T-SQL'E GENEL BAKIŞ

2

Transact-SQL Kavramı

Transact-**SQL** (**T-SQL**), SQL Server'ın programsal alt yapısı için geliştirilen, ileri seviye veritabanı geliştirme ve yönetim özellikleri bulunan bir **sorgulama dili**dir.

T-SQL ile ilgili gerekli detaylı eğitim ve anlatımlar ileriki bölümlerde yapılmıştır.

Bu bölümde T-SQL ile ilgili temel ve en çok kullanılacak özellikler tüm detaylarıyla inceleyeceğiz.

Bu bölümde öğreneceğimiz T-SQL cümleleri şunlardır;

- **`SELECT`**
- **`INSERT`**
- **`UPDATE`**
- **`DELETE`**

Bu dört sorgu ifadesi T-SQL sorgu dilinin en temel sorgu özellikleridir. Veri İşleme Dili (***DML = Data Manuplation Language***) olarak bilinen bu sorgu ifadeleri, veritabanında veri seçme, veri ekleme, veri güncelleme, veri silme gibi temel işlemleri gerçekleştirir. Bir veritabanı uygulamasında en çok kullanılan ve uygulamanın **veritabanı uygulaması** olmasını sağlayan T-SQL ifadeleridir.

T-SQL sorgulama dili, SQL Server'ın tanıdığı ve sorgulamaları kendi veritabanı motorunda diğer RDBMS veritabanı motorları tarafından tanınamayacak bir dildir. Ancak bununla birlikte T-SQL'in SQL standartlarını da desteklediğini belirtmiştik. SQL Server giriş seviye ANSI SQL-92 standartlarını destekler. ANSI standartlarındaki SQL çoğu durumda T-SQL'e göre daha performanslı olacaktır. Eğer geliştirdiğiniz veritabanı uygulamasının her RDBMS için geçerli ve performanslı olmasını isterseniz, kullanmanız gereken sorgu dili T-SQL değil, SQL standartları olmalıdır. Yani, performans ve RDBMS platformlarından bağımsız bir veritabanı oluşturmak için mümkün olduğunda ANSI standartlarına uyumlu olmalısınız.

T-SQL İLE İLGİLİ KURALLAR

NESNE VE DEĞİŞKEN İSİMLENDİRME KURALLARI

T-SQL'de tablo, sütun, index vb. diğer tüm nesnelerin isimleri belirlerken uymamız gereken bazı standartlar var. Bu standartların sebebi diğer programlama dillerinde olduğu gibi çeşitli hatalara karşı uygulamamızı korumak ve programlamayı daha düzenli ve geliştiriciler için anlaşılabilir şekilde hazırlamak.

- Harf ([a-z] ya da [A-Z]) ile ya da alt çizgi (_) ile başlamalıdır.
- Tanımlayıcı isimleri @, @@, #, ##, $ gibi özel karakterler ile başlamamalıdır.

Bu özel karakterler sistem tarafından şu amaçlar için ayrılmıştır;

- **@ :** Değişken tanımlarken değişken adının başında kullanılır.
- **@@ :** Sistem değişkenlerinin isimlerinin başına gelir.
- **# ve ## :** Geçici nesne belirteci olarak kullanılır.

- SQL ve T-SQL mimarisinde tanımlanmış (`SELECT`, `INSERT`, `UPDATE` gibi) kelimeler kullanılmamalıdır.
- Tanımlayıcı isimlerinde Türkçe harfler (`Ü`, `ü`, `Ç`, `ç`, `Ö`, `ö`, `İ`, `ı`, `Ğ`, `ğ`, `Ş`, `ş`) ve boşluk yer almamalıdır.

Bu kurallara uymayan bir tanımlayıcı kullanılacak ise tanımlayıcı isimleri köşeli parantez (`[]`) ya da (`"`) çift tırnak içerisinde yazılması gerekir.

Çift tırnak kullanımı için gerekli açıklama **SQL Server Veritabanı Nesneleri** bölümündeki **İsimlendirme Kuralları** konusunda anlatılmıştır.

Tanımlayıcı İsimlendirme Notasyonları

Notasyon Kavramı

Tüm programlama dillerinde değişken ve nesne isimlendirmelerinde bazı kısıtlamalar mevcuttur. Tavsiye edilenlerden bazıları boşluk bırakmamak ve çoğu karakterin kullanılamamasıdır. Bu durumda otomatikman isimlendirilen nesnelerin anlaşılması konusunda da bazı kısıtlamalar getirilmiş oluyor.

Bu kısıtlamalar, notasyon kavramı ile bir nebze dengelenmiştir. Notasyonlar tanımlama işlemlerinde bazı kurallar ve düzen getirmeyi hedefler. Tanımlama işlemini büyük ya da küçük harf ile başlatmak, kelimeler arasına alt çizgi koymak gibi sıralanabilir. Birçok farklı notasyondan kabul gören bazıları aşağıdaki gibidir.

Camel Notasyonu

İlk kelimenin tamamen küçük harf, geri kalan tüm kelimelerin baş harflerinin büyük şekilde yazılmasıdır.

Örnek: `ileriSeviyeSqlServerTsql`, `kodLab`, `dijibilCom`

Pascal Notasyonu (Deve Notasyonu)

Kelimelerin ilk harflerinin büyük diğer harflerin küçük yazılmasıdır.

Örnek: `İleriSeviyeSqlServerTsql`, `KodLab`, `DijibilCom`

Alt Çizgi (Underscore) Notasyonu

Kelimelerin arasına altçizgi koyulmasıdır.

Örnek: `İleri_Seviye_Sql_Server_Tsql`, `Kod_Lab`, `Dijibil_Com`

Büyük Harf (Uppercase) Notasyonu

İsimlendirmenin büyük harflerle yapılmasıdır. Tüm notasyonlarda olduğu gibi anlamayı kolaylaştırmak için bu notasyon ile diğer bir notasyon olan alt çizgi kullanımı gerçekleştirilebilir.

Örnek: `İLERİSEVİYESQLSERVERTSQL`, `KODLAB`, `DIJIBILCOM`

Alt Çizgili: `İLERİ_SEVİYE_SQL_SERVER_TSQL`, `KOD_LAB`, `DIJIBIL_COM`

Macar Notasyonu

Pascal notasyonuna ek olarak isimlendirmenin başına, kullanılan değişken tipinin veya kontrolün kısaltma ile yazılmasıdır.

Örnek: **`tblKODLAB`**, **`spDIJIBILCOM (sp = stored procedure, tbl = tablo)`**

Açıklama Satırları

Tüm programlama dilleri ve veritabanı yazılımlarında var olan ve geliştiricilerin kod blokları arasında açıklamalar, bilgi ve hatırlatıcı metinler girebilmesini sağlayan bir özelliktir. Bu açıklama satırı, tek satır ve çok satırlı olabilmektedir.

Açıklama satırı için iki farklı yöntem vardır. Bunlar iki tane yan yana tire işareti (`--`) ve bu şekilde (`/* */`) kullanımıdır.

Açıklamanız tek satır ise;

```
-- 'CREATE TABLE kodlab' komutu ile kodlab adında bir tablo oluşturun.
```

Açıklamanız çok satırlı ise;

```
/*
  'CREATE TABLE dijibil' komutu ile
  dijibil adında bir tablo oluşturun.
*/
```

Açıklama haline getirilen satırlar otomatik olarak açık yeşil ile renklendirilir ve kodların veritabanı motoruna gönderilmesi durumunda veritabanı motoru buradaki metinleri kod olarak görmediği için çalıştırmaz.

NULL Kavramı

Bilgisayar teriminde boşluk ile hiçlik (**`NULL`**) aynı değildir. Veritabanında ilgili bir kayıt için klavyeden boşluk (**`space`**) tuşu kullanılarak değer verilebilir. Bu o kaydın olmadığı anlamına gelmez. Bu boş kaydın karşılığı ASCII'de (ASCII-32) vardır. Ancak boşluk dahil hiç bir değer girilmemiş, işlem yapılmamış kayıtlar **`NULL`** olarak nitelendirilir. **`ANSI`** Standardında değeri bilinmediği için **`NULL`** değerler, **`NULL`** değer kontrolü yani kaydın **`NULL`** olup olmadığı hariç, herhangi bir işleme tabi tutulamazlar. İki **`NULL`** değerin birbirine eşit olup olmadığını kontrol etmek gibi bir işlem yapılamaz.

T-SQL'de Yığın Kavramı

SQL Server'da yığın kavramı, sorguların sırayla işleme alınmasını ifade eder. Çalışma sırasında SQL Server'a birden fazla gönderilen sorgu yığınlar halinde ele alınır.

GO Komutu

GO komutu bir yığının sonunu belirtmek için kullanılır. Bir yığın SQL Server'da işlenmeye başladığı anda önce ayrıştırılır (*parse*), sonra derlenir (*compile*) ve son olarak sorgu çalıştırılır (*execute*).

Kullanımı:

```
Komutlar
GO
```

USE Komutu

T-SQL kod bloklarının, hangi veritabanı için çalıştırılacağını belirtmek için kullanılır.

Kullanımı:

```
USE
Veritabani_Adi
```

PRINT Komutu

T-SQL içerisinde kullanılan değişkenlerin değerleri ya da hatalar sonucu oluşacak hata mesajları gibi sonuçların sorgu çalıştığı anda gösterilmesini sağlar.

Kullanımı:

```
PRINT gosterilecek_deger
```

Örnek 1:

```
PRINT 'print edilecek değer'        -- Sıradan bir metinsel ifade de olabilir.
```

Örnek 2 :

```
PRINT @degisken_adi
```

Örnek 3:

```
USE kodlab
GO
DECLARE @kitap VARCHAR(29)
SET @kitap = 'İleri Seviye SQL Server T-SQL'
GO
PRINT @kitap
GO
```

Kitap değişkenine, boşluklar dahil 29 karakterlik veri girildiği için **VARCHAR(29)** veri boyutu atanmıştır.

Veri Tanımlama Dili

Veri Tanımlama Dili (***DDL*** = ***D****ata* ***D****efinition* ***L****anguage*), SQL Server'da veritabanı nesneleri üzerinde yönetim işlemleri gerçekleştirmek için kullanılır. **CREATE** ile veritabanı, tablo, index, trigger, stored procedure ya da bir başka veritabanı nesnesi oluşturabilir. **ALTER** ile daha önce oluşturulmuş bir nesne üzerinde yapısal değiştirme işlemi gerçekleştirilir. **DROP** ile de mevcut bir veritabanı nesnesi silinir.

Kısaca özetlemek gerekirse;

CREATE	Herhangi bir veritabanı nesnesi oluşturur.
ALTER	Mevcut bir nesneyi yapısal olarak değiştirir.
DROP	Mevcut bir nesneyi veritabanından siler.

SQL Server'da Nesne İsimleri

SQL Server Management Studio'da sorgu hazırlarken görünen araçları kullanarak giriş sırasında server seçmek ve bağlanmak, sorgu ekranında veritabanı seçimi gibi kolaylıklarından dolayı önemli bir nokta olan nesne isimlendirmelerini özellikle incelememiz mimariyi anlamak adına faydalı olacaktır.

SQL Server, bize kolay kullanım ve çeşitli araçlar sunar. Fakat arka planda aslında her şey nesneler şeklinde birbirine bağlı olarak seçilir ve ilişkilendirilir. Soldaki Tree (*ağaç*) yapısından bir veritabanı seçersiniz, sonra bu veritabanı içerisinde tablolar sekmesini açar ve işlem yapacağınız tabloyu seçerek istediğiniz işlemi yaparsınız.

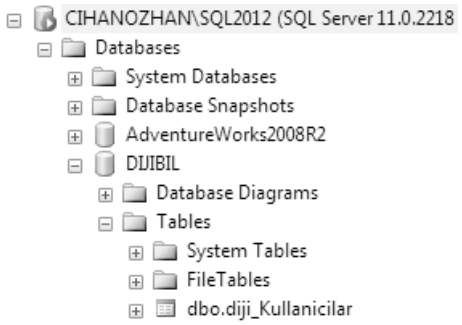
Programsal olarak bu yapı veritabanında şu şekilde ilişkilendirilerek yönetilir.

```
[server_ismi.[veritabani_ismi.[sema_ismi]]].nesne_ismi
```

Bir **sorgu ekranı** (*Query Window*) açtıktan sonra, üst kısımdaki **Available Databases** kısmında işlem yapacağınız veritabanı seçili ise, sadece nesne adını belirterek işlem yapabilirsiniz. Bu ilişkilendirme olayı SSMS ile şu şekilde gerçekleştirilir.

- SSMS açılırken **Server Name** kısmında bağlanacağınız server'ı seçerek **Connect** butonu ile bağlanılır.
- Sol **Tree** yapısında bağlanılan server içerisindeki **Databases** sekmesinden veritabanı sekmesi genişletilir.
- Seçilen veritabanı içerisinde **Tables** bölümünden işlem yapacağınız tabloyu seçebilirsiniz.

```
CIHANOZHAN\SQL2012.DIJIBIL.dbo.diji_Kullanicilar;
```



Bu sıralamanın doğruluğundan emin olmak için kendi server, veritabanı ve nesne adınıza göre aşağıdaki gibi sorgu hazırlayıp çalıştırabilirsiniz.

```
SELECT * FROM [CIHANOZHAN\SQL2012].DIJIBIL.dbo.diji_Kullanicilar;
```

Sorgumuzda **`server_ismi`** değerim olan **`CIHANOZHAN\SQL2012`** değerini kare parantez (**`[]`**) içerisine almamın sebebi; sorgunun veritabanı motoruna doğru şekilde gönderilmesini ve yazım da kullanılan karakterin hata vermeden çalışmasını sağlamak içindir. Kare parantez ile ilgili detaylı açıklamaları, nesne isimlendirme kuralları konusunda yapmıştık.

Schema İsmi (Ownership / Sahiplik)

Schema (*şema*), ANSI standartlarında var olan bir standarttır. SQL Server'da schema özelliği eski sürümlerden beri desteklenmektedir. Fakat bu isimlendirme ownership, owner gibi farklı şekillerdeydi. SQL Server sahiplik anlamına gelen ve aynı hizmet için var olan bu özelliğin adını ANSI standartlarına uyumlu hale getirerek **schema** adını vermiştir. Schema kavramı bir ANSI standardı olmakla birlikte, sadece SQL Server'da değil, Oracle gibi diğer veritabanlarında da bu şekilde ve bu amaç için kullanılır.

Önceki sürümlerde SQL Server'da, sahip nesneyi oluşturan ya da veritabanı sahibiydi. Fakat yeni versiyonlarla birlikte nesne bir sahipten çok bir schema'ya atanır. Sahiplik belirli bir kullanıcıya bağlı iken, schema birden fazla kullanıcı tarafından paylaşılabilir ya da bir kullanıcı birden fazla schema hakkında sahip olabilir.

Varsayılan Schema: dbo

SQL Server'da veritabanını oluşturan kullanıcı, veritabanının sahibi ya da dbo olarak tanımlanır. Veritabanı sahibi olan kullanıcı tarafından oluşturulan bir nesne, kullanıcının kullanıcı ismi schema'sında değil, dbo schema içinde listelenecektir.

Veritabanı sahibi olmayan bir kullanıcı için şu şekilde örneklendirilebilir.

`KodLab` adında bir veritabanımız olduğunu ve bu veritabanında veritabanı sahibi olmayan, **`Cihan`** adında bir kullanıcı olduğumu düşünelim. Bu veritabanında **`CREATE TABLE`**, yani tablo oluşturma yetkisine sahibim. **`KodLab`** veritabanında **`Kitaplar`** adında bir tablo oluştursam, bu tablonun sahibiyle isimlendirilmiş nesne ismi **`Cihan.Kitaplar`** şeklinde olacaktır. Bu kullanım ile **`Cihan`** kullanıcısı, **`KodLab`** veritabanındaki **`Kitaplar`** tablosunun özel bir sahibi olduğundan, bir başka kullanıcı bu tabloya erişmek istediğinde şema ismi farklı olacağı için **`Cihan.Kitaplar`** şeklinde sahibi nitelendirilmiş nesne ismini vermesi gerekir. Bu durumda sahibi nitelendirilmiş nesne ismimiz **`Cihan.Kitaplar`** olacaktır. Eğer bu tabloyu veritabanı sahibi olan bir kullanıcı oluşturmuş olsaydı bu tablo, varsayılan şema olan dbo şeması içerisinde oluşturulacağı için schema belirtmeye gerek kalmayacak ve **`Kitaplar`** şeklinde nesneye ulaşılabilecekti. Veritabanı sahibi tarafından oluşturulan nesnelere **`dbo.nesne_ismi`** ya da sadece **`nesne_ismi`** şeklinde ulaşılabilmektedir.

sa (*system administrator*) ya da **sysadmin** rolü üyelerini daima dbo takma adını kullanır. **sa**'lar aslında veritabanının gerçek sahibi değildir. Fakat yetki olarak veritabanı sahibinin (dbo) yetkilerine sahiptirler ve **sa** kullanıcılarının oluşturacağı nesnelerin sahibi dbo olacaktır. **db_owner** veritabanı rolüne ait kullanıcılar tarafından oluşturulan nesneler için ise; varsayılan schema dbo değil, kullanıcının varsayılan schema'sı olacaktır.

Veritabanı İsmi

Veritabanında nesne isimlendirmelerinden bahsederken **schema_ismi** bilgisinden sonra **veritabani_ismi** bilgisini belirmemiz gerektiğini öğrenmiştik. Veritabanı ismi nesne isimlendirmeleri ve ilişkilendirmelerinde önemli bir yere sahiptir.

Bir tane **DIJIBIL**, bir de **KodLab** adında veritabanımız olsun. Biz **DIJIBIL** veritabanında işlem yapmak için oturum açtığımızı ve işlem sırasında **KodLab** veritabanındaki Kitaplar tablosundan **DIJIBIL** veritabanındaki **diji_Kitaplar** tablosu arasında **JOIN** ile ilişkilendirme yapacağımızı ya da sıradan bir **SELECT** ile diğer veritabanına geçmeden diğer veritabanındaki bir tabloyu sorgulamak ya da herhangi bir işlem ile iki ayrı veritabanındaki iki farklı tablo arasında işlem yapacağımızı varsayalım. Bu tablolar arasında işlem yapabilmemiz için veritabanlarının isimlerine ihtiyacımız vardır.

KodLab veritabanı ile oturum açtığımızda, **DIJIBIL** veritabanına ait bir tabloya sorgulama yapmak için;

```
SELECT * FROM DIJIBIL.dbo.diji_Kullanicilar;
```

dbo varsayılan bir schema olduğu için **dbo** yazmadan da şu şekilde çalışılabilir;

```
SELECT * FROM DIJIBIL..diji_Kullanicilar;
```

SSMS ile oturum açtığınız ve sorgu penceresi (Query Window) oluşturduğunuzda size geçerli veritabanına sorgu yapmaya hazır bir ekran gelir. Bu kısımda gerçekleştirdiğiniz sorgular her zaman geçerli veritabanında sorgulanır. Bu geçerli veritabanı sizin için varsayılan veritabanıdır.

SQL Server'da dbo varsayılan schema'dır. Bir kullanıcı, oluşturduğu nesne için schema belirtmemişse bu nesne SQL Server tarafından ilk olarak dbo nesnesinde aranır. Bu nedenle, dbo içerisinde olmayan bir nesne belirttiğinize eminseniz performans açısından SQL Server'ın dbo içerisindeki nesneleri araması sırasındaki kaybı önlemek için schema adını belirtmenizde fayda vardır.

Server Tarafından İsimlendirme

SQL Server, içerisinde sadece veritabanları arasında bağlanma ve sorgular geliştirmeye değil, aynı zamanda farklı server'lar arasında da sorgular geliştirmeye izin verir. Bu server'lar sadece SQL Server değil, Oracle, DB2 ya da benzeri büyük veritabanlarını da destekler.

Bağlantılı server'lar (*Linked Servers*), geliştiriciden çok veritabanı yöneticilerinin görev alanına girer. Bu nedenle bu özelliğin nasıl isimlendirildiğini öğrenerek devam edeceğiz.

Bir server ile bir başka server arasındaki bağlantı ile isimlendirme şu şekilde gerçekleşir;

```
sunucu_ismi.veritabani_ismi.sema_ismi.nesne_ismi
```

CREATE ile Nesne Oluşturmak

CREATE ifadesi ile veritabanındaki nesnelerden herhangi biri oluşturulabilir. **CREATE** ifadesi genel bir oluşturucu ifadedir. **CREATE** ifadesinden sonra oluşturulacak nesnenin özelliklerine göre farklı parametreler alır.

```
CREATE nesne_adi
```

CREATE DATABASE ile Veritabanı Oluşturmak

SQL Server'da bir veritabanı oluşturmanın en temel hali şu şekildedir;

```
CREATE DATABASE veritabani_adi
```

Bize şablon görevi gören model veritabanımızdaki varsayılan ayarlar ile bir veritabanı oluşturmak bu kadar kolaydır. Ancak SQL Server'da tüm detaylarıyla

hakim olmanızı önerdiğim konulardan biri de, T-SQL kullanarak veritabanı oluşturmak ve bu kullanım sırasındaki karışık bir çok parametrenin ne anlam ifade ettiğini öğrenmenizdir. İleri seviye veritabanı programlama yapmak için veritabanını oluşturan T-SQL yapısını iyi kavramanız gerekir. Bir veritabanı için varsayılan ayarların ne olduğunu **SQL Server Veritabanı Nesneleri** bölümündeki `model` konusunda anlatmıştık.

SQL Server'da yeni bir nesne oluşturulduğunda, bu nesneyi oluşturan kişi sistem yöneticisi ya da veritabanı sahibi değil ise, nesneyi oluşturan kişi haricinde kimse bu nesneye erişemez. Tabi ki bu oluşturulan nesnenin gerekli erişim ayarlarını daha sonra da ayarlayarak değiştirebilirsiniz.

T-SQL ile veritabanı oluşturmak için Microsoft tarafından hazırlanan söz dizimi ve kuralları içeren script çok uzun olduğu için burada yayınlamıyorum. Ancak dilerseniz aşağıdaki bağlantıdan ulaşabilirsiniz.

http://msdn.microsoft.com/en-us/library/ms176061.aspx

Yukarıda verdiğim script dosyası ya da bağlantıdan edindiğiniz kodların önemli olan bazı özellikleri tek tek detaylarıyla inceleyelim.

ON

Veritabanının veri ve log dosyasının yerini belirtmek için kullanılır. `ON` operatöründen sonra gelen `PRIMARY` anahtar kelimesini fark etmiş olmalısınız. Bunun anlamı, belirtilen dosya bilgilerinin fiziksel olarak birincil dosya grubuna ait olduğudur.

NAME

Veritabanının mantıksal ismini belirtir. SQL Server veritabanı dosyasına ulaşmak için bu ismi kullanır. Veritabanını ya da dosyayı yeniden boyutlandırmak gibi işlemler için bu isim kullanılır.

FILENAME

Veri ve log dosyasının bulunduğu dosyaların işletim sistemi üzerindeki gerçek fiziksel dosya isimlerini belirtir. İsimlendirme veri ve log dosyası için farklı şekilde yapılır.

Veri dosyası uzantısı: **mdf**

İkincil dosya uzantısı: **ndf**

Log dosyası uzantısı: **ldf**

Veri ve log dosyası için varsayılan dosya yolu;

Program Files(x86)\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA

Bu dosya yolundaki `DATA` içerisinde `DIJIBIL` adındaki bir veritabanının veri ve log dosyaları varsayılan olarak şu şekilde isimlendirilir;

Veri Dosyası: **DIJIBIL.mdf**

Log Dosyası: **DIJIBIL_log.ldf**

Bilgisayarınız 64 Bit destekli ise, kurulumunu yaptığınız SQL Server'a göre Program Files (x86) klasörü içerisinde bulunabilir. Örneğin, benim bilgisayarımda SQL Server 2008 ve SQL Server 2012 ayrı ayrı kuruludur.

SQL Server 2012'nin kurulu olduğu dosya yolum ise;

Program Files\Microsoft SQL Server\MSSQL11.SQL2012

SQL2012 ismi, iki veritabanı Instance'ının farklı olması gerektiği için kurulum sırasında verdiğim bir isimdir.

Veri Terimleri Büyüklükleri

SQL Server, veriyi tüm bilgisayar sistemlerindeki standartlara göre depoladığı ve yönettiği için, veritabanı boyutlandırma ya da benzeri boyut hesaplamalarında aşağıdaki eşleştirmelere uymanız gerekir.

1 **B**it(**b**)

1 **B**yte(**B**) = 8 bit

1 **K**ilo**B**yte(**KB**) = 1024 Byte

1 **M**ega**B**yte(**MB**) = 1024 KiloByte

1 **G**iga**B**yte(**GB**) = 1024 MegaByte

1 **T**era**B**yte(**TB**) = 1024 GigaByte

1 **P**eta**B**yte(**PB**) = 1024 TeraByte

1 **E**xa**B**yte(**EB**) = 1024 PetaByte

1 **Z**etta**B**yte(**ZB**) = 1024 ExaByte

1 **Y**otta**B**yte(**YB**) = 1024 ZettaByte

Zettabyte Dünyada ki bütün verilerin toplamının ancak 1,8 Zettabyte olduğunun tahmin edildiğini ve 1 Zettabyte'ın 250 milyar adet DVD ya da 36 milyon yıllık HD video görüntüsüne denk geldiğini düşünürsek, şimdilik YottaByte'ı bilmesek de olur

SIZE

Veritabanının dosya boyutunu gösterir. Boyut varsayılan olarak **MB** (**M**ega**B**yte) cinsinden belirtilir. Ancak boyutu **KB** (**K**ilo**B**yte), **GB** (**G**iga**B**yte) ya da **TB** (**T**era**B**yte) cinsinden de belirtebilirsiniz.

`model` veritabanını anlatırken bahsettiğimiz gibi, dosya boyutu en az model veritabanı boyutu kadar olması gerekir. Dosya boyutu belirtilirken tam sayı cinsinden, yani ondalık bölüm olmadan belirtilmesi gerekir.

`SIZE` parametresiyle bir değer belirtmezseniz varsayılan olarak model veritabanı boyutu ile oluşturulacaktır.

MAXSIZE

Veritabanı dosya boyutunun genişleyebileceği en yüksek büyüklüğü belirtir. `MAXSIZE` parametresi de `SIZE` gibi varsayılan olarak MB cinsinden değer alır. Fakat dilerseniz KB, GB ya da TB cinsinden dosya boyutu belirtebilirsiniz. Bu parametre değerinin belirtilmesi zorunlu değildir. Eğer bir değer belirtilmezse varsayılan olarak bir `MAXSIZE` değeri atanmayacak ve bu durumda `MAXSIZE` değeri sabit disk (HDD)'in kapasitesiyle sınırlı olacaktır.

`MAXSIZE`, üzerinde sürekli anlık işlem yapıldığı durumlarda çok kritik ve takip edilmesi gereken bir özelliktir. Eğer dosya boyutları (mdf, ndf), `MAXSIZE`'ın maksimum değerine ulaşırsa, kullanıcılar veri girişi yapmak istediğinde hata alırlar. Log tutma işleminde de maksimum değere ulaşıldığında log tutulamayacaktır. Bir veritabanı veri dosyası (mdf), sadece veri giriş işlemlerinde bir girdi olduğu için hızlı büyümez. Ancak log dosyaları her işlem için log dosyasına (ldf) loglama işlemi gerçekleştirdiği için sürekli büyüyecektir.

Bir veritabanının `MAXSIZE`'ı verilmediği takdirde üst limit sabit diskin üst sınırı ile eş değerdir. Veritabanı sürekli genişleyecek ve sabit diski doldurana kadar herhangi bir sorun çıkarmayacaktır. Ancak sabit disk alanını da doldurduğunda, bu sadece bir veritabanı hatası değil, işletim sistemi dahil diğer tüm programların da çalışmamasına sebep olacaktır. Bu nedenle size tavsiyem, mutlaka bir `MAXSIZE` değeri ve `FILEGROWTH` değeri belirlemeniz yönünde olacaktır. Kontrolü SQL Server'a bırakmayın. Tüm ayar ve yönetimi kendiniz belirleyin.

FILEGROWTH

FILEGROWTH özelliği, veritabanı dosyasının bir seferde ne kadar genişleyeceğini(büyüyeceğini) belirtmek için kullanılır. Veritabanının başlangıç genişlik değeri **SIZE** ile olabilecek en yüksek değeri **MAXSIZE** arasında geçerlidir. Bu değeri KB, MB, GB ya da TB olarak belirleyebilirsiniz. Dilerseniz bu özelliği yüzdesel olarak da belirleyebilirsiniz. Başlangıç değeri 100 MB olan bir veritabanının her seferinde %10 büyümesini isterseniz, 10 MB büyüyerek 110 MB olacaktır. **MAXSIZE** sınırını ve sabit disk (HDD) üst sınırınızı da hesaba katarak, bir büyüme oranı belirlemeniz gerekir.

LOG ON

Log bilgilerinin tutulacağı özel dosyalar kümesi ve bu dosyaların bulunacağı yeri belirlemenizi sağlar. Bu özellik belirtilmezse, log dosyası (ldf) boyutu, veri dosyası (mdf) boyutunun %25'i olacak şekilde tek bir dosya oluşturur.

Profesyonel uygulamalarda veri güvenliği ve sabit disk yazma-okuma gibi bir çok parametre hesaplanarak veritabanı mimarisi oluşturulur. Log dosyası ile veri dosyasının konumlarını belirlemek için de dikkat edilmesi gereken, bu iki dosyanın farklı dosyalarda saklanmasıdır. Hatta log ve veri dosyasının farklı sabit disklerde tutulmasını tavsiye ederim. Hem veritabanı ve sabit disklerin veri okuma-yazma hızını daha etkin kullanabilmek hem de sabit diskte oluşabilecek çökme, disk sürücü hataları gibi durumlarda ek güvenlik sağlayacaktır.

```
LOG ON
( NAME = kodlab_db_log,
    FILENAME = 'C:\Databases\kodlab_db\kodlab_db_log.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB );
```

COLLATE

Veritabanında büyük/küçük harf duyarlılığı, işaret duyarlılığı ve sıralama düzeni gibi özellikleri belirler. SQL Server kurulumu sırasında varsayılan olarak belirlenir. Ancak veritabanı seviyesinde bu özellikleri değiştirebilirsiniz.

FOR ATTACH

Mevcut veritabanı dosyaları kümesini, server'a bağlamak için kullanılır. Bu dosyalar, daha önce veritabanına bağlı olan ve **sp_detach_db** komutu ile veritabanından uygun şekilde ayrılmış dosyalardır. Veritabanına dosya eklemek için **FOR ATTACH** kullanmak zorunda değilsiniz. **sp_attach_db** komutunu kullanarak da dosyaları veritabanına ekleyebilirsiniz. Fakat **FOR ATTACH** özelliği kullanılarak oluşturulan bir **CREATE DATABASE** sorgusunda 32.000 adet dosyayı bağlayabiliriz. Bu dosya bağlayabilme sayısı **sp_attach_db**'de sadece **16**'dır.

FOR ATTACH özelliğini kullanacaksanız, dosya konumu bilgisinin **ON PRIMARY** kısmını tam olarak belirtmelisiniz. Veritabanından ayrılmış dosyaları, aynı veritabanı dosyasına bağlayacağınız sürece, dosyanın konumunu gösteren diğer parametreleri kullanmayabilirsiniz.

DB_CHAINING ON | OFF

SQL Server'da **sahiplik** kavramı vardır. Bu özellik bir nesnenin sahibini ve bu nesneye erişim izinlerini yönetmek için kullanılır. Ancak sahiplik kavramı beraberinde sahiplik zinciri diye nitelendirdiğimiz bir sorunu getirir. Sahiplik zinciri, bir kişinin oluşturduğu nesneye bağlı başka bir nesnenin başka bir kullanıcı tarafından oluşturulmasıyla oluşur. Yani, A kişisinin oluşturduğu nesneye bağlı bir nesnenin B kişisi tarafından oluşturulması anlamına gelmektedir.

DB_CHAINING ON olarak belirlerseniz, veritabanı arasında sahiplik zincirleri çalışır. **DB_CHAINING OFF** olarak belirlerseniz çalışmaz.

Sahiplik zinciri nesnelere erişim, yönetim ve sahipliklerin içinden çıkılamaz haline gelebilmesini sağlayabilir. Bu nedenle bu tür sahiplik zinciri oluşturulabilecek durumlardan mutlaka kaçınmanızı tavsiye ederim.

TRUSTWORTHY

TRUSTWORTHY özelliği, veritabanınızdaki bir nesnenin, sistem dosyası gibi SQL Server ortamı dışındaki nesnelere erişmek istediğinizi ve nesneye **impersonation context** erişimi vermek istediğinizi belirtir. Bu özelliği daha sonra detaylarıyla inceleyeceğiz.

T-SQL İLE VERİTABANI OLUŞTURMAK

En temel haliyle, **KodLab** adında bir veritabanı oluşturalım.

```
USE master
CREATE DATABASE KodLab
```

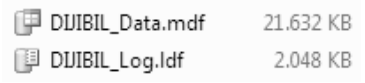
Şimdi de **DIJIBIL** adında detaylı bir veritabanı oluşturalım.

İlk olarak, sabit diskimizin **C:** dizininde, veritabanı dosyalarımızın tutulacağı **Databases** adında bir klasör oluşturuyoruz. Bu klasörü oluşturmazsanız aşağıdaki sorgu çalıştırılma sırasında hata verecektir.

Daha sonra **SSMS**'de sorgu penceresi (*Query Window*) oluşturarak aşağıdaki kodları çalıştırıyoruz.

```
CREATE DATABASE DIJIBIL
ON  PRIMARY
(
 NAME = N'DIJIBIL_Data',
 FILENAME = N'C:\Databases\DIJIBIL_Data.mdf',
 SIZE = 21632KB,
 MAXSIZE = UNLIMITED,
 FILEGROWTH = 16384KB
)
LOG ON
(
 NAME = N'DIJIBIL_Log',
 FILENAME = N'C:\Databases\DIJIBIL_Log.ldf',
 SIZE = 2048KB,
 MAXSIZE = 2048GB,
 FILEGROWTH = 16384KB
)
GO
```

Bu sorgu ile birlikte artık **DIJIBIL** adında bir veritabanına sahibiz. Şimdi **Databases** klasörümüzdeki veritabanı dosyalarımızın isim ve boyut değerlerine bakalım. Sonra da bu sorguda neler yaptığımızı sırasıyla inceleyelim.



Veritabanı oluşturmak için gerekli olan en temel ifadeyi yazıyoruz.

```
CREATE DATABASE DIJIBIL
```

Bu veritabanına ait birincil dosya grubunun belirtileceğini bildirmemiz için gerekli ifadeyi yazıyoruz.

```
ON PRIMARY
```

Bundan sonraki bilgileri, iki ayrı parantez içerisindeki iki ayrı blokta belirtiyoruz.

```
(
 NAME = N'DIJIBIL_Data',
 FILENAME = N'C:\Databases\DIJIBIL_Data.mdf',
 SIZE = 21632KB,
 MAXSIZE = UNLIMITED,
 FILEGROWTH = 16384KB
)
```

Bu ilk kod bloğunda veritabanımızın veri dosyasını tanımlıyoruz.

- **NAME** : Veri dosyasının adı.
- **FILENAME** : Veri dosyamızın işletim sistemindeki fiziksel adı.
- **SIZE** : Veritabanı boyutunu 21632KB (KiloByte) olarak belirtiyoruz.
- **MAXSIZE** : Veritabanı boyutunu limitsiz olarak belirtiyoruz. Limit, sabit diskin kapasite üst sınırı olacaktır.
- **FILEGROWTH** : Veritabanı genişlemesi gerektiğinde, her seferinde 16384KB olarak artmasını istiyoruz.

Veritabanı için belirttiğimiz **KB** cinsinden değerleri **MB** cinsinden de verebilirdik. Bu durumda veri dosyamızın **SIZE** özelliğindeki **21632KB** değeri, **21MB** (**M**ega**B**yte) olarak verebilirdik. (1MB = 1024KB)

Bu bilgiler veri dosyası için hazırlandı. Şimdi Log dosyamızın tanımlamasını inceleyelim.

Log dosyasını tanımlayacağımızı bildirmek için ilk parantezli kod bloğundan sonra;

```
LOG ON
```

Şimdi ikinci kod bloğu ile veritabanımızın log dosyasını belirtebiliriz.

```
(
 NAME = N'DIJIBIL_Log',
 FILENAME = N'C:\Databases\DIJIBIL_Log.ldf',
 SIZE = 2048KB,
 MAXSIZE = 2048GB,
 FILEGROWTH = 16384KB
)
```

Veri dosyası için belirttiğimiz tüm parametreleri log dosyası için de farklı değerler vererek kullandık.

Log dosyasındaki bilgilerin veri dosyasından farkları;

- **`FILENAME`** : Log dosyasının uzantısı **ldf**'dir. Ve veritabanı adından sonra Data değil, Log yazılır.
- **`SIZE`** : Başlangıç olarak log dosyası yüksek boyutlara sahip olmadığı için daha düşük verdik.
- **`MAXSIZE`** : Log dosyası sürekli genişleyeceği için belli bir boyut sınırı koyduk.

Temel ve detaylı olarak iki farklı veritabanı oluşturduk. Şimdi bu veritabanının yapısına bakalım.

`sp_helpdb` sistem prosedürü ile bir veritabanının yapısını gözlemleyebilirsiniz. Bu sistem prosedürü, veritabanın içerisindeki nesneleri değil, yapısını incelemek için oluşturulmuştur.

```
EXEC sp_helpdb 'DIJIBIL';
```

`EXEC` ve **`EXECUTE`** komutları stored procedure'leri çalıştırmak için kullanılır. İlerleyen konularda detaylarıyla anlatılacaktır.

	name	db_size	owner	dbid	created	status	compatibility_level
1	DIJIBIL	23.13 MB	Cihanozhan\Cihan Özhan	9	Oct 19 2012	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	110

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	DIJIBIL_data	1	C:\Databases\DIJIBIL_Data.mdf	PRIMARY	21632 KB	Unlimited	16384 KB	data only
2	DIJIBIL_Log	2	C:\Databases\DIJIBIL_Log.ldf	NULL	2048 KB	2147483648 KB	16384 KB	log only

Bu sorgunun sonucunda gelen veriler ikiye ayrılır.

İlk veriler;

Veritabanı dosyasının mantıksal ve SQL Server mimarisindeki parametrik ayarlarını gösterir. Buradaki `db_size` ile veri ve log dosyasının toplam boyutunu, `dbid` değeri SQL Server'da kaç adet veritabanı oluşturulduğunu, veritabanın oluşturulma tarihi, sahiplik bilgisi (*owner*) gibi bilgileri gösterir.

İkinci veriler;

Veritabanının fiziksel dosyaları hakkında bazı bilgileri gösterir. Bunlar, ekran görüntüsünde gördüğünüz gibi anlaşılması kolay bilgilerdir.

Video: SSMS kullanarak bir veritabanı oluşturma detayları

CREATE TABLE İLE TABLO OLUŞTURMAK

`CREATE` ile veritabanı oluşturabildiğimiz gibi tablo da oluşturabilmekteyiz. `CREATE` konusuna giriş kısmında söylediğimiz gibi, bu ifade tüm nesneleri oluşturmak için kullanır.

TABLO VE SÜTUN İSİMLERİ

Tablo ve sütun isimlendirme bazı standartlara sahip olsa da genel olarak geliştiriciden geliştiriciye değişiklik gösterir. Bazı geliştiriciler bu standartları beğenmiyor olacak ki, okumanızın bile zor olduğu isimlendirmeler yapabilmekteler. Ancak size tavsiyem; mutlaka okunaklı, açık ve anlamı bozmayacak şekilde bir standart ile geliştirme yapmanızdır. Bu isimlendirmelerdeki sade ve okunaklılık sizin de işinizi kolaylaştıracak ve sonraki geliştirme sürecinde çok işinize yarayacaktır.

Genel olarak nesne isimlendirme kuralları bu durumda da geçerlidir. Bu konuda gerekli açıklamayı **SQL Server Veritabanı Nesneleri** bölümünden inceleyebilirsiniz.

Diğer bazı kurallar şunlardır:

- İsimleri kısa ve anlamlı tutun.
- İsimlendirirken her kelimenin baş harfini büyük, devamında küçük harfler kullanın.

- Türkçe karakter (`i`, `İ`, `ğ`, `Ğ` gibi) kullanmayın.
- Boşluk bırakmamaya çalışın. Bitişik ve her kelime baş harfini büyük yazın. Boşluk mutlaka gerekliyse, köşeli parantez (`[]`) kullanın.
- Anlam bozulmayacak şekilde kısaltmalar kullanın (`No`, `ID`, `Net[Network]` gibi).

Veri Tipleri

Tablo tasarımlarında veri tipleri birçok sebeple önemlidir. SQL Server'da doğru veri tipini seçmek tecrübe ve veri tiplerine tam hakimiyet gerektirir. Doğru seçilmeyen veri tipi, performans kaybı, hata oluşturma, veri bütünlüğünü bozmak ve veritabanında gereksiz alan kaplama gibi bir çok soruna sebebiyet verebilir.

Bu nedenle uygulama mimarisinde kullanılması gereken veri tipini doğru seçmek için iyi araştırma ve proje analizi gerekmektedir.

DEFAULT

Kullanıcı tarafından belirtilmeyen ya da belirtilmesi zorunlu olmayan durumlarda SQL Server tarafından verilen otomatik değerdir.

Buna bir örnek olarak, kayıt tarihini kullanıcıdan almak yerine, `GETDATE()` fonksiyonu ile o anın tarih ve zaman bilgisini, otomatik olarak sistem tarafından veritabanına girilmesi sağlanabilir.

Bununla ilgili kullanım ve anlatımları ileriki bölümlerde detaylarıyla yapacağız.

IDENTITY

SQL Server'da `IDENTITY`, kayıt ekleme gibi işlemlerde büyük öneme sahiptir. Bir sütunu, `IDENTITY` sütun olarak belirlediğinizde, SQL Server her kayıt için o sütuna sayı atar. Atanan sayı, **seed** (*başlangıç*) ve **increment** (*artış miktarı*) kadar artar ya da azalır.

`Seed` ve `increment` için varsayılan değer `1`'dir. Bu, seed ile `1`'den başlar, `increment` ile birer birer artar anlamına gelir. Bu varsayılan değerleri değiştirebilirsiniz. Örneğin, `5`'den başlar ve beşer beşer artırabilirsiniz. Bu durumda ilk kayıt için `5` değeri verilir ve `5`, `10`, `15` şeklinde artarak devam eder.

`IDENTITY` sütunlar sayısal olmak zorundadır. Otomatik artan özelliğinin sayısal olmaması zaten düşünülemez. Veri tipi olarak da `INT` (*integer*) ve

BIGINT kullanımı yaygındır. Ancak bunlardan daha küçük sayısal veri tipleri de kullanılabilir. Bu tamamen **IDENTITY** sütunun maksimum alabileceği değer ile alakalıdır. Çok yüksek kayıtlara ulaşmayacak bir sütun için **BIGINT** kullanmak I/O'da gecikmeye ve performansı olumsuz yönde etkileyecek bir durum oluşturur.

IDENTITY sütun kullanırken, bazı durumlarda önceki kayıtlar için kullanılmış olan numaraları tekrar kullanmak isteyebilirsiniz. Örneğin; 1'den başlayıp 10'a kadar gidilmiş bir kayıtta 4. kayıt silinmişse, bu kayıt numarası boş kalacaktır. Boş kalan bu kayıt numarasını, yeni bir kayıt için kullanmak istersek, **SET IDENTITY INSERT ON** özelliğini kullanmalıyız. Bu özellik **IDENTITY** sütuna otomatik değer atanmasını engelleyecektir. Böylece **IDENTITY** değerli silinmiş kaydın yerine yeni bir kayıt ekleyebilirsiniz.

IDENTITY sütunlar, genel olarak tablolardaki birincil anahtarlarda yeni sayısal değer oluşturmak için kullanılır. **IDENTITY** sütun ile **PRIMARY KEY** (*birincil anahtar*) kavramı, bir arada çok sık kullanıldığı için aynı ya da benzer gibi görünse de tamamen farklı kavramlardır. **IDENTITY** belirli bir sayıdan başlayarak belirli bir aralık ile artan ya da azalan sayısal değer üretir. **PRIMARY KEY** ise, bir sütunda bulunan değerin benzersiz olmasını sağlar. **IDENTITY** bir değeri sıfırlayarak daha önce artırarak saydırdığınız aralıkta yeniden saydırmaya başlayabilirsiniz. Ancak sütunun **PRIMARY KEY** olması bu durumda ortaya çıkacak veri tekrarını önler. Yani **IDENTITY**, 1 ile 10 numaraları arasında birer birer artarak değer üretirken **PRIMARY KEY**'de bu üretilen değerlerin, o sütunda benzersiz olmasını garanti eder.

NOT FOR REPLICATION

Replikasyon işlemi, veritabanının yerel ya da uzak sunucuya bazı bilgileri ya da tüm bilgileri kopyalanma işlemini gerçekleştirir. **NOT FOR REPLICATION** parametresi de veritabanı bir başka veritabanına kopyalandığında, bir kimlik değeri verilip verilmeyeceğini belirler.

ROWGUIDCOL

Bu özelliği replikasyon işleminde bir **IDENTITY** sütun kullanmaya benzetebiliriz. İki tablo arasında veri ilişkilendirme yapıldığında **IDENTITY** sütun kısmi olarak yeterli gelebilir. Ancak daha karmaşık verilerin replikasyon işlemine tabi tutulduğu bir durumda **IDENTITY** bize doğru bir yetenek sunmayacaktır.

Bu durumda **IDENTITY** değerleri mutlaka çakışacak ve ileriye dönük bir çözüm sunamayacaktır.

Bu durumda, bize daha büyük veriler için unique (*benzersiz*) değer üretecek bir özelliğe ihtiyaç duyarız. Bu tür sorunlara çözüm olması için oluşturulan **GUID** (***G****lobal* ***U****nique* ***ID****entifier*) kavramı devreye girer. GUID 128-bit bir değer üreten, 38 basamaklı ondalık şeklinde bir değerdir. GUID ile her saniye yeni bir değer üretseniz bile, teorik olarak milyonlarca yıl sonra tekrar aynı değeri üretme ihtimaline sahiptir. GUID'in ürettiği değeri tekrar etme olasılığı, kimi hesaplamalara göre de milyarlarca değer de bir ihtimaldir. Büyük ihtimalle emin olmamız gereken şu ki; bu değer kolay kolay kendini tekrarlamayacaktır.

Bilgisayar ya da web programlamada da kullanılan GUID değerini üretmek için Windows işletim sisteminde gömülü **Win32 API** vardır. Bu API ile değer üretilerek kullanılır. Aynı şekilde bu değeri üretecek bir de **SQL Server fonksiyonu** vardır. SQL Server'da GUID değer üreten fonksiyonun adı **NEWID()**'dir.

Bu fonksiyonu kullanarak nasıl bir değer üretildiğine bakalım.

```
SELECT NEWID() AS GUID;
```

	GUID
1	E1E00357-10FA-4471-BBD9-E6D96B9907B3

COLLATE

Sütun seviyesinde büyük/küçük harf duyarlılığı, işaret duyarlılığı ve sıralama düzeni gibi özellikleri belirler.

NULL / NOT NULL

Tablodaki bir sütunun, **NULL** değeri kabul edip etmeyeceğini belirtmek için kullanılır. **NULL** olarak geçilemeyecek bir sütuna değer girilmez ve **NULL** bırakılırsa istenilen işlem yapılmayacaktır. **NULL** kavramı detaylı olarak uygulama konularında anlatılacaktır.

Sütun Kısıtlamaları

Tablodaki sütunumuza girilecek veriyi kontrol edip bazı kısıtlamalar uygulamamız gerekebilir. Örneğin; tarih tutulacak bir alanda 100 sene öncenin

tarihinin girilmesine izin vermek, programsal bir mantık hatası olacaktır. Ya da yılın aylarını içeren bir sütunda 13. ay olarak bir değer girilmesine kısıtlama getirmemek bir hatadır.

Bu gibi sorunları önlemek için sütun bazlı kısıtlama yapılmalıdır.

Hesaplanmış Sütunlar

Tablodaki sütunlar ile elde edilen, hesaplama ve anlık veri gösterimi gibi durumlarda görüntülenecek kayda bir sütun ismi vermemizi sağlayan özelliktir. Kendine ait bir değer ya da veri yoktur. Gösterdiği veriyi farklı işlemlerden alır. Amacı veriler için bir takma isim oluşturmaktır.

Söz dizimi:

```
sutun_ismi AS hesaplanan_sutun_ifadesi
```

Kullanımı:

```
SELECT FirstName + ' ' + LastName AS [İsim] FROM Person.Person;
```

Tablo Kısıtlamaları

Tablo kısıtlamaları (*constraints*), tabloya eklenecek veri için kısıtlama getirmekte kullanılır. Tablo seviyeli kısıtlamalar **PRIMARY**, **FOREIGN KEY** ve **CHECK** constraint'leri içerir.

ON

ON ifadesi, tablo tanımının hangi dosya grubu üzerinde yer alacağını gösterir. Bir tabloyu **ON** ifadesini kullanmayarak varsayılan dosya grubuna (**PRIMARY**) yerleştirebilirsiniz.

Tablo Oluşturmak

Oluşturduğumuz **DIJIBIL** veritabanında **diji_Kullanicilar** adında bir tablo oluşturalım.

```
USE DIJIBIL
GO
CREATE TABLE diji_Kullanicilar(
 kul_ID     INT            NOT NULL,
```

```
 kul_ad             VARCHAR(20)  NOT NULL,
 kul_soyad          VARCHAR(20)  NOT NULL,
 kul_telefon        VARCHAR(11)  NULL,
 kul_email          VARCHAR(320) NOT NULL
);
```

Tablo oluştururken **kul_telefon** sütununda **NULL** değere izin verdik. Bunun sebebi; kayıt olacak kullanıcıların telefon numarasını vermek zorunda bırakmamaktır. Diğer bilgiler gereklidir, fakat telefon için bu gereklilik söz konusu değildir. Tabi ki isterseniz **kul_telefon** sütununu da **NOT NULL** tanımlayarak telefon bilgisini zorunlu hale getirebilirsiniz.

RFC 2821'e göre geçerli bir e-mail adresinin uzunluğu en fazla 320 karakterdir. Ve **VARCHAR** veri tipinin alabileceği maksimum alabileceği uzunluk 8000 bytes'dır. E-mail adresinin maksimum değerinin üzerinde bir değer atarsanız, bu gereksiz veri alanı kullanımı, yazılım uygulaması ile veri boyutu uyuşmazlığı ve injection saldırılarında güvenlik zafiyeti oluşturabilir. İyi bir geliştirici, ister son kullanıcı olsun, ister T-SQL sorgularını kullanacak geliştirici olsun, kullanıcının hata yapma ihtimalini hesap ederek geliştirme yapmalıdır. Injection ile ilgili detaylı anlatım, sonraki veri güvenliğiyle ilgili bölümlerde anlatılacaktır.

diji_Kullanicilar tablosunu hakkında yapısal bilgileri öğrenelim.

```
EXECUTE sp_help 'diji_Kullanicilar';
```

ALTER İLE NESNELERİ DEĞİŞTİRMEK

ALTER ifadesi, **CREATE** ifadesiyle oluşturulan nesneler üzerinde yapısal değişiklikler yapılabilmesini sağlar.

```
ALTER <nesne_tipi> <nesne_adi>
```

VERİTABANINI DEĞİŞTİRMEK

ALTER ifadesiyle, **DIJIBIL** veritabanımızın yapısında bazı değişiklikler yaparak inceleyelim.

Veritabanı üzerinde değişiklik yapmadan önce yapısını inceleyelim;

```
EXEC sp_helpdb DIJIBIL;
```

Biz **DIJIBIL** veritabanını bir test ve uygulama veritabanı olarak hazırladık. Bu nedenle boyutlarını çok küçük tuttuk. Şimdi bu veritabanına dışarıdan 250MB'lık bir veri ekleyeceğimizi düşünelim. Veritabanımızın veri dosyası 21MB büyüklüğünde ve **FILEGROWTH**, yani tek seferde boyut büyütme-genişletme değerimiz 16MB idi. 250MB'lik veri eklemesini yaparken, 21MB'lik veritabanımız 16MB'lik parçalar halinde, defalarca genişletme işlemine tabi olacaktır. Tek seferde yapılabilecek genişletme işlemi yerine, defalarca bu işlem yapılacaktır. Bu da performans açısından olumsuz bir durum teşkil eder. Doğru olan bu işlemin tek seferde yapılmasıdır.

Şimdi, veritabanı boyutunu genişletmek için **DIJIBIL** veritabanımızın yapısında değişiklik yapalım.

DIJIBIL veritabanımızın veri depolama kapasitesini artırmak için veri dosyasının (MDF) genişliğini artırmamız gerekir. Veritabanını oluştururken bu dosyaya verdiğimiz **NAME** bilgisine ihtiyacımız olacak. Biz veri dosyasının **NAME** bilgisine **DIJIBIL_Data** adını vermiştik.

```
ALTER DATABASE DIJIBIL
MODIFY FILE
(
 NAME = DIJIBIL_Data,
 SIZE = 300MB
);
```

ALTER ifadesinde kullandığımız **NAME** bilgisi veri dosyamızın adını içerir. Bu sorgu ile **DIJIBIL** veritabanına ait **DIJIBIL_Data** dosyasının (veri dosyamızın adı), **SIZE** yani dosya büyüklüğü değerini 300MB yapmak istediğimizi bildirerek gerekli genişletme işlemini yapmış oluyoruz.

Bu sorgunun sonucunda fiziksel dosyaların yeni bilgilerini ilgili klasörden inceleyelim.

Şimdi de, yeni veritabanı boyutumuzu **sp_helpdb** sistem prosedürümüzle görelim.

```
EXEC sp_helpdb DIJIBIL;
```

	name	db_size	owner	dbid	created	status	compatibility_level
1	DIJIBIL	302.00 MB	Cihanozhan\Cihan Özhan	9	Oct 19 2012	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	110

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	DIJIBIL_data	1	C:\Databases\DIJIBIL_Data.mdf	PRIMARY	307200 KB	Unlimited	16384 KB	data only
2	DIJIBIL_Log	2	C:\Databases\DIJIBIL_Log.ldf	NULL	2048 KB	2147483648 KB	16384 KB	log only

Veritabanı genişletme ile ilgili önemli bir husus da şudur; **DIJIBIL** veritabanımızın **MAXSIZE** parametre değeri **UNLIMITED** olarak bildirildi. Ancak bu değer 300MB'den düşük olsaydı da bu sorgu çalıştığında hata vermeyecekti. Bunun sebebi veritabanı boyutunu bilinçli bir şekilde, açıkça değiştirmiş olmamızdır. Bu durumda veritabanının **MAXSIZE** değerini de yeni boyutlandırmaya göre yeniden güncelleyip, yükseltmeniz gerekecekti.

Eğer **MAXSIZE** olarak **UNLIMITED** değil de, 100MB vermiş olsaydık ve veritabanının **MAXSIZE** değerine kadar **FILEGROWTH** değeri ile otomatik olarak SQL Server'ın kendisinin artırmasını bekleseydik, 250MB veri eklemeye çalıştığımızda **MAXSIZE** değerini aşamayacak ve verilerin kalan kısmı bu sınırlamadan dolayı eklenemeyecekti.

Ayrıca gene **MAXSIZE** ile 100MB sınırlaması getirdiğimiz bir veritabanına yukarıda yaptığımız gibi T-SQL kullanarak veritabanı genişletme işlemi uyguladığımızda yeni boyut (300MB), **MAXSIZE**'ın üzerinde olacağı için yeni **MAXSIZE** değeri en son yaptığımız genişletme işlemindeki değer olacaktır. Ve veritabanı dolmuş olacağı için yeniden bir **MAXSIZE** tanımlaması yapmanız gerekecektir.

Veritabanı Tablosunu Değiştirmek

SQL Server'ın temel nesnelerinden olan tablolar en fazla işleme tabi tutulan nesnelerdir. Veri ile ilgili tüm işlemler ve nesnelerin tablolar üzerinden gerçekleştirilmesi, zaman içerisinde veri ya da çeşitli yapılandırma gereksinimleri nedeniyle, birçok düzenleme ve yapısal değişiklik yapılmasını zorunlu hale getirir.

Bu işlem için **diji_Kullanicilar** tablosunu kullanacağız. Öncelikle tablomuzun yapısını inceleyelim.

```
EXEC sp_help diji_Kullanicilar;
```

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	kul_ID	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	kul_ad	varchar	no	20			no	no	no	Turkish_CI_AS
3	kul_soyad	varchar	no	20			no	no	no	Turkish_CI_AS
4	kul_telefon	varchar	no	11			yes	no	yes	Turkish_CI_AS
5	kul_email	varchar	no	320			no	no	no	Turkish_CI_AS

Bu sorguyu çalıştırdığınızda ekranınıza 5 ayrı sonuç listelenecektir. Ben size, tablomuz ile doğrudan ilgili olan 2. sonucu gösteriyorum.

Şimdi, tablomuzda kullanıcılarımızın adres ve kayıt tarih bilgilerini saklamak için **kul_adres** ve **kul_kayitTarih** sütunları ekleyelim.

```
ALTER TABLE diji_Kullanicilar
ADD
kul_adres       VARCHAR(150) NULL,
kul_kayitTarih  DATETIME     NOT NULL DEFAULT GETDATE();
```

Bu sorgu ile birlikte artık tablomuza **VARCHAR** veri tipinde ve 150 karakterlik veri alabilen **kul_adres** sütunumuz ve kullanıcı kayıt bilgisini tutan **kul_kayitTarih** olmak üzere iki sütunumuz eklenmiş oldu. **kul_adres** sütunumuz **NULL** geçilebilir, fakat **kul_kayitTarih** sütunumuza veri girişi yapılması **NOT NULL** ile zorunlu tutulmaktadır. Eğer yazılım ya da son kullanıcı tarafından veritabanına kayıt tarih bilgisi gönderilmez ise **DEFAULT** ile belirttiğimiz **GETDATE()** sistem fonksiyonu, sistemin o anki zaman bilgisini ekleyerek boş bırakılmamasını garanti edecektir.

DROP İLE NESNE SİLMEK

SQL Server'da nesneleri silmek için **DROP** kullanılır. **DROP**'un amacı veri silmek değil, nesnelerin kaldırılmasını (silinmesi) sağlamaktır.

```
DROP <nesne_tipi> <nesne_ismi>
```

Bir Veritabanı Tablosunu Silmek

Veritabanında bir tablo silmek için, tek satırlık **DROP** sorgusu yeterlidir. Bunun için;

```
DROP TABLE <tablo_ismi>
```

DIJIBIL veritabanımızda silme işlemi için oluşturduğum **diji_Kitaplar** tablosunu silelim.

```
USE DIJIBIL                          Command(s) completed successfully.
GO
DROP TABLE diji_Kitaplar;
```

Burada, silme işlemini gerçekleştiren komut aslında sadece 3. satırdaki **DROP** ile başlayan komuttur.

USE DIJIBIL yazılmasının sebebi; yukarıdaki **Available Databases** bölümünde, farklı ve bu silme işlemini yapacağımız veritabanıyla aynı tablolara sahip bir veritabanı seçiliyse **DROP** başarıyla çalışacak ve bu tabloyu silecektir. Ancak bu silme işlemiyle, bizim istediğimiz veritabanında değil, farklı ve belki de önemli verilerin bulunduğu bir veritabanındaki kayıtların bulunduğu tabloyu silmiş olabiliriz. Bu nedenle, veritabanında her ne için ve ne silerseniz silin, bu işlemi 'nokta atışı' tabir edilecek şekilde net olarak belirtmek için **USE** ifadesiyle birlikte veritabanını belirtmenizi mutlaka öneririm.

Bu sorguda bulunan GO komutu da zorunlu değildir. GO olmadan da sorgunuz başarılı ve eksiksiz şekilde çalışacaktır. Ancak gelişmiş sorgu yapılarında, uzun kod bloklarında işlemlerin çalışma sıra ve anlaşılması gibi sebeplerden dolayı faydalı olacak bir kullanım alışkanlığıdır.

Bir Veritabanını Silmek

Bir veritabanını silmek, bir tablo silmek kadar kolaydır. Bu işlem için gerekli söz dizimi;

```
DROP DATABASE <veritabani_ismi>
```

KodLab ismiyle oluşturduğumuz veritabanını silelim.

```
USE master
GO
DROP DATABASE KodLab;
```

Bu sorgu çalıştığında **KodLab** isimli veritabanı silinmiş olacaktır.

Veritabanı silme işlemi şu sebeplerden dolayı başarısız olabilir. Bunlar;

- SSMS'de geçerli olan bir veritabanını silmeye çalışmadığınıza emin olun.
- Silmeye çalıştığınız veritabanına açılmış bir bağlantı olmadığına emin olun. Bunun için SSMS'yi kontrol edebileceğiniz gibi, **sp_who** sistem prosedürüyle de öğrenebilirsiniz.

Veri İşleme Dili (DML = Data Manuplation Language)

Veri İşleme Dili (*DML*), tablolar üzerinde ekleme, güncelleme, seçme ve silme işlemlerini yapan ifadelerden oluşur. **INSERT** ile veri ekleme, **UPDATE** ile güncelleme, **SELECT** ile seçme ve görüntüleme, **DELETE** ile de veriyi silme işlemlerini gerçekleştirebiliriz. Veritabanı programlamanın veri ile ilgili kısmı temel olarak bu dört ifadeden oluşmaktadır.

Bu bölümde veri işleme dilinin temellerini anlatacağız.

INSERT ile Veri Eklemek

Bir tabloya kayıt eklemek için kullanılır.

```
INSERT INTO tablo_ismi(sutun1_isim[,sutun2_ismi,...])
VALUES(deger1[,deger2,deger3,...])
```

diji_Kullanicilar tablomuza yeni bir kullanıcı ekleyelim.

```
INSERT INTO diji_Kullanicilar
VALUES(1,'Cihan','Özhan',
  '02124537488','cihan.ozhan@dijibil.com',
  'Istanbul',GETDATE());
```

Bu sorguda tablo adının yanında herhangi bir sütun tanımlamasını yapmamamız dikkatinizi çekmiş olmalı. Sütun tanımlaması yapmak zorunda değiliz. Ancak bu durum bazı kurallara tabi tutulmuştur. Eğer yukarıdaki örnek gibi, herhangi bir sütun tanımlamadan veri girişi yapmak istiyorsanız; **VALUES** kısmında tablodaki tüm sütunlara, soldan sağa doğru sıralı şekilde veri eklemesi yapmamız gerekir. Yani, sütun ismi belirtmeden veri giriyorsak SQL Server gerekli sütunların isimlerini ve değerlerini bizden isteyerek sorumluluğu bize bırakıyor.

Şimdi de, sütun isimleriyle birlikte yeni bir kayıt ekleyelim.

```
USE DIJIBIL
GO
INSERT INTO diji_Kullanicilar(
 kul_ID, kul_ad, kul_soyad,
 kul_telefon,kul_email,kul_adres,
 kul_kayitTarih
)
VALUES(2, 'Kerim','Fırat','02124532122',
 'kerim.firat@dijibil.com','Istanbul',GETDATE());
```

Bu şekilde de başarılı bir kayıt ekleme işlemi gerçekleşecektir. Sütun isimleriyle **INSERT** işleminde dikkat edilmesi gereken, sütun isimleriyle bu sütunlara vereceğimiz değerlerin, aynı sırayla veritabanına gönderilmesi gerektiğidir.

İki soru tipi için de ortak bir özellik de, tablomuzda bulunan **kul_ID** sütunlarını sorgumuzda yazmamış olmamızdır. Bunun nedeni, **kul_ID** sütunu **PRIMARY** ve **IDENTITY** sütun olmasıdır. Bu özelliklere sahip sütun, kendi değerini otomatik olarak ürettiği için bizim herhangi bir değer vermemize gerek yoktur. Eğer **kul_ID** sütununa değer gönderseydik, sorgu çalışmayacak ve hata üretecekti.

INSERT sorgunuzun sütun kısmına **kul_ID**, değer kısmına da ilk değer olarak bir sayısal veri ekleyerek sorguyu çalıştırıp, çıkan hata sonucunu inceleyebilirsiniz.

SELECT İLE VERİ SEÇMEK

Bir tabloda bulunan kayıtları seçmek için kullanılır. Veritabanı programlamanın veri işlemlerinde en çok kullanılan ve en karmaşık hallerde sorgular geliştirilen özelliğidir. Konular ilerledikçe neden **SELECT** ifadesinin daha çok kullanıldığını fark edeceksiniz.

```
SELECT sutun_ismi1[,sutun_ismi2,...]
FROM tablo_ismi
```

En temel **SELECT** sorgumuz ile önceki **INSERT** konusunda girdiğimiz kayıtları seçelim.

```
USE DIJIBIL
GO
SELECT * FROM diji_Kullanicilar;
```

	kul_ID	kul_ad	kul_soyad	kul_telefon	kul_email	kul_adres	kul_kayitTarih
1	1	Kerim	Fırat	02124531234	kerim.firat@dijibil.com	Istanbul	2013-01-10 00:16:01.187
2	2	Cihan	Özhan	02123456789	cihan.ozhan@dijibil.com	Istanbul	2013-01-10 00:16:04.637

Sorguda kullandığımız yıldız (*) işaretinin anlamı, belirtilen tabloda bulunan tüm sütunların seçilmek istenmesidir. Yıldız işareti bizi tüm sütunları tek tek yazmaktan kurtardığı için faydalıdır diyebiliriz. Fakat, yıldız işaretinin önemli dezavantajları vardır. Dezavantajlarını anlayabilmek için öncelikle nasıl çalıştığını inceleyelim.

Yıldız işaretiyle hazırladığımız sorgu veritabanı motoruna gönderilir. Veritabanı motoru yıldız (*) işaretinin ne olduğunu, hangi sütunları ifade ettiğini anlamak için sorgu sonunda bulunan tablo adının yapısını sorgulayarak sistemde bu tabloya (**diji_Kullanicilar**) ait tüm sütunların isimlerini elde eder. Daha sonra bu isimleri açık **SELECT** sorgusu halinde sıraya sokar ve bu şekilde sorguyu çalıştırarak sonucu ekrana getirir. Bu işlem, veritabanına ek sorgular gerçekleştirmesini zorunlu hale getirdiği için ağ trafiği, veritabanı gücünün gereksiz kullanımı, sorgular çoğaldığı için de sorgunun daha uzun sürmesi gibi bir çok performans kaybını beraberinde getirir. Ayrıca bir tabloda genel olarak tüm sütunları istemeyiz. Fakat yıldız (*) kullanımında bize tüm sütunlar ve bu sütunlara ait tüm kayıtlar getirileceği için yıldız işaretinin dezavantajlarının yanında bir de gereksiz kayıtların getireceği performans kaybı da eklenecektir. Bu nedenle yıldız (*) işaretini profesyonel uygulamalarda kullanmamanız kesinlikle tavsiye edilir. Bu işareti kullanabileceğiniz alanlar test ve geliştirme süreçleri olmalıdır.

Yıldız (*) işaretiyle hazırladığımız sorguyu açık sütun isimleriyle tekrar hazırlayalım.

```
USE DIJIBIL
GO
SELECT
 kul_ID,kul_ad,kul_soyad,kul_telefon,
 kul_email,kul_adres,kul_kayitTarih
FROM
 diji_Kullanicilar;
```

Bu sorgu sonucunda gelen kayıt ile yıldız (*) işaretiyle gelen kayıtlar tamamen aynıdır. Ancak performans olarak bu sorgulama yöntemi yıldız (*) işaretliye göre daha performanslıdır.

UPDATE İLE VERİ GÜNCELLEMEK

Tabloda satır ya da satırların değerlerini değiştirir-günceller.

```
UPDATE tablo_ismi
SET  alan = deger
WHERE sart_tanimlari
```

Tablomuzda **ID** değeri 3 olan kullanıcının adres bilgisini "*İstanbul / Fatih*" olarak değiştirelim.

```
UPDATE diji_Kullanicilar
SET  kul_adres = 'İstanbul / Fatih'
WHERE kul_ID = 2;
```

2	2	Kerim	Fırat	02124532122	kerim.firat@dijibil.com	İstanbul / Fatih	2013-02-18 19:30:32.267

UPDATE işleminde güncellenecek kayıt konusunda dikkatli olmalıyız. Standart olarak **UPDATE** ifadesi, **WHERE** filtreleme ifadesi olmadan da çalışacaktır. Ancak **WHERE** olmadan çalışan bu sorgu, tablodaki tüm kayıtların adres bilgilerini değiştirecektir.

DELETE İLE VERİ SİLMEK

Tabloda bulunan verileri silmek için kullanılır. **UPDATE** işleminde olduğu gibi, **DELETE** sorgusunda da hangi verilerin silinmek istendiği **WHERE** ifadesiyle belirtilmelidir. **WHERE** ifadesi kullanılmayan **DELETE** sorgusu, tablodaki tüm kayıtların silinmesine neden olacaktır.

```
DELETE FROM tablo_adi
WHERE sart_tanimlari
```

diji_Kullanicilar tablomuzdaki 3 **ID**'li kaydı silelim.

```
DELETE FROM diji_Kullanicilar WHERE kul_ID = 2;
```

Sizin tablonuzda kayıt ekleme ve silme denemelerinize göre 3 **ID**'li kayıt olmayabilir. Farklı bir **ID** bilgisine sahip kaydı silebilirsiniz.

DELETE sorgumuzda **WHERE** filtresi kullanmadığımızda oluşacak sonuçları kendiniz test etmek için;

```
DELETE FROM diji_Kullanicilar;
```

Bu sorguyu çalıştırdığınızda ilgili tablodaki tüm kayıtların silindiğini göreceksiniz. Bu nedenle, **DELETE** sorgusunu kullanırken sonuçlarını tekrar gözden geçirmelisiniz.

Bu ifade de dikkat edilmesi gereken bir nokta da sileceğiniz verinin türüdür. Benzersiz (*unique*) bir değer silecekseniz, bu silme işlemi sonucunda tek kayıt silineceği kesindir. Ancak adı ***C*** harfi ile başlayan kullanıcıları silmek istediğinizde bu işlem sonucunda kaç kayıt silineceğini tahmin edemeyebilirsiniz. Bu nedenle, **DELETE** ve **UPDATE** ifadesini kullanırken kapsam hesaplaması yapılmalıdır.

VERİ KONTROL DİLİ (DCL = DATA CONTROL LANGUAGE)

Veri Kontrol Dili (DCL), bir veritabanına erişecek kullanıcıları ve rollerin izinlerini değiştirmek için kullanılır.

GRANT	Kullanıcıların verileri kullanmasına ve T-SQL komutlarını çalıştırmasına izin verir.
DENY	Kullanıcıların verileri kullanmasını kısıtlar.
REVOKE	Daha önce yapılan tüm kısıtlama ve izinleri iptal eder.

SQL Server'da DCL komutlarını kullanabilmek için varsayılan olarak yetki sahibi olan gruplar;

- **sysadmin**
- **dbcreator**
- **db_owner**
- **db_securityadmin**

Veritabanı sunucusuna dışarıdan erişebilmek için bir login oluşturulmalıdır. **DIJILogin** isminde bir login oluşturalım.

```
CREATE LOGIN DIJILogin WITH PASSWORD = 'login_sifre';
```

Oluşturduğumuz **DIJILogin** isimli login ile sunucuya erişebilmemiz için bir User (Kullanıcı) tanımlayalım.

```
CREATE USER DIJIUser FOR LOGIN DIJILogin;
```

Kullanıcı (*User*) ve giriş (*Login*) tanımlamalarımızı aynı isimler ile yaptıysak, kullanıcı tanımlarken kullandığımız **FOR LOGIN** ifadesine gerek yoktur.

```
CREATE APPLICATION ROLE AppRole
WITH PASSWORD = 'app_role_pwd',
DEFAULT_SCHEMA= AppRole;
```

GRANT İLE YETKİ VERMEK

Veritabanı kullanıcısına, veritabanı rolü ya da uygulama rolüne izinler vermek için kullanılan komuttur.

```
GRANT <ALL veya izinler>
ON <izin_verilenler>
TO <hesaplar>
```

ALL ifadesi, tüm hakları vermek için kullanılır.

DIJIUser kullanıcısına tablo oluşturma izni verelim.

```
GRANT CREATE TABLE
TO DIJIUser;
```

Aynı anda **AppRole** isimli uygulama rolümüzü de yetkilendirebiliriz.

```
GRANT CREATE TABLE TO AppRole, DIJIUser;
```

Yeni bir kullanıcı oluşturarak hem veritabanı, hem de tablo oluşturma izni verelim.

newDijiUser isimli yeni kullanıcımızı oluşturalım.

```
CREATE USER newDijiUser FOR LOGIN DIJILogin;
```

newDijiUser kullanıcımıza veritabanı ve tablo oluşturma izni verelim.

```
GRANT CREATE DATABASE, CREATE TABLE TO newDijiUser;
```

WITH GRANT OPTION İLE BASAMAKLI YETKİLENDİRME

Bu deyim ile izin verilen bir kullanıcının bu nesne üzerinde aldığı izni bir başka kullanıcıya verebilmesi için kullanılır.

```
GRANT SELECT, INSERT ON diji_Kullanicilar
TO AppRole
WITH GRANT OPTION;
```

Bu durumda **AppRole** ile belirtilen role sahip herkes, **diji_Kullanicilar** tablosu üzerinde başkalarına da **SELECT** ve **INSERT** erişim izni tanımlama hakkına sahip olacaktır.

DENY ile Erişimi Kısıtlamak

Kullanıcıların erişimlerini kısıtlamak için kullanılır.

```
DENY { ALL veya izinler} TO {kullanıcılar}
```

DIJIUser isimli kullanıcımızın tablo oluşturma yetkisini kaldıralım.

```
DENY CREATE TABLE TO DIJIUser;
```

DIJIUser isimli kullanıcımızın **diji_Kullanicilar** tablosunda **SELECT** işlemi yapmasını engelleyelim.

```
DENY SELECT ON diji_Kullanicilar TO DIJIUser;
```

REVOKE ile Erişim Tanımını Kaldırmak

Tüm kısıtlama ve izinleri iptal etmek için kullanılır. Bir nesneyi oluşturan kullanıcının, nesne üzerindeki yetkilendirme ve kullanım hakkı iptal edilemez.

REVOKE komutunu, **sys_admin** rolü ya da **db_owner**, **db_securityadmin** sabit veritabanı rollerine sahip kullanıcılar ve nesne için, **dbo** olan kullanıcı çalıştırabilir.

```
REVOKE {ALL veya izinler} {TO veya FROM} {hesaplar}
```

PUBLIC rolüne verilen tüm yetkileri kaldıralım.

```
REVOKE ALL TO PUBLIC
```

DIJIUser kullanıcımıza uyguladığımız **diji_Kullanicilar** tablosunda **SELECT** işlemini yapamama kısıtlamasını kaldıralım.

```
REVOKE SELECT ON diji_Kullanicilar TO DIJIUser;
```