Veri Tabanı Yönetim Sistemleri

SQL Server VERİ TABANI OLUŞTURMA

- CREATE DATABASE ifadesi yeni bir veri tabanı oluşturur. Aşağıda CREATE DATABASE ifadesinin minimal sözdizimi gösterilmektedir:

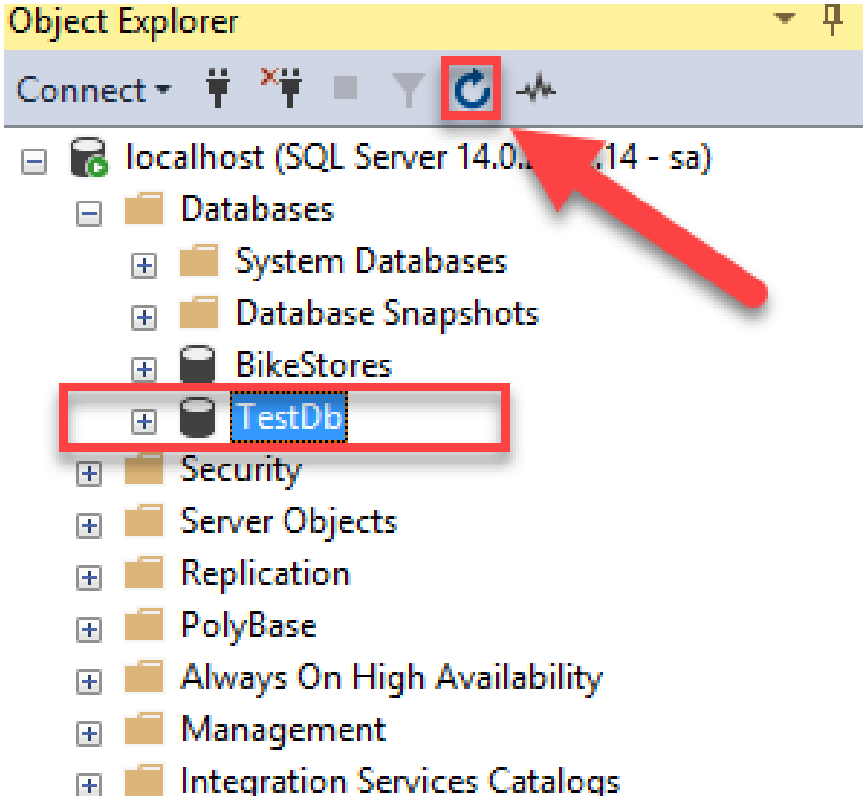
```
CREATE DATABASE database_name;
```

- Bu sözdiziminde CREATE DATABASE anahtar sözcüğünden sonra veri tabanının adı belirtilir.
- **NOT:** Veri tabanı adı, SQL Server örneği içinde benzersiz olmalıdır. Ayrıca SQL Server tanımlayıcısının kurallarına da uymalıdır. Genellikle, veri tabanı adı en fazla 128 karaktere sahiptir.

- Aşağıdaki ifade TestDb isimde yeni bir veri tabanı oluşturur:

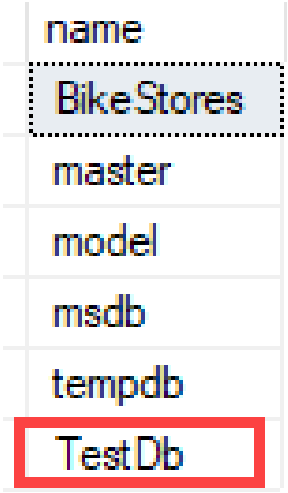
```
CREATE DATABASE TestDb;
```

- İfade başarıyla yürütüldüğünde, yeni oluşturulan veri tabanını Nesne Gezgini'nde görüntüleyebilirsiniz. Yeni veri tabanı görünmüyorsa, nesne listesini güncellemek için Yenile düğmesine tıklayabilir veya F5 klavye tuşuna basabilirsiniz.



- Bu ifade SQL Server'daki tüm veri tabanlarını listeler:

```
SELECT
    name
FROM
    master.sys.databases
ORDER BY
    name;
```

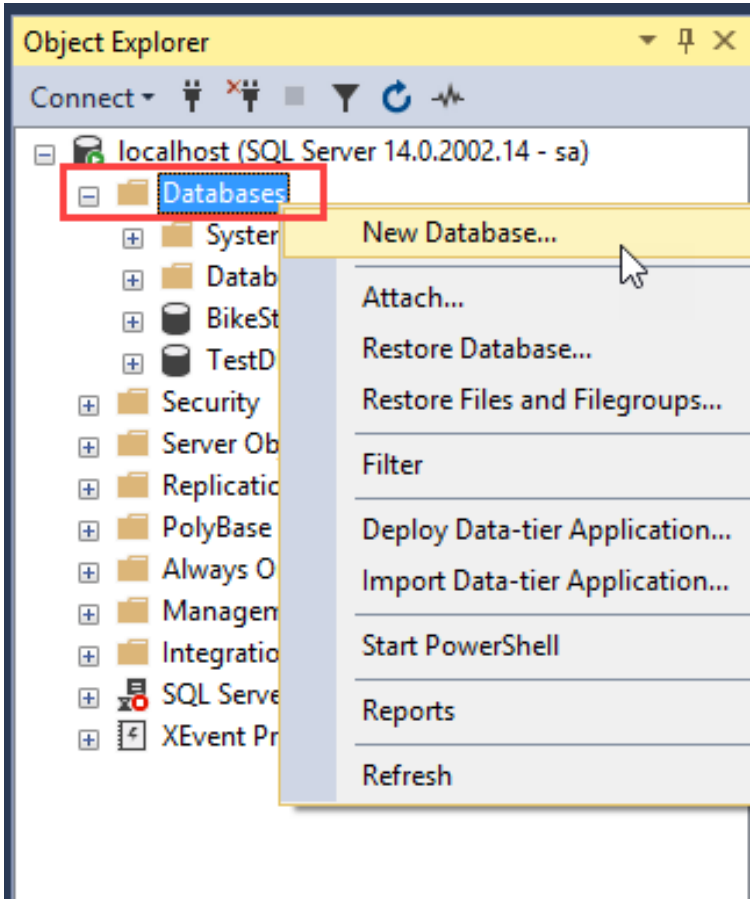


- Veya sp_databases stored procedure'ünü çalıştırabilirsiniz.

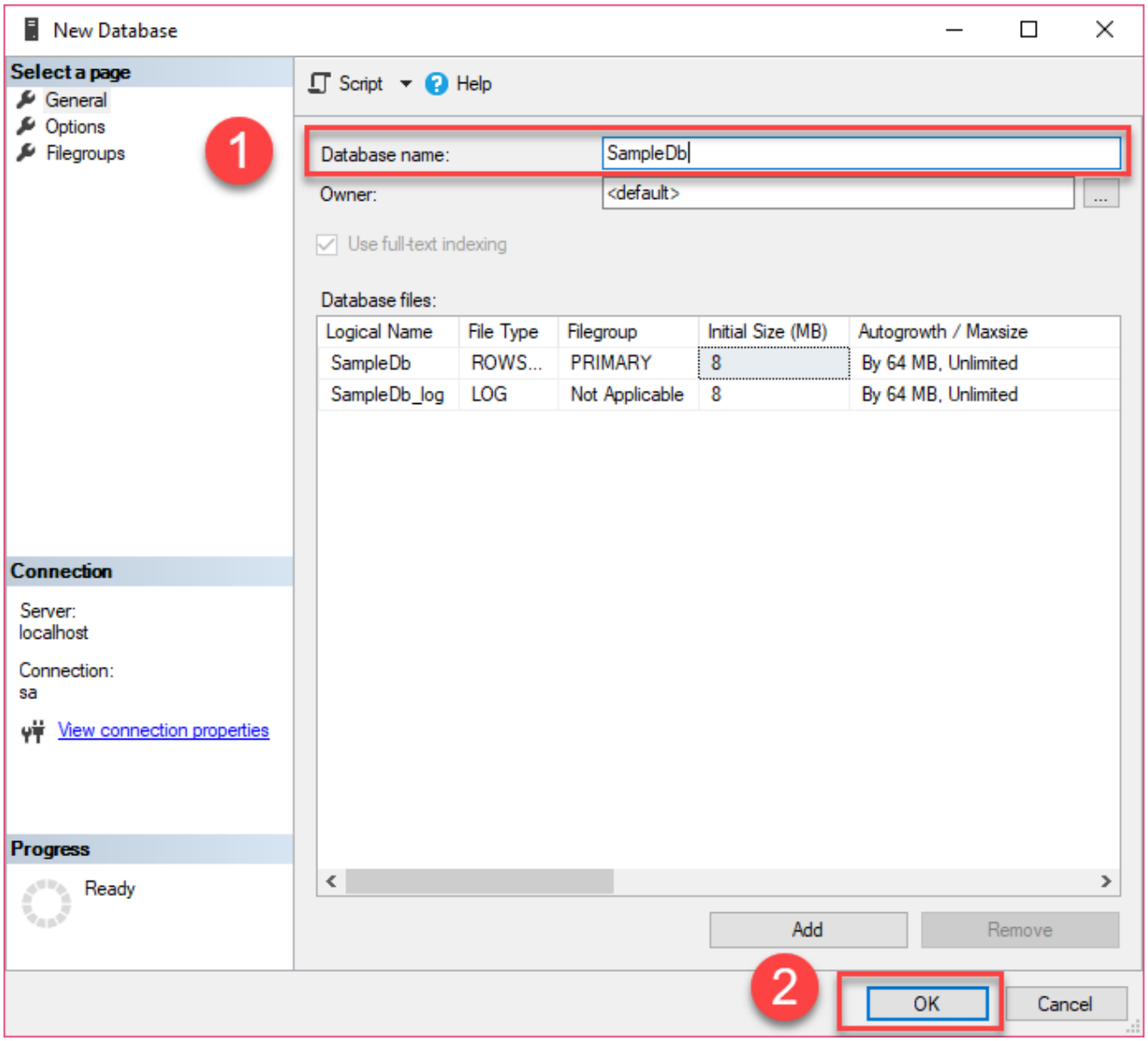
```
EXEC sp_databases;
```

SQL Server Management Studio'yu kullanarak yeni bir veri tabanı oluşturma

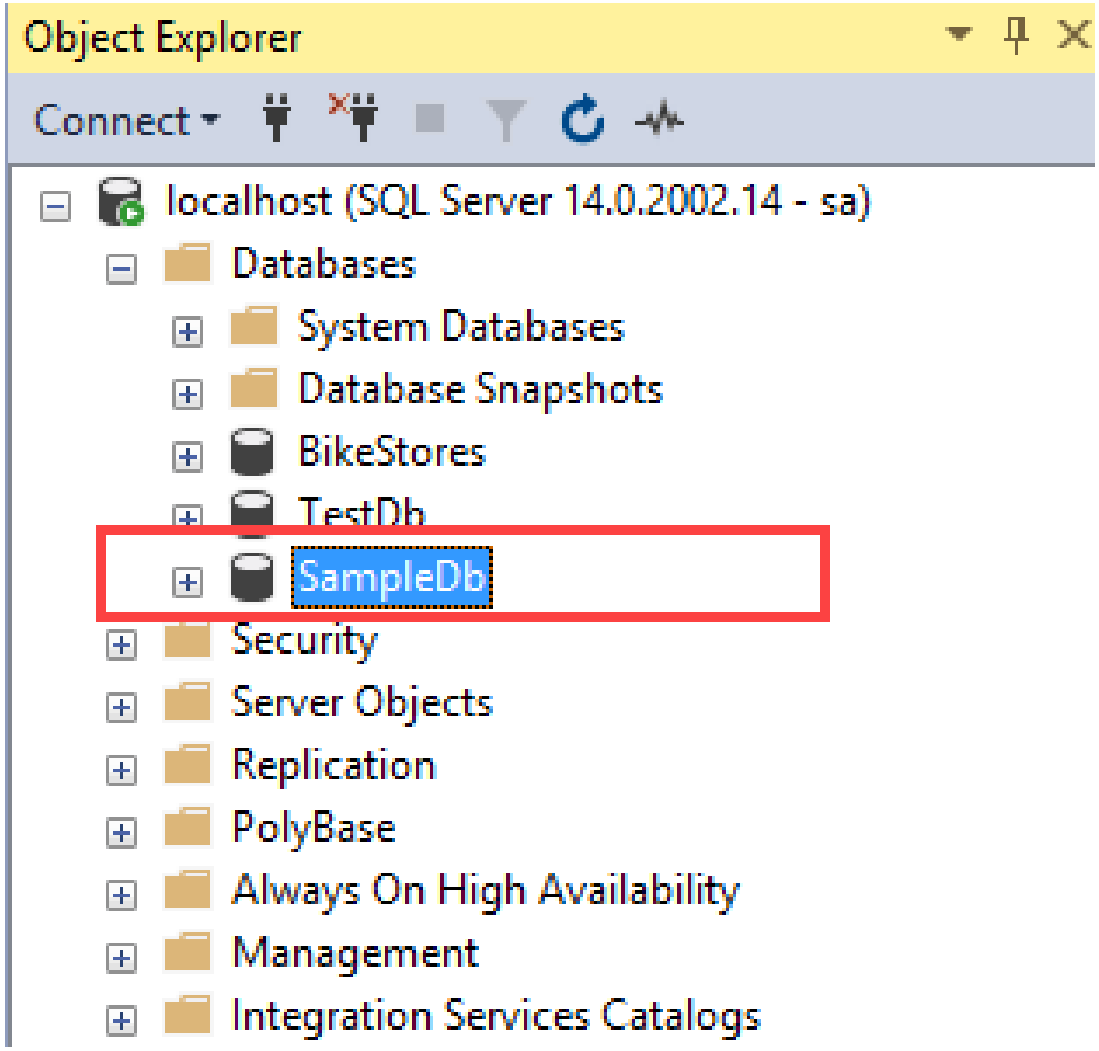
- Öncelikle **Veritabanına sağ tıklayıp Yeni Veri tabanı…** menü öğesini seçin.



- İkinci olarak veritabanının adını (örneğin **SampleDb) girin ve Tamam** butonuna tıklayın .



- Üçüncüsü, yeni oluşturulan veritabanını Nesne Gezgini'nden görüntüleyin:



SQL Server VERİ TABANINI SİLME (DROP)

Bir veri tabanını silmek için SQL Server DROP DATABASE ifadesini kullanma

- Mevcut bir veri tabanını bir SQL Server örneğinden kaldırmak için bu DROP DATABASE ifadesi kullanılır.
- DROP DATABASE ifadesi, aşağıdaki sözdizimiyle bir veya daha fazla veri tabanını silmenize olanak tanır:

```
DROP DATABASE  [ IF EXISTS ]
    database_name
    [,database_name2,...];
```

```
DROP DATABASE  [ IF EXISTS ]
    database_name
    [,database_name2,...];
```

- Bu sözdiziminde DROP DATABASE anahtar sözcüğünden sonra silmek istediğiniz veri tabanının adını belirtirsiniz.
- Tek bir ifade kullanarak birden fazla veri tabanını silmek istiyorsanız, DROP DATABASE anahtar sözcüğünden sonra virgülle ayrılmış bir veri tabanı adları listesi kullanabilirsiniz.
- IF EXISTS seçeneği SQL Server 2016'dan (13.x) edinilebilir. Bir veri tabanını yalnızca veri tabanı zaten mevcutsa koşullu olarak silmenize olanak tanır.
- IF EXISTS seçeneğini belirtmeden var olmayan bir veri tabanını silmeye çalışırsanız, SQL Server bir hata verecektir.

- Bir veri tabanını silmeden önce aşağıdaki önemli noktalardan emin olmalısınız:
  - İlk olarak, DROP DATABASE ifadesi veri tabanını ve veri tabanı tarafından kullanılan fiziksel disk dosyalarını siler. Bu nedenle, gelecekte geri yüklemek istemeniz durumunda veri tabanının bir yedeğine sahip olmalısınız.
  - İkincisi, şu anda kullanılan veri tabanını bırakamazsınız.
- Şu anda kullanılan bir veri tabanını bırakmaya çalışmak aşağıdaki hataya neden olur:

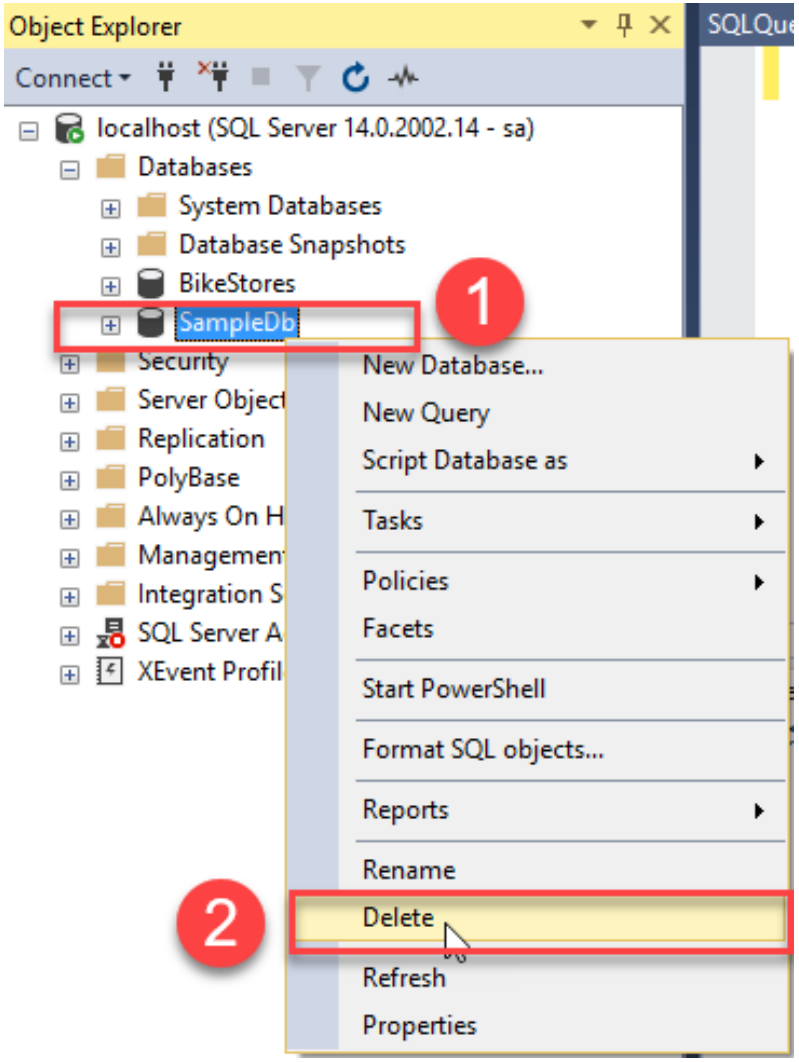
```
Cannot drop database "database_name" because it is currently in use.
```

- Aşağıdaki örnekte, TestDb veri tabanını silmek için DROP DATABASE ifadesi kullanılmaktadır:

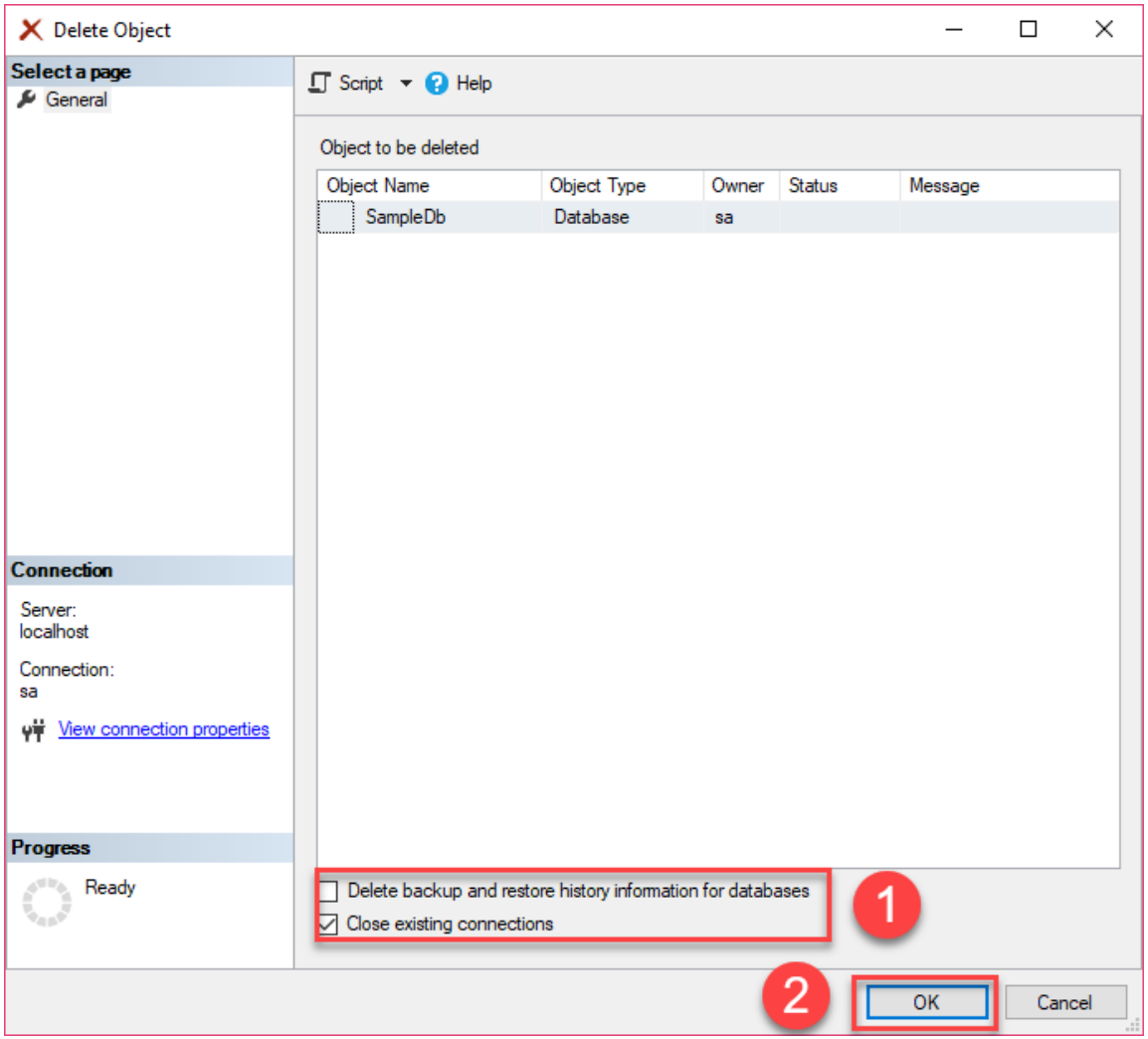
```
DROP DATABASE IF EXISTS TestDb;
```

Bir veri tabanını silmek için SQL Server Management Studio'yu kullanma

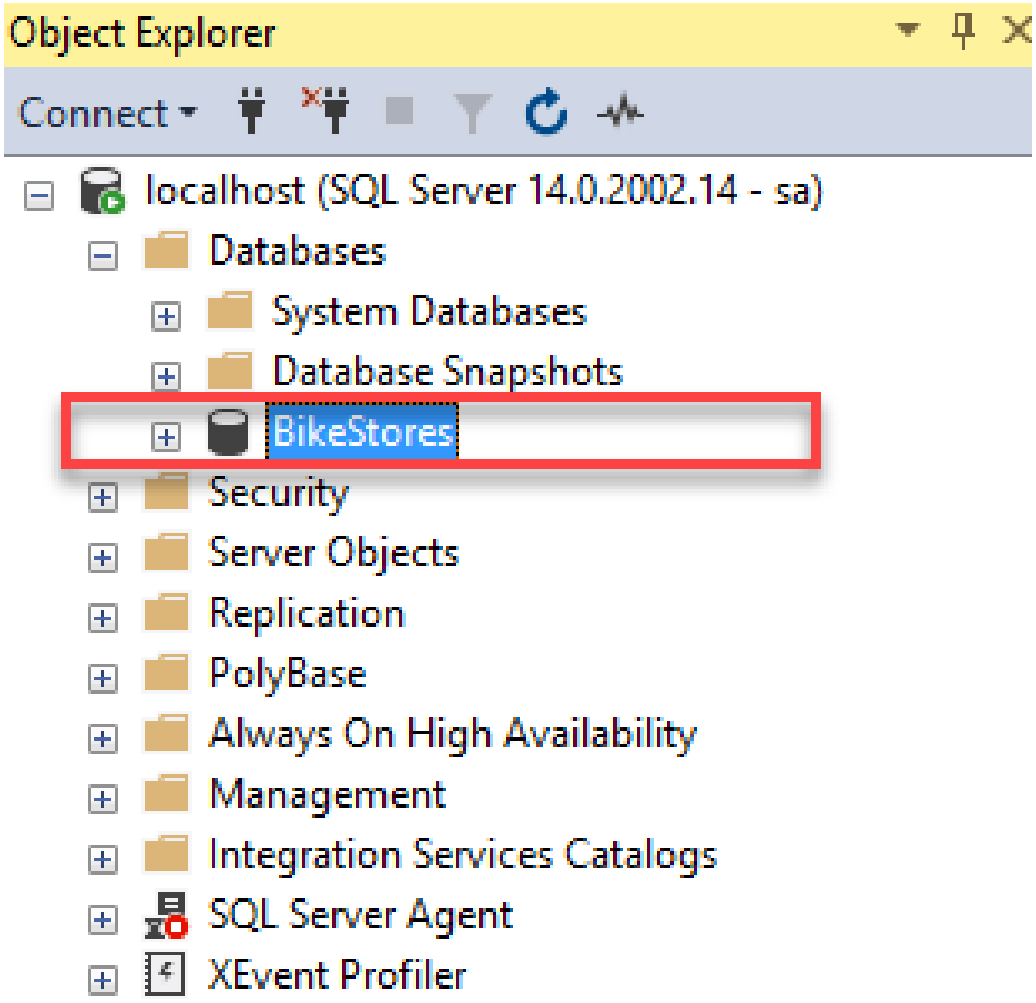
- SampleDb veri tabanını silmek için şu adımları izleyebilirsiniz:
  - Öncelikle silmek istediğiniz veri tabanı ismine sağ tıklayıp **Sil** menü öğesini seçin:



- İkinci olarak, **Veri tabanları için yedekleme ve geri yükleme geçmişi bilgilerini sil** onay kutusunun işaretini kaldırın, **Mevcut bağlantıları kapat** onay kutusunu işaretleyin ve veritabanını silmek için **Tamam düğmesine tıklayın.**



- Üçüncüsü, veri tabanının Nesne Gezgini'nden kaldırıldığını doğrulayın.



SQL Server ŞEMA OLUŞTURMA

SQL Server'da şema nedir?

- Bir şema, tablolar, view, trigger, stored procedure, dizinler vb. içeren veri tabanı nesnelerinin bir koleksiyonudur. Bir şema, mantıksal olarak ilişkili veri tabanı nesnelerinin sahibi olan şema sahibi olarak bilinen bir kullanıcı adıyla ilişkilendirilir.
- Bir şema her zaman bir veri tabanına aittir. Öte yandan, bir veri tabanının bir veya birden fazla şeması olabilir. Örneğin, BikeStores örnek veri tabanımızda iki şemamız var: sales(satışlar) ve production(üretim). Bir şema içindeki bir nesne, sales.orders gibi schema_name.object_name biçimi kullanılarak nitelendirilir. İki şemadaki iki tablo aynı adı paylaşabilir, bu nedenle hr.employees ve sales.employees olabilir.

SQL Server'da yerleşik şemalar

- SQL Server bize, yerleşik veri tabanı kullanıcıları ve rolleriyle aynı adlara sahip bazı önceden tanımlanmış şemalar sağlar; örneğin: dbo, guest, sys ve INFORMATION_SCHEMA.
- SQL Server'ın sys ve INFORMATION_SCHEMA şemalarını sistem nesneleri için ayırdığını, bu nedenle bu şemalarda herhangi bir nesne oluşturamayacağınızı veya bırakamayacağınızı unutmayın.
- Yeni oluşturulan bir veri tabanı için varsayılan şema, dbo kullanıcı hesabına ait olan dbo'dur. Varsayılan olarak, CREATE USER komutuyla yeni bir kullanıcı oluşturduğunuzda, kullanıcı varsayılan şeması olarak dbo'yu alır.

SQL Server CREATE SCHEMA ifadesine genel bakış

- CREATE SCHEMA ifadesi, mevcut veri tabanında yeni bir şema oluşturmanıza olanak tanır.
- Aşağıda CREATE SCHEMA ifadesinin basitleştirilmiş hali gösterilmektedir:

```
CREATE SCHEMA schema_name
    [AUTHORIZATION owner_name]
```

- Bu sözdiziminde,
  - Öncelikle, CREATE SCHEMA ifadesinde oluşturmak istediğiniz şemanın adını belirtin.
  - İkinci olarak, AUTHORIZATION anahtar sözcüğünden sonra şemanın sahibini belirtin.

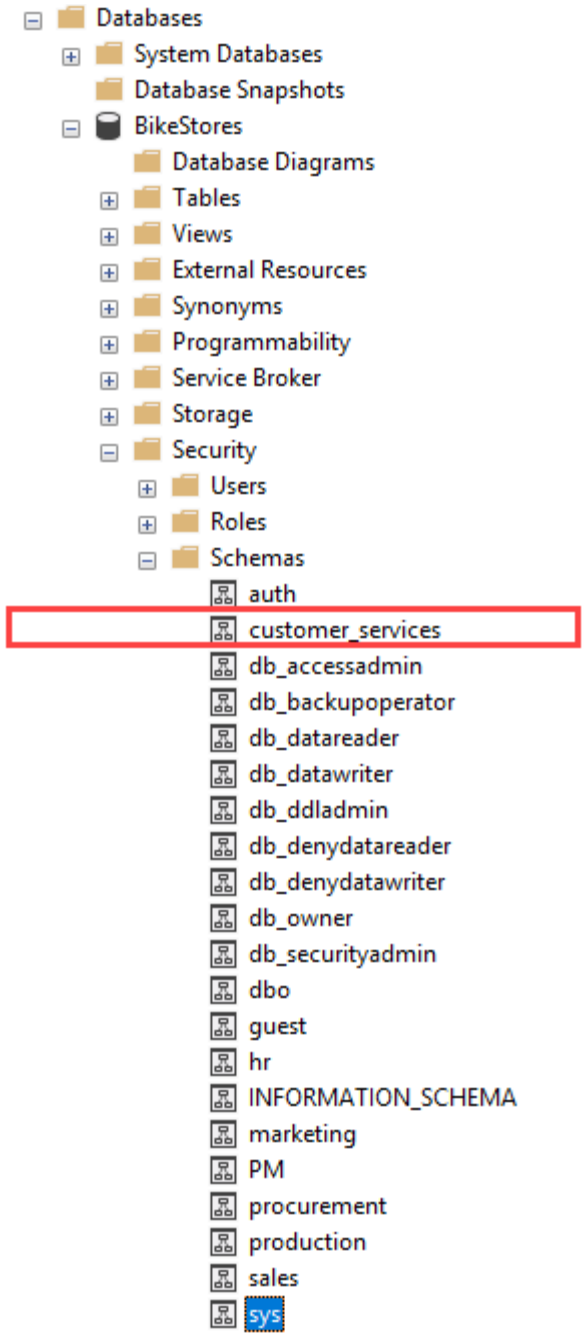
SQL Server CREATE SCHEMA ifadesi örneği

- Aşağıdaki örnek, customer_services şemasını oluşturmak için CREATE SCHEMA ifadesinin nasıl kullanılacağını gösterir:

```
CREATE SCHEMA customer_services;
GO
```

- GO komutunun SQL Server Management Studio'ya GO ifadesine kadar olan SQL ifadelerini yürütülmek üzere sunucuya göndermesini söylediğini unutmayın.

- İfadeyi yürüttüğünüzde, yeni oluşturulan şemayı veritabanı adının **Güvenlik > Şemalar (Security > Schemas)** bölümünde bulabilirsiniz.



- Mevcut veri tabanındaki tüm şemaları listelemek istiyorsanız, aşağıdaki sorguda gösterildiği gibi sys.schemas'tan şemaları sorgulayabilirsiniz:

```
SELECT
    s.name AS schema_name,
    u.name AS schema_owner
FROM
    sys.schemas s
INNER JOIN sys.sysusers u ON u.uid = s.principal_id
ORDER BY
    s.name;
```

- Çıktı şu şekilde:

schema_name	schema_owner
auth	dbo
customer_services	dbo
db_accessadmin	db_accessadmin
db_backupoperator	db_backupoperator
db_datareader	db_datareader
db_datawriter	db_datawriter
db_ddladmin	db_ddladmin
db_denydatareader	db_denydatareader
db_denydatawriter	db_denydatawriter
db_owner	db_owner
db_securityadmin	db_securityadmin
dbo	dbo
guest	guest
hr	dbo
INFORMATION_SCHEMA	INFORMATION_SCHEMA
marketing	dbo
PM	dbo
procurement	dbo
production	dbo
sales	dbo
sys	sys

- customer_services şemasına sahip olduktan sonra, şema için nesneler oluşturabilirsiniz. Örneğin, aşağıdaki ifade customer_services şemasında jobs adlı yeni bir tablo oluşturur:

```
CREATE TABLE customer_services.jobs(
    job_id INT PRIMARY KEY IDENTITY,
    customer_id INT NOT NULL,
    description VARCHAR(200),
    created_at DATETIME2 NOT NULL
);
```

SQL Server ŞEMA DEĞİŞTİRME

SQL Server ALTER SCHEMA ifadesine genel bakış

- ALTER SCHEMA ifadesi, aynı veri tabanındaki bir şemadan diğerine güvenli bir veri aktarmanıza olanak tanır.

- Aşağıda ALTER SCHEMA ifadesinin sözdizimi gösterilmektedir:

```
ALTER SCHEMA target_schema_name
    TRANSFER [ entity_type :: ] securable_name;
```

```
ALTER SCHEMA target_schema_name
    TRANSFER [ entity_type :: ] securable_name;
```

- Bu sözdiziminde:
  - target_schema_name, nesneyi taşımak istediğiniz geçerli veri tabanındaki bir şemanın adıdır. SYS veya INFORMATION_SCHEMA olamayacağını unutmayın.
  - Entity_type, Object, Type veya XML Schema Collection olabilir. Varsayılanı Object'tir. Entity_type, sahibi değiştirilen varlığın sınıfını temsil eder.
  - Object_name, target_schema_name'e taşımak istediğiniz güvenli öğenin adıdır.

- Bir stored procedure'ü, işlevi, view'ı veya trigger'ı taşırsanız, SQL Server bu güvenli hale getirilebilir öğelerin şema adını değiştirmez. Bu nedenle, taşıma için ALTER SCHEMA ifadesini kullanmak yerine bu nesneleri yeni şemada bırakıp yeniden oluşturmanız önerilir.

- Bir nesneyi (örneğin tablo veya eşanlamlı) taşırsanız, SQL Server bu nesneler için başvuruları otomatik olarak güncellemez. Başvuruları yeni şema adını yansıtacak şekilde manuel olarak değiştirmeniz gerekir.

- Örneğin, stored procedure'de başvurulan bir tabloyu taşırsanız, stored procedure'ü yeni şema adını yansıtacak şekilde değiştirmeniz gerekir.

SQL Server ALTER SCHEMA ifadesi örneği

- Öncelikle dbo şemasında offices adında yeni bir tablo oluşturun:

```
CREATE TABLE dbo.offices
(
    office_id      INT
    PRIMARY KEY IDENTITY,
    office_name    NVARCHAR(40) NOT NULL,
    office_address NVARCHAR(255) NOT NULL,
    phone          VARCHAR(20),
);
```

- Daha sonra dbo.offices tablosuna birkaç satır ekleyelim:

```
INSERT INTO
    dbo.offices(office_name, office_address)
VALUES
    ('Silicon Valley','400 North 1st Street, San Jose, CA 95130'),
    ('Sacramento','1070 River Dr., Sacramento, CA 95820');
```

- Daha sonra ofisi bir ID'ye göre bulan bir stored procedure oluşturun:

```
CREATE PROC usp_get_office_by_id(
    @id INT
) AS
BEGIN
    SELECT
        *
    FROM
        dbo.offices
    WHERE
        office_id = @id;
END;
```

- Daha sonra bu dbo.offices tablosunu sales(satış) şemasına aktarın:

```
ALTER SCHEMA sales TRANSFER OBJECT::dbo.offices;
```

- usp_get_office_by_id stored procedure'ü çalıştırırsanız:

```
exec usp_get_office_by_id @id=1;
```

- SQL Server bir hata verecektir:

```
Msg 208, Level 16, State 1, Procedure usp_get_office_by_id, Line 5 [Batch
Invalid object name 'dbo.offices'.
```

- Son olarak, yeni şemayı yansıtmak için stored procedure içindeki dbo.offices'i manuel olarak sales.offices'e değiştirin:

```
CREATE PROC usp_get_office_by_id(
    @id INT
) AS
BEGIN
    SELECT
        *
    FROM
        sales.offices
    WHERE
        office_id = @id;
END;
```

SQL Server TABLO OLUŞTURMA

SQL Server CREATE TABLE ifadesine giriş

- Tablolar, veri tabanında veri depolamak için kullanılır. Tablolar, bir veri tabanı ve şema içinde benzersiz şekilde adlandırılır . Her tablo bir veya daha fazla sütun içerir. Ve her sütun, depolayabileceği veri türünü tanımlayan ilişkili bir veri türüne sahiptir, örneğin sayılar, dizeler veya zamansal veriler.
- Yeni bir tablo oluşturmak için CREATE TABLE ifadesini aşağıdaki şekilde kullanabilirsiniz:

```
CREATE TABLE [database_name.][schema_name.]table_name (
    pk_column data_type PRIMARY KEY,
    column_1 data_type NOT NULL,
    column_2 data_type,
    ...,
    table_constraints
);
```

```
CREATE TABLE [database_name.][schema_name.]table_name (
    pk_column data_type PRIMARY KEY,
    column_1 data_type NOT NULL,
    column_2 data_type,
    ...,
    table_constraints
);
```

- Bu sözdiziminde:
  - Öncelikle, tablonun oluşturulduğu veri tabanının adını belirtin. database_name, mevcut bir veri tabanının adı olmalıdır. Bunu belirtmezseniz, database_name varsayılan olarak geçerli veri tabanına ayarlanır.
  - İkincisi, yeni tablonun ait olduğu şemayı belirtin.
  - Üçüncüsü, yeni tablonun adını belirtin.

```
CREATE TABLE [database_name.][schema_name.]table_name (
    pk_column data_type PRIMARY KEY,
    column_1 data_type NOT NULL,
    column_2 data_type,
    ...,
    table_constraints
);
```

- Dördüncüsü, her tablonun bir veya daha fazla sütundan oluşan bir birincil anahtarı olmalıdır. Genellikle, önce birincil anahtar sütunlarını ve ardından diğer sütunları listelersiniz. Birincil anahtar yalnızca bir sütun içeriyorsa, sütun adından sonra PRIMARY KEY anahtar sözcüklerini kullanabilirsiniz. Birincil anahtar iki veya daha fazla sütundan oluşuyorsa, PRIMARY KEY kısıtlamasını bir tablo kısıtlaması olarak belirtmeniz gerekir. Her sütunun, ifadede adından sonra belirtilen ilişkili bir veri türü vardır. Bir sütunda NOT NULL ve UNIQUE gibi bir veya daha fazla sütun kısıtlaması olabilir.
- Beşinci olarak, bir tabloda FOREIGN KEY, PRIMARY KEY, UNIQUE ve CHECK gibi tablo kısıtlamaları bölümünde belirtilen bazı kısıtlamalar olabilir.

SQL Server CREATE TABLE örneği

- Aşağıdaki ifade, müşterilerin mağaza içi ziyaretlerini takip etmek için sales.visits adında yeni bir tablo oluşturur:

```
CREATE TABLE sales.visits (
    visit_id INT PRIMARY KEY IDENTITY (1, 1),
    first_name VARCHAR (50) NOT NULL,
    last_name VARCHAR (50) NOT NULL,
    visited_at DATETIME,
    phone VARCHAR(20),
    store_id INT NOT NULL,
    FOREIGN KEY (store_id) REFERENCES sales.stores (store_id)
);
```

- Bu örnekte:
  - Tablonun oluşturulduğu veri tabanının adını açıkça belirtmediğimiz için, visits tablosu BikeStores veri tabanında oluşturulur. Şema için, bunu açıkça belirtiriz, bu nedenle visits tablosu sales şemasında oluşturulur.

```
CREATE TABLE sales.visits (
    visit_id INT PRIMARY KEY IDENTITY (1, 1),
    first_name VARCHAR (50) NOT NULL,
    last_name VARCHAR (50) NOT NULL,
    visited_at DATETIME,
    phone VARCHAR(20),
    store_id INT NOT NULL,
    FOREIGN KEY (store_id) REFERENCES sales.stores (store_id)
);
```

- Visits tablosu altı sütundan oluşur:

- Visit_id sütunu tablonun birincil anahtar sütunudur. IDENTITY(1,1), SQL Server'a sütun için birden başlayıp her yeni satır için birer birer artan tam sayı sayılarını otomatik olarak oluşturmasını söyler.
- First_name ve last_name sütunları, VARCHAR türünde karakter dizisi sütunlarıdır. Bu sütunlar 50 karaktere kadar depolayabilir.
- visited_at, müşterinin mağazayı ziyaret ettiği tarih ve saati kaydeden bir DATETIME sütunudur.
- Telefon sütunu, NULL kabul eden değişken bir karakter dizisi sütunudur.
- store_id sütunu, müşterinin ziyaret ettiği mağazayı tanımlayan kimlik numaralarını depolar.
- Tablonun tanımının sonunda bir FOREIGN KEY kısıtlaması bulunur. Bu yabancı anahtar, visits tablosunun store_id sütunundaki değerlerin stores tablosundaki store_id sütununda bulunmasını sağlar.