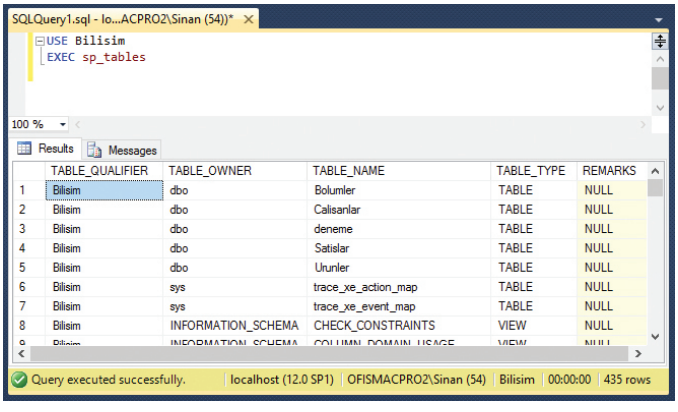
Sonuç:

DATABASE_NAME	DATABASE_SIZE	REMARKS
Bilisim	7168	NULL
Master	7552	NULL
Model	5312	NULL
Msdb	20608	NULL

İNTERNET

Bu ünitedeki uygulama komut dosyalarına https://goo.gl/IZUEzf adresinden ulaşabilirsiniz. Ayrıca ilk ünitedeki komut satırı dosyası da çalıştırılarak örnek veritabanının oluşturulması gerektiğini unutmayınız.

Örnek Uygulama 3.2. Örnek veritabanındaki aktif tablo ve görünümleri listelemek için "sp_tables" sistem saklı yordamı kullanılabilir. Aşağıda SQL Managment Studio'da çalıştırılan saklı yordam ve sonucu bulunmaktadır.



SIRA SİZDE 1

Örnek veritabanındaki Urunler tablosunun yapısı hakkında ayrıntılı bilgi almak isteyelim. Bunun için hangi saklı yordamı kullanırsınız?

YEREL SAKLI YORDAMLAR

Yerel saklı yordamlar, kullanıcı tarafından oluşturulan saklı yordamlar olduğundan kullanıcı tabanlı saklı yordamlar olarak da adlandırılır. Giriş bölümünde belirtildiği gibi saklı yordamlar ile farklı SQL yığınları hızlı bir biçimde kullanıcıların ortak kullanımına açılmış olur. Saklı yordamlar, veritabanı programlamada performansı yüksek işler yapmak için kullanılan kritik araçlardan birisidir. Saklı yordamları isimlendirirken "sp_" ön ekini kullanmak okunabilirliğini arttırır. Yerel saklı yordamların parametreli ve parametresiz genel yazım şekli aşağıdaki gibidir.

Parametresiz Kullanım	Parametreli Kullanım

`CREATE PROCEDURE sp_SaklıYordamIsmi`
`AS`
`BEGIN`
`  --Komutlar`
`END`
	`CREATE PROCEDURE sp_SaklıYordamIsmi`
	`  @Param1 veri_türü ,`
	`  @Param2 veri_türü`
	`AS`
	`BEGIN`
	`  --Komutlar`
	`END`

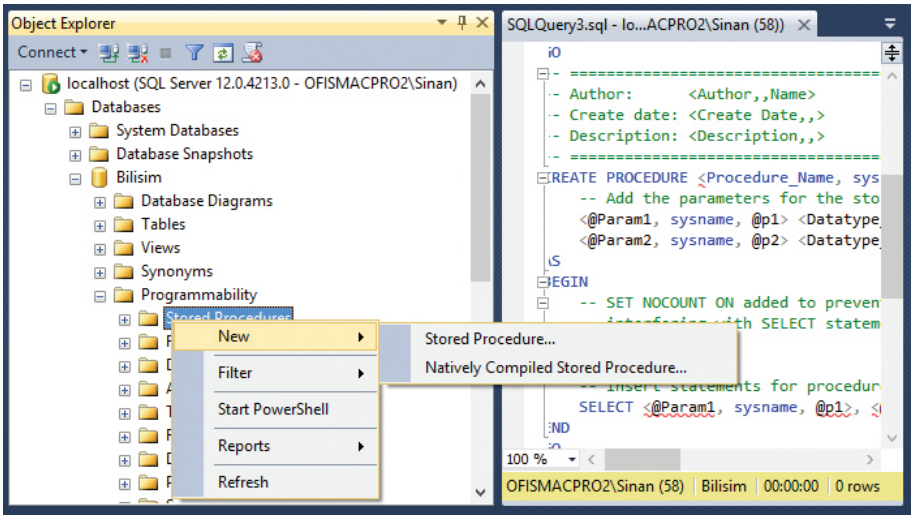
Yerel Saklı Yordamların Yardımcı ile Oluşturulması

SQL Server Management Studio'da kullanıcıların sql sorgu ve nesne oluşturmalarına yönelik yardımcılar kullanılarak hızlı bir şekilde yazılabilmektedir. Bunun için Object Explorer üzerinde ilgili nesne üzerinde sağ fare tuşuna tıklanarak şablon kodlara ulaşılabilir. Örneğin bir saklı yordam yazmak için Database → Bilişim → Programmability → Stored Procedure kısmına gidilerek Stored Procedure ifadesi üzerinde sağ fare tuşuna tıklanarak New Stored Procedure seçeneğine basıldığında şablon bir stored procedure kodu yeni bir sorgu ekranında açılacaktır.

Örnek veritabanı "Bilişim" için kimin hangi bölümde çalıştığını listeleyen bir yerel saklı yordam oluşturulmak istensin. Bunun için kitabın ilk bölümünde oluşturduğumuz veritabanı "Bilişim " veritabanına ulaşmak için SQL Server Management Studio programını açarak veritabanına bağlanmak gerekiyor. Sonrasında Resim 3.2'de görüldüğü gibi yeni bir saklı yordam (stored procedure) oluşturulur.

Resim 3.2

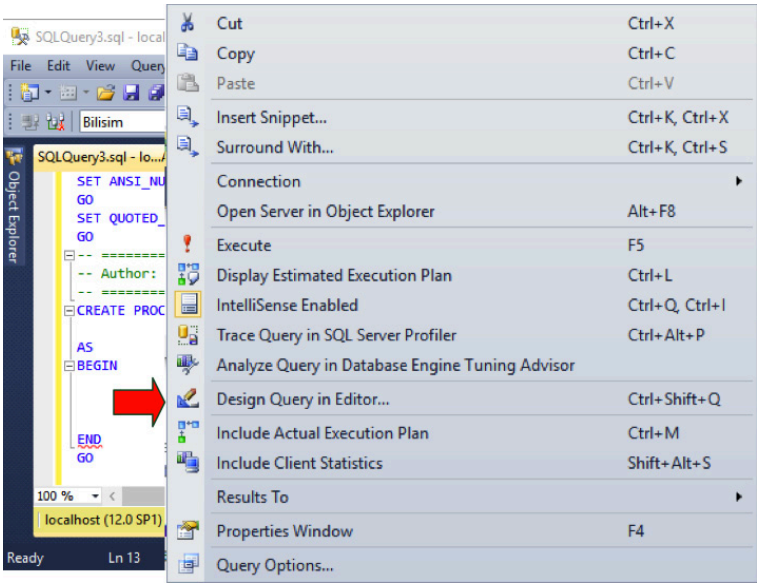
SQL Server Management Studio ortamında yerel saklı yordam oluşturma



Kullandığınız SQL Server yazılımının sürümüne ve lisansına bağlı olarak SQL Server Management Studio'da görüntülenen menüler ve seçenekler değişebilir. Ancak bu değişiklikler burada tanımlanan örnek işlemlerde sıkıntı yaratacak değişiklikler olmayacaktır.

DİKKAT

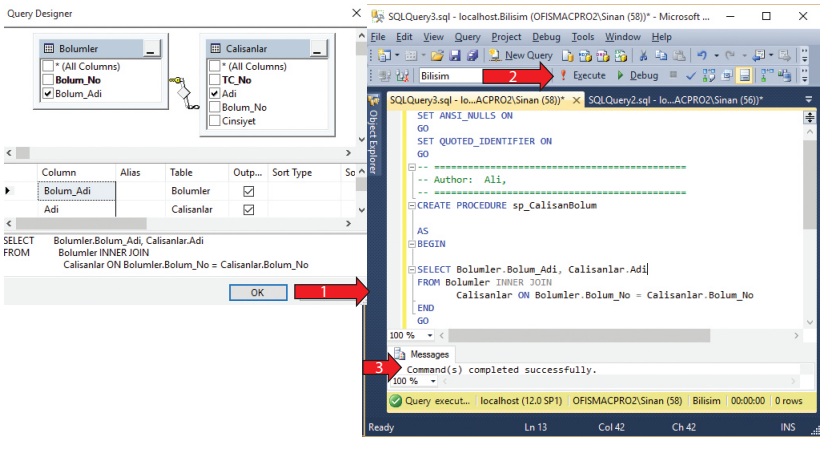
Açılan SQL sorgu penceresinde şablon bir yerel saklı yordam kodu hazır bulunmaktadır. Mevcut örnek için parametre girdisine ihtiyaç olmadığı için parametre ile ilgili kısımlar silinir. Burada yeşil ile gösterilen açıklamalar da silinebilir. "CREATE PROCEDURE" adımında yerel saklı yordamın adı yazılır. Daha sonra Resim 3.3'teki gibi "BEGIN-END" komutları arasında sağ tuşa tıklayarak "Design Query in Editor" seçilir.



Resim 3.3

Saklı yordam için gerekli sorguyu yardımcı kullanarak oluşturma için "Design Query in Editor" seçilir ya da Ctrl+Shift+Q kısayol tuşu kullanılır.

Açılan sorgu tasarım penceresinde (Query designer) saklı yordamda yer alacak sorgu tasarlanır. Daha sonra bu sorgu "Ok" tuşuna basılarak saklı yordam yazma ekranına SQL kodu ile geri dönülür (Resim3.4). Daha sonra hazırlanması biten saklı yordam "Execute" tuşuna basılarak çalıştırılır. Çalışma sonrasında eğer bir hata yoksa komutun başarılı olarak çalıştığına yönelik mesaj alınır.



Resim 3.4

Sorgu tasarım penceresinde Calisanlar ve Bolumler seçimi ve gösterilecek veri alanları seçimi

Bu örnekte grafik ortam kullanılarak sorgu oluşturulmuştur. Aşağıda yer alan komut çalıştırılarak bu saklı yardım oluşturulabilir.

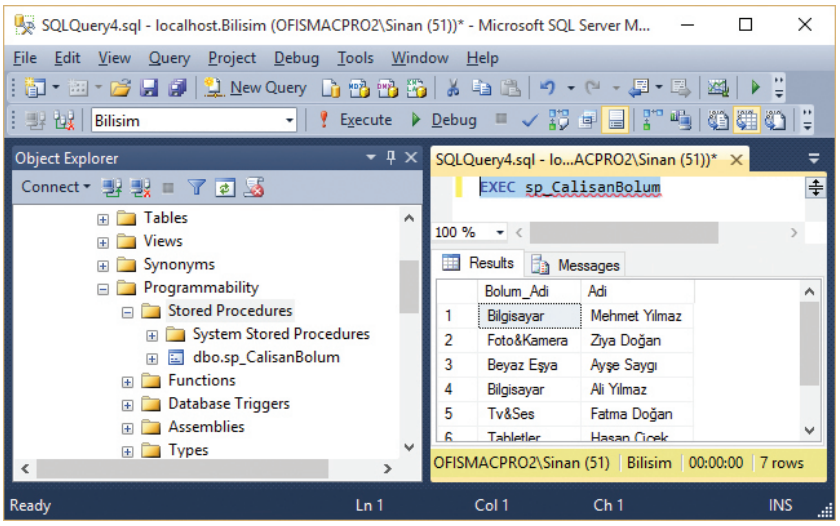
```
CREATE PROCEDURE sp_CalisanBolum
  AS
BEGIN
 SELECT Bolumler.Bolum_Adi, Calisanlar.Adi
    FROM Bolumler INNER JOIN
       Calisanlar ON Bolumler.Bolum_No = Calisanlar.Bolum_No
END
```

DİKKAT

Saklı yordam oluşturmak için kullanılan "Create Procedure" ifadesi bir kez çalıştırılabilir. İkinci kez çalıştırıldığında aynı isimli bir yordam tekrar oluşturulmak "istendiğinden bu nesne zaten var" şeklinde bir uyarı alınacaktır.

Resim 3.5

Yerel saklı yordamın nesne tarayıcısı penceresinde yeri ve EXEC sp_CalisanBolum komutu ile elde edilen sonuç



SIRA SİZDE 2

Örnek veritabanımız "Bilisim"de çalışan isimleri ve satış yaptıkları ürün isimlerini listeleyen saklı yordamı yazınız.

Yerel saklı yordamların amacı daha önceden optimize edilmiş komut kümelerinin oluşturulmasıdır. Ancak önemli bir işlevi ise özellikle istemcilerin standart isteklerine yanıt verecek yapılardır. Örneğin bir web sayfası kullanıcısı ya da bir banka terminalindeki yazılımın talep ettiği veriyi istemciye ulaştırmasıdır. Bu açıdan bakıldığında saklı yordamlar kullanıcıların veri isteklerini hızlı olarak hazırlayan ve gönderen yapılardır. Bir saklı yordam, istemciden bir veya daha fazla parametreyi alarak standartlaştırılmış sorgu ifadelerini çalıştırır. Çalıştırılan sorgular ile istemciye bir değer ya da bir tablo döndürülebilir, veri ekleme, güncelleme ya da silme işlemi yapılabilir.

Örnek Uygulama 3.3. Örnek veritabanında yer alan "Urunler" tablosunda "Urun_Sayısı" alanının belli bir sayıdan büyük olan ürünlerin "Urun_Adi" ve "Urun_Fiyati" alanlarını listeleyen bir yerel saklı yordam oluşturalım. Bu saklı yordamın oluşturulması için istemcinin bir parametre girmesi ve bu parametreye göre sorgunun çalıştırılarak verinin elde edilmesi sağlanmalıdır. Bu amaçla aşağıda oluşturulan komutlarda görüleceği üzere `@Urun_Sayisi_Parametresi` adında tamsayı(int) veri türünde bir parametre tanımlanmıştır.

```
CREATE PROCEDURE sp_UrunleriListele
(@Urun_Sayisi_Parametresi int )
AS
BEGIN
    SELECT * FROM Urunler WHERE Urun_Sayisi>@Urun_Sayisi_Parametresi
END
```

Oluşturulan "sp_UrunleriListele" saklı yordamı "EXEC sp_UrunleriListele 10" komutu ile 10 parametresi ile çalıştırılarak aşağıdaki sonuç elde edilir.

Urun_No	Urun_Adi	Urun_Sayisi	Bolum_No
12	Buzdolabı	12	2000.00
17	Samsung Telefon	12	2500.00

Dikkatli olan okuyucular Örnek 3.3'de istenilen çıktı ile elde edilen çıktının aynı olmadığının farkına varmışlardır. Bunun nedeni aslında sadece "Urun_Adi" ve "Urun_Fiyati"alanlarının listelenmesi istenmesine rağmen örnekte "*" ifadesi kullanarak tüm alanların listelenmesi sağlandı. Bu hatayı düzeltmek için mevcut yordamı "**DROP PROCEDURE sp_UrunleriListele**" komutu ile silip tekrar oluşturabiliriz ya da mevcut yordamı aşağıdaki komutu kullanarak düzeltebiliriz.

Aşağıdaki düzeltme işleminde @Urun_Sayisi_Parametresi ifadesinin yanına varsayılan değer eklendiğine dikkat ediniz. 0'olarak eklenen bu parametre yordamın parametresiz olarak çalıştırılması durumunda hata vermeden ilgili değerin 0 dan büyük olduğu satırları elde edecektir.

```
ALTER PROCEDURE sp_UrunleriListele
(@Urun_Sayisi_Parametresi int =0)
AS
BEGIN
  SELECT Urun_Adi, Urun_Fiyati FROM Urunler
WHERE Urun_Sayisi>@Urun_Sayisi_Parametresi
END
```

Örnek Uygulama 3.4: Örnek veritabanı "Bilisim" için bölümler tablosuna yeni bir kayıt ekleme işlemini saklı yordamları kullanarak yapalım.

Saklı Yordamı Oluşturan Komut	Saklı Yordamın Çalıştırılması ve Sonucu

`CREATE PROCEDURE sp_BolumEkle`
`(`
`@BolumAdi nvarchar(50)`
`)`
`AS`
`BEGIN`
`  INSERT INTO Bolumler(Bolum_Adi)`
`  VALUES (@BolumAdi)`
`END`
	`EXEC sp_BolumEkle "Ev Aletleri"`
	`SELECT * FROM Bolumler`
	`Sonuç:`

	Bolum_No	Bolum_Adi
1	1	Bilgisayar
2	2	Tabletler
3	3	Foto&Kamera
4	4	Beyaz Eşya
5	5	Tv&Ses
6	6	Telefon
7	7	Ev Aletleri

Bolum_No alanı birincil anahtar alanı olduğundan ve tablo oluşturulurken "Is Identity" alanı "True" yapıldığından tabloya veri girdikçe SQL sunucu tarafından otomatik olarak değer atanmaktadır. Biz bu alan için başlangıç değerini ve artış miktarını 1 yaptığımızdan 1'den başlayarak birer birer artan değerler bu alana atanmaktadır.

Örnek Uygulama 3.5. Örnek veritabanı için satışlar tablosunda fiyatı 2500 TL'den küçük olan ürünleri silen bir yerel saklı yordam tasarlayalım.

Saklı Yordamı Oluşturan Komut

```
CREATE PROCEDURE sp_UrunSatisSil
(
    @urunFiyati decimal(18,2)
)
AS
BEGIN
    DELETE FROM Satislar
    WHERE Fiyat<@urunFiyati
END
```

Saklı Yordamın Çalıştırılması ve Sonucu

```
EXEC sp_UrunSatisSil 2500.00
SELECT * FROM Satislar
```

Sonuç:

S...	Ur...	Calisan_TC_No	Mi...	Fiyat	Tarih
1	1	48975626633	2	2500.00	2015-03-10
6	11	78965412305	3	2500.00	2015-08-15

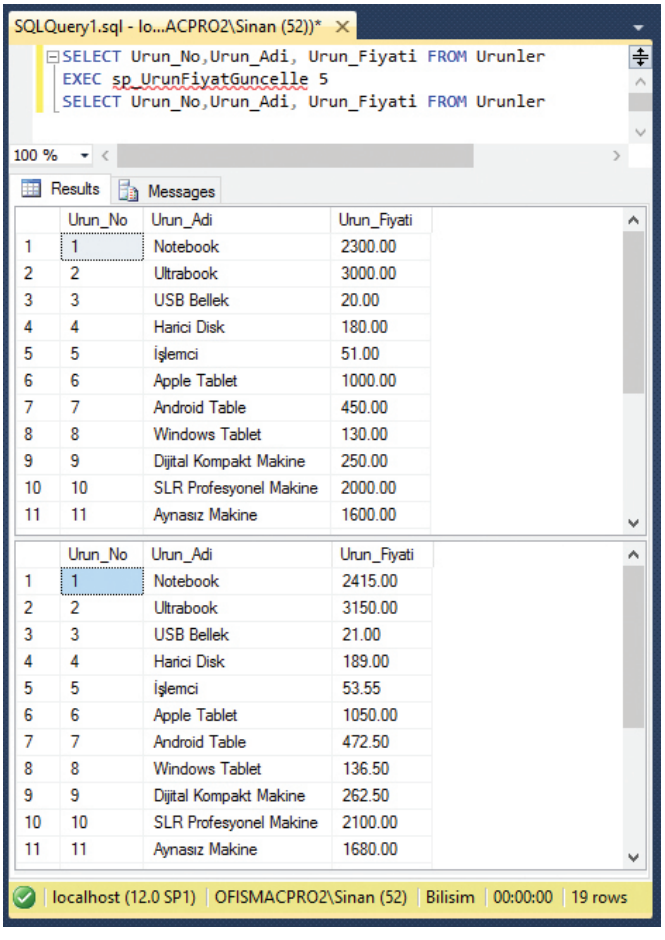
Örnek Uygulama 3.6: Örnek veritabanında yer alan "Urunler" tablosunda her bir ürünün fiyatını verilen orana göre arttıran saklı yordamı oluşturalım.

```
CREATE PROCEDURE sp_UrunFiyatGuncelle (@artisMiktariYuzde float )
AS
BEGIN
    UPDATE Urunler
    SET Urun_Fiyati=Urun_Fiyati+Urun_Fiyati*@artisMiktariYuzde/100
END
```

Ürün fiyatını güncelleme amacı ile oluşturulmuş saklı yordam Resim 3.6'da görüldüğü gibi çalıştırıldığında Urun fiyatındaki değişim fark edilecektir.

Resim 3.6

Urun fiyatını istenilen oranda arttıran saklı yordamın çalışma öncesi ve sonrası veriler



SİSTEM FONKSİYONLARI

Sistem fonksiyonlarının bir kısmı "@@" işareti ile başlar. SQL sunucu tarafından sunulan bazı fonksiyonlar parametre içermezler. SQL Server tarafından tanımlanan fonksiyonların bir kısmı evrensel değişkenler olarak da bilinip kullanıcı tarafından oluşturulamazlar. Tablo 3.2'de sistem fonksiyonlarının gruplamaları verilmiştir.

Tablo 3.2
SQL sunucu genel sistem fonksiyon grupları

Fonksiyon Grup Adı	İşlevi
Kümeleme Fonksiyonları (Aggregate Functions)	Belli bir veri kümesinde işlem yapıp tek değer döndüren; `Avg` (ortalama alır), `Count` (veri kümesi satır sayısını döndürür) vb. fonksiyonlardır.
Yapılandırma Fonksiyonları (Configuration Functions)	Sunucunun mevcut yapılandırılması hakkında bilgi veren; `@@Servername` (sunucu adı), `@@version` (SQL sunucu sürümü) vb. fonksiyonlardır.
İmleç Fonksiyonları (Cursor Functions)	İmleçler ile ilgili veri döndüren Bölüm 2 de örnek olarak kullanılan fonksiyonlardır.
Tarih ve zaman fonksiyonlar (Date and Time Functions)	Tarih ve zaman üzerinde işlemler yapan ve karakter, nümerik değer veya tarih bilgisi döndüren; `Getdate` (tarih al), `Month` (ay) vb. fonksiyonlardır.
Matematiksel Fonksiyonlar (Mathematical Functions)	Girdi değerlerine bağlı nümerik değer döndüren; `Log`(logaritma), `Abs` (mutlak değer) vb. fonksiyonlardır.
Metaveri Fonksiyonları (Metadata Functions)	Veritabanı ve veritabanı nesneleri üzerinde bilgi döndüren; `Object_Id`(Nesne numarası), `Object_Name`(Nesne Adı) vb. fonksiyonlardır.
Güvenlik Fonksiyonları (Security Functions)	Kullanıcılar ve rolleri hakkında bilgi döndüren; `User_Id` (kullanıcı numarası), `User_Name` (kullanıcı adı) vb. fonksiyonlardır.
Dizgi Fonksiyonları (String Functions)	Dizgiler üzerinde işlemler yapan; `Len` (dizgi uzunluğu), `Reverse` (dizgiyi tersine çeviren) vb. fonksiyonlardır.
Sistem ile ilgili statiksel Fonksiyonlar (System Statistical Functions)	İstatiski bilgi sağlayan; `@@Total_Errors` (toplam hata sayısı), `@@Total_Read` (toplam okuma sayısı) vb. fonksiyonlardır.

Sistem fonksiyonlarına nesne tarayıcısı penceresinde "Bilisim → Programmability → Functions → System Functions" dizini altından ulaşılabilir. Kitap içinde farklı yerlerde sistem fonksiyonlarının kullanımı örnekler içinde verilmektedir.

SQL sunucunun dilini gösterip Türkçe'ye ayarlamak için kullanılacak SQL komutlarını yazınız.

SIRA SİZDE 3

KULLANICI TANIMLI FONKSİYONLAR

Kullanıcı tanımlı fonksiyonlar, kullanıcılar tarafından tanımlanan tek bir değer veya tablo döndürmek için kullanılan ilişkisel veritabanı nesneleridir. **SQL** sunucu içerisinde tanımlı olmayan fonksiyonları hazırlamak için kullanıcı tanımlı fonksiyonlar kullanılır. Saklı yordamlardan farkı bu bölümün başında verilmiştir. Örnek veritabanı olan Bilisim veritabanımızın üzerinde yer alan kullanıcı tanımlı fonksiyonlara erişebilmek için, nesne tarayıcı penceresinde "Bilisim → Programmability → Functions" dizininden erişebiliriz. Kullanıcı tanımlı fonksiyon oluşturmak için kullanılan temel SQL komutları Tablo 3.3'te açıklamalarıyla birlikte verilmektedir.

Tablo 3.3 *Kullanıcı tanımlı fonksiyon için kullanılan komutlar*

Komut Adı	Açıklama
CREATE FUNCTION	Fonksiyon oluşturmak için kullanılır.
ALTER FUNCTION	Fonksiyonda değişiklik yapmak için kullanılır.
DROP FUNCTION	Mevcut olan fonksiyonu silmek için kullanılır.

Bilgisayar programlama dillerinde fonksiyonlar genellikle parametre göndererek bir değer döndüren yapılardır. Buradaki fonksiyonlar buna ek olarak bir tablo olarak da geri dönebilmektedir.

Sayıl değerli fonksiyonlar (Scalar-valued functions), Tablo değerli fonksiyonlar (Table-valued functions) ve Kümeleme fonksiyonları (Aggregate functions) olmak üzere üç çeşit kullanıcı tanımlı fonksiyon mevcuttur.

Sayıl Değerli Fonksiyonlar

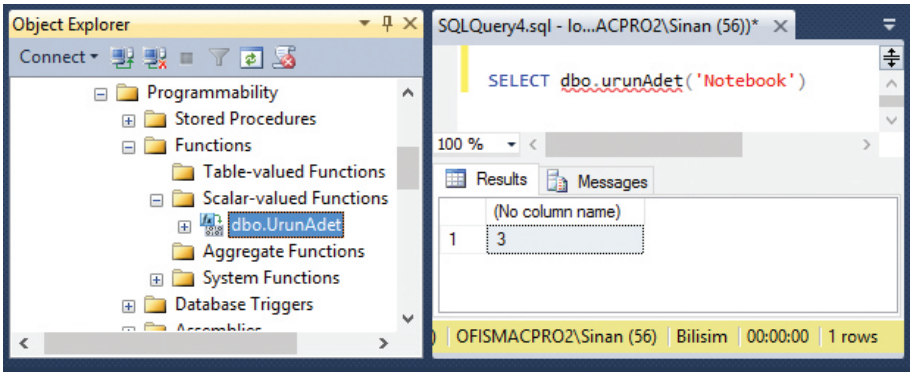
Tek bir sayısal değer döndüren fonksiyonlardır. Genel yazım şekli aşağıdadır.

```
CREATE FUNCTION Fonksiyon_Adi(Parametreler)
RETURNS geriDonusTipi
AS
BEGIN
   -- Sorgular
RETURN geriDonusDegeri
END
```

Örnek Uygulama 3.7: "Urunler" tablosunda istenilen üründen kaç adet olduğunu bulup değer döndüren kullanıcı tanımlı fonksiyon oluşturalım. "Create Function" komutuyla başlayacak bu komut dizesi kendi adı ile bir değer döndürecektir.

```
CREATE FUNCTION UrunAdet(@urunAdi nvarchar(50))
RETURNS int
AS
BEGIN
     DECLARE @urunAdedi int
      SET @urunAdedi=(SELECT Urun_Sayisi FROM Urunler WHERE Urun_Adi=@
urunAdi)
     RETURN @urunAdedi
END
```

Fonksiyon içerisinde yer alan "`SET @urunAdedi=(SELECT Urun_Sayisi FROM Urunler WHERE Urun_Adi=@urunAdi)`" ifadesinde farklı bir kullanımı görmekteyiz. Bu yapıda bir sorgunun içeriği bir değişkene atanmıştır. Burada ilgili sorgu mutlaka tek bir değer döndürecek şekilde tasarlanmalıdır. Aksi durumda hataya neden olacağı tahmin edilebilir. Oluşturduğumuz fonksiyonu çalıştırmak için `SELECT dbo.urunAdet('Notebook')` sorgusunu kullanabiliriz. Oluşturduğumuz fonksiyonun bulunduğu konumu ve çalıştırma sonucu aşağıda yer almaktadır.



Fonksiyonun oluşturulduğu veritabanı ile çağrıldığı veritabanı aynı olmalıdır. Örneğin, bir fonksiyonu "master" veritabanı altında oluşturup "Bilisim" veritabanı içinde kullanamazsınız. Bu nedenle veritabanı nesnelerini oluşturmak için oluşturduğumuz komutları doğru veritabanı için çalıştırmak oldukça önemlidir.

Örnek Uygulama 3.8. Örnek veritabanında belirli bir ayda en fazla ciro elde eden satışları yapan personelin adını getiren bir fonksiyon oluşturulmak istensin. Bu durumda "Satislar" tablosundaki tarih bilgisini sınırlayacak iki parametre tanımlanmalıdır.

```
Create Function En_fazla_SatisYapan (@Yil int, @Ay int) Returns Nvarchar(100)
    AS
    BEGIN
    Declare @Calisan Nvarchar(100)

    SELECT TOP (1) @Calisan=Calisanlar.Adi
      FROM  Satislar INNER JOIN Calisanlar
            ON Satislar.Calisan_TC_No = Calisanlar.TC_No
      --Ay ve Yıl Parametreleri ile veri kümesi sınırlanır
      WHERE (YEAR(Satislar.Tarih) = @Yil) AND (MONTH(Satislar.Tarih) = @Ay)
      --Çalışana göre gruplanarak Toplam Satışa göre azalan sıralanır.
      GROUP BY Satislar.Calisan_TC_No, Calisanlar.Adi
      ORDER BY SUM(Satislar.Fiyat) DESC

      Return @Calisan
    END
```

3.8 örnek uygulamasına benzer şekilde bu örnekte de "`SELECT TOP (1) @Calisan=Calisanlar.Adi.....`" yapısı ile azalan olarak sıralanmış sorgunun ilk ifadesi @Calisan değişkenine aktarılmaktadır. Fonksiyonun "`Select [dbo].[En_fazla_SatisYapan](2015,7)`" komutu ile 2015 yılı 7. ay için çalıştırıldığında "Ali Yılmaz" sonucunu getireceğini kendi bilgisayarınızda deneyimleyebilirsiniz.

Tablo Değerli Fonksiyonlar

Tablo değerli fonksiyonlar (table-valued functions) çalıştıktan sonra geriye bir tablo ile dönen fonksiyonlardır. Bu nedenle bu tür sorgular genellikle bir seçme sorgusunun tablosu yerine yazılabilirler (Select * from fn_tablogetir()). Genel yazım şekli aşağıdadır.

```
CREATE FUNCTION Fonksiyon_Adi(Parametreler)
RETURNS TABLE
AS
RETURN (SELECT SQL_sorgusu)
```

Örnek Uygulama 3.9: "Calisanlar" tablosunu listeleyen bir fonksiyon oluşturalım ve bu fonksiyonu kullanarak çalışanları listeleyelim.

```
CREATE FUNCTION fn_calisanlariListele()
RETURNS TABLE
AS
RETURN SELECT * FROM Calisanlar
```

Oluşturulan bu fonksiyon "`Select * from fn_calisanlariListele()`" komutu ile kullanılabilir. Sonuç olarak "`SELECT * FROM Calisanlar`" sorgusu ile aynı sonucu vereceği açıktır.

Bu ünitedeki uygulama komut dosyalarına https://goo.gl/IZUEzf adresinden ulaşabilirsiniz. Ayrıca ilk ünitedeki komut satırı dosyası da çalıştırılarak veritabanının oluşturulması gerektiğini unutmayınız.

Örnek Uygulama 3.10: Örnek 3.9'daki fonksiyonu cinsiyeti girdi parametresi alacak şekilde 'ALTER' komutu ile güncelleyelim.

```
ALTER FUNCTION fn_calisanlariListele(@cinsiyet nchar(1))
RETURNS TABLE
AS
RETURN SELECT * FROM Calisanlar WHERE Cinsiyet=@cinsiyet
```

Oluşturulan tablo değerli fonksiyonu "E" parametresi ile "`SELECT * FROM fn_calisanlariListele('E')`" komutu ile çalıştırabiliriz. Bu durumda elde edilecek veri kümesi aşağıda gösterilmiştir.

Sonuç:

TC_No	Adi	Bolum_No	Cinsiyet
17895632417	Ziya Doğan	3	E
65638749631	Ali Yılmaz	1	E
89652341780	Hasan Çiçek	2	E

Alter komutu ile fonksiyona yapılan güncellemeden sonra fonksiyonun eski hâli artık kullanılamaz. Parametre girdisi verilmesi gerekir.

Oluşturulan fn_calisanlariListele adlı fonksiyon "DROP FUNCTION fn_calisanlariListele" komutu ile silinebilir.

Tablo değerli fonksiyonlarda tablo tanımlanarak da geri dönüş sağlanabilir. Bu bir sorgu sonucunu döndürmek yerine daha karmaşık işlemler sonucu çıkan çıktı, tabloyu geri döndürmek için kullanılabilir. Genel yazım şekli aşağıdadır.

```
CREATE FUNCTION Fonksiyon_Adi(Parametreler)
RETURNS @TableName TABLE (Alan_tanımlaması)
AS
BEGIN
     SQL_Sorgu
     RETURN
END
```

Örnek Uygulama 3.11: Örnek veritabanı için hangi bölümden kaç adet ürün satıldığı ile ilgili bölüm adı ve satılan ürün adedi bilgisini dönen fonksiyonu oluşturup ve bu fonksiyonu kullanarak 1 nolu bölümden yapılan satışları listeleyelim.

```
CREATE FUNCTION fn_SatisBilgileri(@BolumNo int)
RETURNS @BolmSatTablosu TABLE
(    Bolum_Adı nvarchar(50),
     UrunSayisi int
)
AS
BEGIN
      INSERT INTO    @BolmSatTablosu
         SELECT Bl.Bolum_Adi,SUM(Sat.Miktar)
         FROM Urunler Ur, Bolumler Bl, Satislar Sat
         WHERE Bl.Bolum_No=@BolumNo AND
         Ur.Bolum_No=Bl.Bolum_No AND
         Ur.Urun_No=Sat.Urun_No
         GROUP BY Bl.Bolum_Adi, Bl.Bolum_No
         HAVING Bl.Bolum_No=@BolumNo
     RETURN
END
```

```
Çalıştırma:
    SELECT * FROM fn_SatisBilgileri(1)
Sonuç:
```

Bolum_Adı	UrunSayisi
Bilgisayar	11

Örnek 3.12, işletmede bölüm bazında yapılan satışların listelenmesini içermektedir. Bu birçok işletme için günlük ve aylık raporların alınmasında katkı sağlayacaktır.

Örnek "Bilisim" veritabanında Çalışan isimleri ve satış sayılarını listeleyen fonksiyonu yazınız.

SIRA SİZDE 4

Kümeleme Fonksiyonları

Kullanıcı tanımlı fonksiyonların sık kullanılmayan türüdür. Sistem fonksiyonları altında bulunan MIN(), MAX(), AVG(), SUM() vb. fonksiyonlar bu tip fonksiyonlara örnektir. Ancak sistem fonksiyonları altında bulunan kümeleme fonksiyonları, ihtiyaç olan bir işlem için yeterli değil ise kullanıcı tanımlı fonksiyon tanımlanır. Kümeleme fonksiyonları diğer SQL fonksiyonlar gibi oluşturulmazlar. NET platformunda yeni bir sınıf oluşturarak SQL server yazılım kodları ile birleştirilmesi gerekir.

Özet

Saklı yordamların ve fonksiyonların kullanımlarını açıklamak.

Saklı yordamlar ve fonksiyonlar, özellikle veritabanı sistemi üzerinde farklı kullanıcıların ortak ihtiyacı olan belirli bir görevi yerine getirmek için tasarlanmış, sunucu üzerinde tutulan, birden fazla tablo üzerinde işlem yapabilen, program içinden farklı parametreler ile çağrılarak kullanılabilen SQL tabanlı komut kümeleridir. Saklı yordam ve fonksiyonlar, sistem tarafından özel bir amaç için tanımlanmış olabileceği gibi kullanıcı tarafından tanımlanması da mümkündür. Saklı yordamların ve fonksiyonlarının genel yapısı birbirine benzemekle beraber aralarında bazı kritik farklar vardır. Kullanıcının tanımladığı fonksiyon SQL sorgularının içinde kullanılabilirken, saklı yordamlar kullanılamaz. Fonksiyonlar sayılı değer veya tablo döndürürken saklı yordamların ille de bir değer döndürmesi gerekmez. Saklı yordamlar derlenmiş olarak veritabanında tutulduğu için oldukça hızlı çalışırlar ve fonksiyonlara göre önemli avantaj sağlarken, fonksiyonlar çalışma zamanında derlenir ve çalışırlar.

T-SQL'de uygulamaya yönelik yerel saklı yordam hazırlamak veya kullanmak.

Saklı yordamlar; sistem saklı yordamları, genişletilmiş saklı yordamlar ve yerel saklı yordamlar olmak üzere üçe ayrılmaktadır. Sistem saklı yordamlar, sistemle ilgili yönetim ve bilgi alma için kullanılırlar. Genişletilmiş saklı yordamların ileriki sürümlerde desteklenmeyeceği belirtilmektedir. Yerel saklı yordamlar ise kullanıcı tarafından tanımlanabilen saklı yordamlardır. Saklı yordamlar yaptıkları işlem sonuçlarını döndürmekten ziyade veritabanı üzerinde tutmaya yöneliktir. Bir defa kullanıldıklarında derlenmiş hâlleri sunucu üzerinde saklandığı için sonraki kullanımda oldukça hızlıdırlar. Bir saklı yordam diğer saklı yordamların içinden çağrılarak kullanılabilir. Saklı yordamların tanımlanan giriş ve çıkışları değişmediği sürece, kullanıcıya yansıtılmadan saklı yordamların kodlanmasında değişiklik yapılabilir. Bu ise veritabanı programlamayı kullanıcıdan bağımsız hâle getirir. Modüler yazılım mimarisi için önemli bir katkı sağlarlar.

T-SQL'de uygulamaya yönelik kullanıcı tanımlı fonksiyon hazırlamak veya kullanmak.

Fonksiyonlar, sistem fonksiyonları ve kullanıcı tanımlı fonksiyonlar olmak üzere ikiye ayrılır. Sistem fonksiyonları ile matematiksel işlemlerden, sunucu yapılandırılmasına veya kullanıcı rolleri hakkında bilgi döndürmeye kadar geniş yelpazede özellikleri tüm kullanıcılara sağlarlar. Fonksiyonlar tablo veya sayılı değer döndürdüğü için SQL sorgu yapılarının farklı yerlerinde kullanılabilirler. Kullanıcı tanımlı fonksiyonlar işlem yapılacağı veritabanı üzerinde tanımlanmalıdır.

Kendimizi Sınayalım

1. Saklı yordamlar ve fonksiyonlar ile ilgili aşağıdaki ifadelerden hangisi **yanlıştır?**

a. Saklı yordam ve fonksiyonlar dışarıdan parametreler ile veri alabilir.
b. Fonksiyonlar ilk kullanımdan sonra derlendiği için hızlı çalışırlar.
c. Saklı yordamlar ağ trafiğinin azaltılmasına katkı sağlarlar.
d. Fonksiyonların içinde SQL sorguları kullanılabilir.
e. Saklı yordamların içinde SQL sorguları kullanılabilir.

2. Aşağıdakilerden hangisi **yanlıştır?**

a. Sistem saklı yordamları imleç uygulamaları için kullanılabilir.
b. Sistem fonksiyonları imleç uygulamaları için kullanılabilir.
c. Saklı yordamlar SQL sunucu veritabanı motorunun bakımı için kullanılabilir.
d. Fonksiyonların içinde hata yönetimi yoktur.
e. Sunucu üzerinde bulunan veritabanlarını ve boyutlarını öğrenmek için saklı yordamlar kullanılamaz.

3.

```
CREATE PROCEDURE sp_BolumEkle
(  @BolumAdi nvarchar(50) )
AS
BEGIN
    INSERT INTO Bolumler(Bolum_Adi)
    VALUES (@BolumAdi)
END
```

Yukarıdaki SQL kodu için aşağıdaki ifadelerden hangisi **yanlıştır?**

a. Bir saklı yordam oluşturur.
b. @BölümAdi değişkeni girdi parametresidir.
c. Çalıştırılması için EXEC komutu kullanılması gerekir.
d. Çalıştırıldığında @Bölüm Adi parametresi ile geri döner.
e. sp_BolumEkle isimli saklı yordamı oluşturur.

4.

```
CREATE PROCEDURE sp_UrunFiyat
( @deger float )
AS
BEGIN
    UPDATE Urunler
    SET Urun_Fiyati=Urun_Fiyati+@deger
END
```

Yukarıdaki saklı yordam ile ilgili ifadelereden hangisi doğrudur?

a. Ürün fiyatını % bir değer ile artırır.
b. Alan adı belirtilmediği için çalışmaz.
c. Ürün fiyatı ilgili değişken değeri kadar artırılıp tabloda güncellenir.
d. Saklı yordam geriye ilgili değişkeni döndürür.
e. "EXEC sp_UrunFiyat" ile çalışır.

5. Bir saklı yordamı silmek için kullanılan komut aşağıdakilerden hangisidir?

a. DELETE
b. REMOVE
c. INKOKE
d. DROP
e. RELEASE

6. Aşağıdakilerden hangisi bir kümeleme fonksiyonudur?

a. FETCH
b. SUM
c. SELECT
d. HAVING
e. ALTER

7. Aşağıdaki fonksiyonlardan hangisi "Microsoft SQL Server 2014 - 12.0.2269.0 (X64)" şeklinde veri geri döndürür?

a. @@version
b. Getdate
c. Getname
d. GetTotalRead
e. @@Servername

8. Kullanıcı tanımlı fonksiyonlar ile ilgili aşağıdaki ifadelerden hangisi **yanlıştır?**

a. Parametre ile çalışabilirler
b. Çalışma sonunda mutlaka bir değer ya da tablo döndürürler.
c. Veri güncelleme işlemi gerçekleştirebilirler
d. Tekrarlı işlemler için uygun yapılardır.
e. Saklı yordam içinden çağrılabilirler.

9.

```
Create Function soru9(@a int ,@b int) Returns int as
Begin
      declare @c int
      IF @a>@b
            set @c=@a+@b
       Else
     set @c=@b-@a
Return @c+@b+@a
End
```

Yukarıda tanımlanan fonksiyon dikkate alındığında Select dbo. soru9(5,5) sorgusunun sonucu aşağıdakilerden hangisidir?

a. 3
b. 5
c. 12
d. 10
e. 15

10. Sistem fonksiyonu olan 'MAX' için aşağıdakilerden hangisi doğrudur?

a. Bir T-SQL kodundaki oluşan en büyük hatayı verir.
b. Tablonun alanlarındaki en büyük rakamsal değeri döndürür.
c. ERROR_STATE() ile aynı sonucu döndürür.
d. Sadece saklı yordam içinde kullanılabilir.
e. Sayı dizisindeki maksimum değeri döndürür.

Kendimizi Sınayalım Yanıt Anahtarı

1. b	Yanıtınız yanlış ise "Saklı Yordamlar ve Fonksiyonların Girişi" konusunu yeniden gözden geçiriniz.
2. e	Yanıtınız yanlış ise "Sistem Saklı Yordamları" konusunu yeniden gözden geçiriniz.
3. d	Yanıtınız yanlış ise "Yerel Saklı Yordamlar" konusunu yeniden gözden geçiriniz.
4. c	Yanıtınız yanlış ise "Yerel Saklı Yordamlar" konusunu yeniden gözden geçiriniz.
5. d	Yanıtınız yanlış ise "Yerel Saklı Yordamlar" konusunu yeniden gözden geçiriniz.
6. b	Yanıtınız yanlış ise "Kümeleme Fonksiyonları" konusunu yeniden gözden geçiriniz.
7. a	Yanıtınız yanlış ise "Sistem Fonksiyonları" konusunu yeniden gözden geçiriniz
8. c	Yanıtınız yanlış ise "Kullanıcı Tanımlı Fonksiyonlar" konusunu yeniden gözden geçiriniz
9. d	Yanıtınız yanlış ise "Kullanıcı Tanımlı Fonksiyonlar" konusunu yeniden gözden geçiriniz.
10. e	Yanıtınız yanlış ise "Kümeleme Fonksiyonları" konusunu yeniden gözden geçiriniz

Sıra Sizde Yanıt Anahtarı

Sıra Sizde 1

Yazılımlarda hangi işlem için hangi komutu kullanacağımızı bulmak için genellikle yardım dosyalarını ya da web sayfalarını kullanmaktayız. Yazılım komutları ve yardım dosyaları genellikle İngilizce olduğundan Web sayfalarında da İngilizce ifadelerle aradığımız komutlara ulaşmamız mümkündür. Örneğin burada "T-sql Stored Procedure Reports information about a database object" benzeri bir arama ifadesi bizi aradığımız yordama ulaştırabilecektir.

"sp_help" saklı yordamı belli bir veritabanı nesnesi hakkında detay bilgi almak için kullanılabilir. Bunun için

```
EXEC sp_help 'Urunler'
```

komutunun çalıştırılması yeterlidir.

Sıra Sizde 2

Yerel saklı yordamlar yazılırken veritabanı yönetim sisteminde gerçekleştirilmek istenen işlemleri gerçekleştirecek komutlar tasarlanmalıdır. Bu sayede ilgili kod create procedure program bloğu arasına yerleştirilerek çalıştırılır.

```
CREATE PROCEDURE sp_CalisanUrun
AS
BEGIN
    SELECT Cal.Adi, Urn.Urun_Adi
    FROM Calisanlar Cal, Satislar Sat, Urun-
ler Urn
      WHERE Cal.TC_No=Sat.Calisan_TC_No AND
Sat.Urun_No=Urn.Urun_No
END
```

Sıra Sizde 3

SQL Sunucu için geçerli dili öğrenmek için "SELECT @@ LANGUAGE". Dil ayarını değiştirmek için "SET LANGUAGE 'turkish'" kullanılabilir.

Sıra Sizde 4

```
CREATE FUNCTION fn_SatisBilgileri()
RETURNS @BolmSatTablosu TABLE
(   Calisan_Adı nvarchar(50),
    UrunSayisi int
)
AS
BEGIN
     INSERT INTO    @BolmSatTablosu
       SELECT Cal.Adi, COUNT(*)
        FROM Calisanlar Cal, Satislar Sat
        WHERE Cal.TC_No=Sat.Calisan_TC_No
      GROUP BY Sat.Calisan_TC_No, Cal.Adi

    RETURN
END

Çalıştırma:
     SELECT * FROM fn_SatisBilgileri()
```

Sonuç:

Calisan _Adı	UrunSayisi
Ayşe Saygı	3
Ali Yılmaz	4
Fatma Doğan	2
Kader Kara	1

Yararlanılan ve Başvurulabilecek Kaynaklar

Özseven, T. (2014). **Veri Tabanı Yönetim Sistemleri 2**, Trabzon, Türkiye.

Elmasri, R., Navathe, S.B., (2011). **Fundamentals of Database Systems**, Sixth Edition, Addison-Wesley, USA.

Ullman, J., Widom, J., (2001). **A first course in Database Systems**, 2nd edition, Prentice Hall.

Yarımağan, Ü (2010). **Veri Tabanı Yönetim Sistemleri**, Akademi Yayıncılık.

Transact-SQL Reference: https://msdn.microsoft.com/en-us/library/bb510741.aspx, son erişim tarihi: 12.11.2015