

# SCRIPT VE BATCH KULLANIMI

## 9

SQL Server'da şu ana kadar birçok script hazırladık. Bu script'ler ile nesne oluşturduk (**CREATE**), nesne düzenledik (**ALTER**), veri çektik (**SELECT**). Daha bir çok işlemi, bu script'leri kullanarak gerçekleştirdik.

Bu bölümde, script ve batch kavramları ile **SQLCMD** komut satırı aracını inceleyeceğiz.

## SCRIPT TEMELLERİ

Script kavramı birçok yazılım geliştirme teknolojisinde kullanılan bir terim olmakla birlikte teknik olarak bir dosyaya kaydedilmiş olmayı gerektirir. SQL script'leri metin dosyaları olarak, **.sql** dosya uzantısı ile saklanır.

Script'ler belli görevleri gerçekleştirmek için oluşturulur. Bu nedenle genel olarak tek dosya halinde çalıştırılırlar. Bir script dosyasının içerisindeki kodlar ayrı ayrı çalıştırılmazlar. Script'ler sistem fonksiyonları ve lokal değişkenleri kullanabilirler.

Bir script yapısını kavramak için birçok özelliğin kullanıldığı bir örnek yapalım.

---

```
USE AdventureWorks
GO
DECLARE @Ident INT;

INSERT INTO Production.Location(Name, CostRate, Availability,
ModifiedDate)
VALUES ('Name_Degeri', 25, 1.7, GETDATE());
```

```
SELECT @Ident = @@IDENTITY;
SELECT 'Eklenen satır için identity değeri : '+CONVERT(VARCHAR(3),@Ident);
```

	(No column name)
1	Eklenen satır için identity değeri : 61

Hazırladığımız bu script'i **USE**, **INSERT**, **SELECT**, **CONVERT**, lokal değişken ve sistem fonksiyonu gibi birçok ifade kullanılarak geliştirdik.

## USE İFADESİ

**USE** ifadesi geçerli olması istenen veritabanını seçer. SSMS'in **CTRL+U** kısayolu ile ulaşabildiğiniz Available Databases menüsünden de seçilebilir. SSMS'nin menü ile yaptığı bu işlemin T-SQL karşılığı olarak **USE** ifadesi kullanılır. **USE** ifadesi kullanılmadığında, script çalıştırılırken geçerli olan veritabanında çalıştırılır.

**USE** ifadesini kullanmak zorunlu değildir. Ancak yoğun kod bloklarında hedefi tam belirleyerek herhangi bir yanlış işleme sebebiyet vermemek için kullanılması önerilir. Her veritabanı programcısının birçok kez yaptığı hatalardan biri, geçerli veritabanını belirlemeyi unutmak ve bu nedenle farklı veritabanlarında oluşturulan tablo gibi nesneler ya da daha büyük sorun yaşatabilecek işlemlerdir. Bu nedenle hazırladığınız script'lerin bir dosya olarak çalıştırılacağını unutmadan **USE** ifadesi kullanılıp kullanılmaması konusunda bir kez daha düşünmelisiniz.

## DEĞİŞKEN BİLDİRİMİ

Değişken bildirimi için **DECLARE** ifadesi kullanılır. Kullanımı basittir.

```
DECLARE @degisken_adi degisken_tipi,
        @degisken_adi degisken_tipi
```

Script temelleri kısmını anlatırken **@Ident** adında bir değişken tanımlamıştık. Şimdi basit bir değişken tanımlı gerçekleştirelim.

```
DECLARE @sayi1 INT;
DECLARE @sayi2 INT;
DECLARE @toplam INT = @sayi1 + @sayi2;
SELECT @toplam;
```

	(No column name)
1	NULL

Yukarıdaki işlemde @sayi1, @sayi2 ve @toplam adında 3 değişken tanımlı yaptık.

---

```
DECLARE @sayi1 INT;
DECLARE @sayi2 INT;
```

---

Değişkenleri tanımlarken varsayılan değer atamak zorunda değiliz. Ancak bu durumda yukarıdaki script'i çalıştırdığımızda **NULL** değerinin döndüğünü göreceksiniz. Çünkü değer ataması yapılmayan değişkenler **NULL**'dür.

Değişken tanımlı sırasında değer atayarak tekrar çalıştıralım.

---

```
DECLARE @sayi1 INT = 12;
DECLARE @sayi2 INT = 12;
DECLARE @toplam INT = @sayi1 + @sayi2;
SELECT @toplam;
```

---

(No column name)	
1	24

Değer atayarak çalıştırdığımız sorgu sonucunda dönen değer 24 olacaktır.

Değişken tanımlarken her değişken tanımını **DECLARE** ile ayrı ayrı yapmak genel kullanılan yöntemdir. Ancak tek satır ve tek **DECLARE** kullanarak yan yana, aralarına virgül koyarak da değişken tanımlanabilir.

Veritabanı programlamada değişkenler hayati öneme sahiptir. Kullanımı sırasında hazırladığınız sorguda değişkenden gelen değer sizin istediğiniz sonuç olmama durumunu hesap etmelisiniz. Integer bir değer beklediğiniz bir değişkenden ön göremediğiniz bir sebepten ya da hata yönetimini doğru gerçekleştiremediğiniz için dönecek bir **NULL** değeri, sorgunuzun hata vermesine sebep olacaktır.

## SET İFADESİ KULLANILARAK DEĞİŞKENLERE DEĞER ATANMASI

**SET** ifadesi, T-SQL'de tanımlanan değişkenlere değer atamak için kullanılır.

KDV oranı hesaplayan iki değişken oluşturalım. Bu örnekte KDV değerini nümerik olarak değişken kullanmadan veriyoruz.

---

```
SET @Fiyat = 10;
SET @KdvMiktar = @Fiyat * 0.18;
```

---

Aynı işlemi gerçekleştiren örneği KDV değerini değişken ile belirtmek yapalım.

```
SET @Fiyat = 10;
SET @KdvOran = 0.18;
SET @KdvMiktar = @Fiyat * @KdvOran;
```

Şimdi bu işlemi gerçek örnek ile gerçekleştirelim. Örneğimizde değişkenleri tanımlayıp, gerçek değerler atayarak ekranda görüntüleyeceğiz.

```
DECLARE @Fiyat MONEY;
DECLARE @KdvOran MONEY;
DECLARE @KdvMiktar MONEY;
DECLARE @Toplam MONEY;
SET @Fiyat = 10;
SET @KdvOran = 0.18;
SET @KdvMiktar = @Fiyat * @KdvOran;
SET @Toplam = @Fiyat + @KdvMiktar;
SELECT @Toplam;
```

(No column name)	
1	11,80

**DECLARE** ile tanımladığımız değişkenleri tek satırda da tanımlanabilir.

```
DECLARE @Fiyat MONEY; @KdvOran MONEY, @KdvMiktar MONEY, @Toplam MONEY;
```

Değişkene değer atamayı veritabanındaki veriler üzerinde de gerçekleştirebiliriz.

**Production.Product** tablomuzdaki **ListPrice** sütunu ürünlerin fiyatını tutmaktadır. Bu fiyatlar arasında en yüksek değere sahip olanı değişkenler yardımı ile bulalım.

```
DECLARE @EnYuksekFiyat MONEY;
SET @EnYuksekFiyat = (SELECT MAX(ListPrice) FROM Production.Product);
SELECT @EnYuksekFiyat AS EnYuksekFiyat;
```

EnYuksekFiyat	
1	3578,27

Yukarıdaki işlemin sonucunun doğruluğu, şu şekilde test edilebilir.

```
SELECT ListPrice FROM Production.Product ORDER BY ListPrice DESC;
```

## SELECT İFADESİ KULLANILARAK DEĞİŞKENLERE DEĞER ATANMASI

**SELECT** ifadesi değişkenlerden gelen değerlerin atanmasında kullanılır. Bu ifadeyi bir önceki **SET** ifadesindeki örneklerle inceleyelim.

---

```
DECLARE @EnYuksekFiyat MONEY;
SELECT @EnYuksekFiyat = MAX(ListPrice) FROM Production.Product;
SELECT @EnYuksekFiyat AS EnYuksekFiyat;
```

---

	EnYuksekFiyat
1	3578,27

**SET** kullanımına göre **SELECT**'in daha sade, anlaşılır ve az kod ile geliştirildiğini fark etmiş olmalısınız. Ancak bunun böyle olması her zaman **SELECT** ile değişken ataması gerçekleştireceğiniz anlamına gelmez. İkisinin de gerekli olduğu farklı özellikleri ve kullanım alanları vardır.

### SET ya da SELECT kullanımına nasıl karar vereceğiz?

Örneklerden de anlayacağınız gibi **SET** kullanımında değişken değerlerinin belli olduğu örneklerde kullanırken, **SELECT** işleminde veritabanından veri çekerken kullandık. Bu her zaman böyle olmayacaktır. Ancak genel kullanım açısından bu durum belirgin bir özelliktir.

- Veritabanından sorgu sonucu elde edilen değişken değer atamaları için **SELECT** kullanın.
- Değişkenden değişkene ya da değeri bilinen bir değer ataması için **SET** kullanın.

## BATCH'LER

Batch, T-SQL ifadelerinin tek bir mantıksal birim içinde gruplandırılmasıdır. Batch içindeki tüm ifadeler bir uygulama planı içerisinde birleştirilir, bir arada derlenir ve söz diziminin doğruluğu onaylanır. Derleme sırasında hata meydana gelirse hiçbir ifade çalışmayacaktır. Ancak çalışma zamanında meydana gelen hatadan önce çalışan ifadelerin sonuçları geçerlidir.

Önceki bölümde gördüğümüz script'lerin her biri bir batch oluşturur. Bir script'i birden fazla batch'e ayırmak için **GO** ifadesini kullanırız.

## BATCH'LERİ NE ZAMAN KULLANIRIZ?

Batch'ler, script'te bazı şeylerin script'teki diğer ifadelerden önce ya da farklı olarak uygulanması gerektiğinde kullanılır.

Batch içinde yer alması gereken ifadeler;

- **CREATE VIEW**
- **CREATE PROCEDURE**
- **CREATE RULE**
- **CREATE DEFAULT**
- **CREATE TRIGGER**

Batch yönetimini hatasız ve mantıksal olarak gerçekleştirebilmek için batch içerisindeki ifadeleri doğru kullanmalısınız. Bir batch'de **DROP** ile nesne silmek istiyorsanız, bu ifadeyi **CREATE** ifadesinden önce kullanmalısınız. **DROP** ifadesini ayrı olarak ve **CREATE** ifadesinden önceki bir önceki batch olarak çalıştırmanız gerekir. Çünkü **DROP** ile sildiğiniz bir nesneyi **CREATE** ile tekrar oluşturmak istediğinizde, nesne isminin aynı olması durumunda olası bir hatadan kurtulmuş olursunuz.

## GO İFADESİ

go komutu, bir T-SQL komutu değildir. SQL Server'ın **SSMS** ve **SQLCMD** gibi geliştirme ve düzenleme araçları tarafından tanınan bir komuttur. Bu araçlar haricindeki üçüncü-parti (*third-party*) araçlar go ifadesini desteklemeyebilir.

## GO İFADESİ NE İŞE YARAR?

Geliştirme araçlarında hazırlanan T-SQL kodlar veritabanı sunucusuna gönderilmek istendiğinde, go komutu yazılarak sorgunun çalıştırılmaya hazır olduğu ve işleme alınabileceğini belirtir.

Geliştirme aracı, go ifadesi ile karşılaştığında, batch'i sonlandırır ve paketleyerek tek bir birim olarak sunucuya gönderir. Ancak, sunucuya gönderilen T-SQL sorguları içerisinde go ifadesi yer almaz. Veritabanı sunucusu go ifadesinin ne anlama geldiğini dahi bilmez.

## SQLCMD

SQLCMD, T-SQL script'lerini Windows komut ekranında çalıştırmak için kullanılan bir yardımcı SQL Server aracıdır. Toplu sorguların bulunduğu **.sql** dosyalarının çalıştırılması için iyi bir araçtır. Genel olarak bakım, yedekleme, veritabanı oluşturma ya da benzeri toplu sorguların bulunduğu **.sql** dosyalarının çalıştırılması için kullanıldığından dolayı, SQL Server veritabanı yöneticileri daha çok tercih etmektedir.

**SQLCMD**, eski versiyonu olan **OSQL**'in yerini alması için geliştirilmiştir.

### SQLCMD Söz Dizimi:

---

Sqlcmd

```
[ -U login id ] [ -P password ] [ -S server ] [ -H hostname ]
[ -E trusted connection ] [ -d use database name ] [ -l login timeout ]
[ -N encrypt connection ] [ -C trust the server certificate ]
[ -t query timeout ] [ -h headers ] [ -s colseparator ] [ -w screen width ]
[ -a packetsize ] [ -e echo input ] [ -I Enable Quoted Identifiers ]
[ -c cmdend ] [ -L[c] list servers[clean output]] [ -q "cmdline query" ]
[ -Q "cmdline query" and exit ] [ -m errorlevel ] [ -V severitylevel ]
[ -W remove trailing spaces ] [ -u unicode output ]
[ -r[0|1] msgs to stderr ] [ -i inputfile ] [ -o outputfile ]
[ -f <codepage> | i:<codepage>[,o:<codepage>]]
[ -k[1|2] remove[replace] control characters ]
[ -y variable length type display width ]
[ -Y fixed length type display width ]
[ -p[1] print statistics[colon format]]
[ -R use client regional setting ] [ -b On error batch abort ]
[ -v var = "value"... ]
[ -X[1] disable commands[and exit with warning]]
[ -? show syntax summary ]
```

---

SQLCMD sorgu ekranını açmak için;

- **Başlat -> Çalıştır -> cmd**
- **Başlat -> Çalıştır -> sqlcmd**

... yazarak giriş yapabileceğiniz gibi, sqlcmd sorgularınızı direkt olarak **Çalıştır** kısmında da yazabilirsiniz. **Direkt Çalıştır** kısmında tüm sorguyu yazacaksanız, sorgunun başında SQLCMD yazmış olmalısınız.

Komut ekranında `sqlcmd /?` yazarak komut satırının desteklediği SQLCMD sorgu parametrelerini görebilirsiniz.

```
C:\Users\dijibil>sqlcmd /?
Microsoft (R) SQL Server Command Line Tool
Version 11.0.2100.60 NT x64
Copyright (c) 2012 Microsoft. All rights reserved.

usage: Sqlcmd [-U login id] [-P password]
[-S server] [-H hostname] [-E trusted connection]
[-M Encrypt Connection] [-C Trust Server Certificate]
[-d use database name] [-l login timeout] [-t query timeout]
[-h headers] [-s colseparator] [-w screen width]
[-a packetsize] [-e echo input] [-I Enable Quoted Identifiers]
[-c cmdend] [-Llc] list servers[Clean output]
[-q "cmdline query"] [-Q "cmdline query" and exit]
[-m errorlevel] [-V severitylevel] [-W remove trailing spaces]
[-u unicode output] [-r[0:1] msgs to stderr]
[-i inputfile] [-o outputfile] [-z new password]
[-f <codepage> | i:<codepage>[.o:<codepage>]] [-Z new password and exit]
[-k[1:2] remove[replac] control characters]
[-y variable length type display width]
[-Y fixed length type display width]
[-p[1] print statistics[Colon format]]
[-R use client regional setting]
[-K application intent]
[-M multisubnet failover]
[-b On error batch abort]
[-v var = "value"...] [-A dedicated admin connection]
[-X[1] disable commands, startup script, environment variables [and exit]]
[-x disable variable substitution]
[-? show syntax summary]

C:\Users\dijibil>
```

SQLCMD üzerinden sunucuya bağlanmak için bir çok farklı parametre kullanılabilir.

Bu parametrelerden bazıları;

- **S** : Sunucu adı
- **U** : Kullanıcı adı
- **P** : Kullanıcı parolası
- **H** : Host adı
- **E** : Güvenilir Bağlantı (*Trusted Connection*)

SQLCMD komut satırındaki parametrelerin küçük büyük harf duyarlı olduğunu bilmeniz gerekir. Bu kısa isimlendirmeler haricinde de bir çok parametre kullanılabilir. Ancak temel olarak bunları kullanarak sunucuya bağlanabilirsiniz.

Bir bağlantı örneği;

---

```
sqlcmd -S DIJIBIL-PC
```

---

```
C:\Users\dijibil>sqlcmd -S DIJIBIL-PC
1>
```

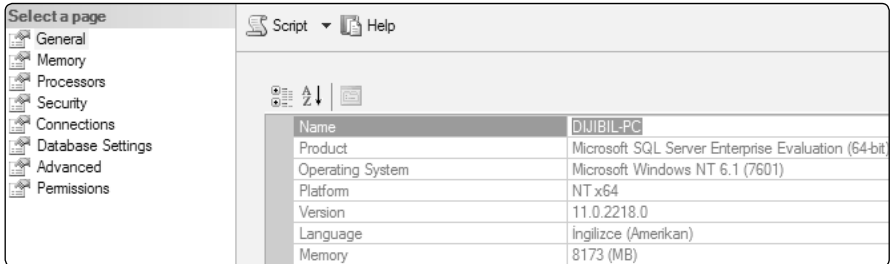
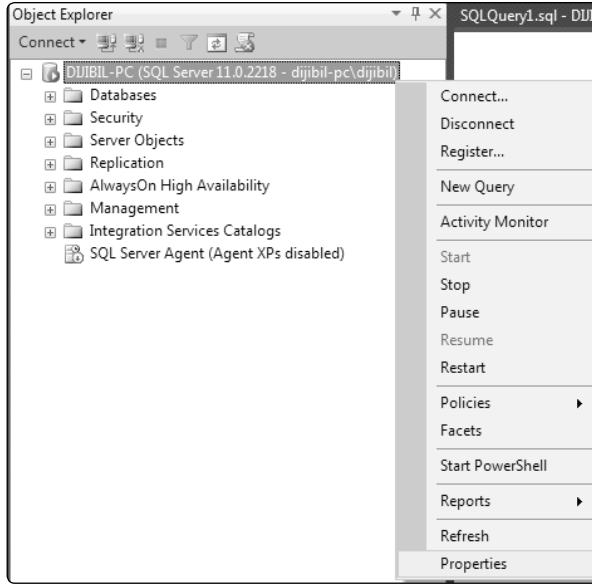


Bu sorgu ile kullanıcı adı ve şifresi kullanmadan en basit şekilde bulunduğunuz bilgisayarındaki SQL Server'a bağlanabilirsiniz.

Bilgisayarımda sunucu adı olarak **DIJIBIL-PC** kullanıldığı için sorguyu buna göre hazırladım. Siz, kendi bilgisayarınızdaki sunucu adını şu şekilde öğrenebilirsiniz.

- SSMS içerisinde Solution Explorer'da, soldaki ağaç yapısının en üstteki sunucu ismini kullanabilirsiniz.

Sunucu adı biraz uzun ise, hata yapmamak için, **Solution Explorer**'daki bu sunucu adına sağ tıklayarak **Properties** ekranında en yukarıda **Name** özelliğinden kopyalayabilirsiniz.



Sunucuya kullanıcı adı ve şifresi ile bağlanmak için;

```
sqlcmd -Ssunucu_ismi123 -Uuser_name123 -Ppwd123
```

Bu şekilde, parametreler ile değerleri arasında boşluk olmayacak şekilde de SQ Server'a bağlanılabilir.

Sunucuya bağlandık. Şimdi ilk sorgularımızı yazarak sunucu hakkında bilgiler edinelim.

Sunucu adını listeleyelim.

```
SELECT @@SERVERNAME;  
GO
```

(No column name)	
1	DIJIBIL-PC

Sunucu versiyonunu listeleyelim.

```
SELECT @@VERSION;  
GO
```

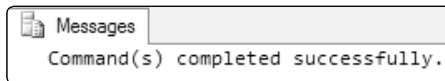
(No column name)	
1	Microsoft SQL Server 2012 - 11.0.2218.0 (X64) Jun 12 2012 13:05:25 Copyright (c) Microsoft Corporation Enterprise Evaluation Edition (64-bit) ...

Sunucu ve sorgularımız başarılı bir şekilde çalışıyor. Artık **AdventureWorks** veritabanımıza erişerek, istediğimiz sorguları çalıştırabiliriz.

Öncelikle, **SQLCMD** içerisinde **AdventureWorks** isimli veritabanına bağlanarak oturum açmalıyız.

```
USE AdventureWorks  
GO
```

Bu sorguyu çalıştırdığınızda ekranda şu mesaj belirecektir.



Bu mesaj ile birlikte, artık ilgili veritabanına bağlanmış durumdasınız. Tüm sorgularınız, belirtilen veritabanı içerisinde çalıştırılacaktır.

Kendi bilgisayarımda, **AdventureWorks** veritabanının 2012 versiyonunu, **AdventureWorks2012** adı ile kullandığım için, bu ismi belirtmem gerekiyordu. Siz hangi veritabanı ile çalışıyorsanız o veritabanının ismini belirtmelisiniz.

Ürünler tablosundaki **ProductID** değeri 1 olan kaydı listeleyelim.

---

```
SELECT ProductID, Name
FROM Production.Product
WHERE ProductID = 1;
```

---

	ProductID	Name
1	1	Adjustable Race

---

**SQLCMD** ekranından çıkarak, tekrar Windows komut satırı ekranına dönmek için, bu iki komut kullanılabilir.

- **quit**
- **exit**

## AKIŞ KONTROL İFADELERİ

Tüm programlama dillerinde bulunan akış kontrolleri, programsal akışı şartları sağlaması, akışın değiştirilmesi, belli görevin belli şartlarda gerçekleşmesi ya da belli görevlerin belirli defa gerçekleşmesi gibi işlemler için kullanılır. **Profesyonel** veritabanı uygulamalarında, akış kontrollerinin kullanılması kaçınılmaz gerekliliktir.

Akış kontrolleri ve yardımcı ifadeleri;

- **IF ... ELSE**
- **CASE**
- **WHILE**
- **WAITFOR**
- **GOTO**

Programlama dillerinden herhangi birinde tecrübesi olan geliştiriciler bu akış kontrollerini belki yüzlerce ya da binlerce kez kullanmış olabilir. T-SQL'de de amaçları aynı olmakla birlikte sadece söz diziminde değişiklikler vardır.

## IF ... ELSE

Tüm programlama dillerinde var olan ve en çok kullanılan akış kontrol ifadelerinden biridir. Durum kontrolü için kullanılan **IF ... ELSE** akış kontrol yapısı tek bir şartı kontrol edebileceği gibi birçok şartı da kontrol edebilir.

### Söz Dizimi:

---

```
IF <Boolean Deyimi>
    <SQL İfadesi> | BEGIN <kodlar> END
[ELSE
    <SQL İfadesi> | BEGIN <kodlar> END]
```

---

IF kullanımı için basit bir örnek yapalım.

---

```
DECLARE @val INT;
SELECT @val = COUNT(ProductID) FROM Production.Product;
IF @val IS NOT NULL
    PRINT 'Toplam ' + CAST(@val AS VARCHAR) + ' kayıt bulundu.';
```

---



Yukarıdaki sorguda, **val** isimli **INT** veri tipinde bir değişken tanımladık. **SELECT** sorgusunda kullandığımız **COUNT** fonksiyonu ile ürünlerin sayısını toplayarak **val** değişkenine aktarıyoruz. Eğer toplama işlemi sonucunda bir kayıt dönerse, kayıt toplamını ekranda göstermesini istiyoruz.

Bu işlemin gerçekleşebilmesi için bir akış kontrolü yapılması gerekir. **IF** kontrol yapısını kullanarak, **val** değişkeninin **NULL** olup olmadığını öğreniyoruz. Eğer **NULL** ise, bu sorgu sonucunda herhangi bir işlem yapılmayacak. Ancak **NULL** değil ise, **PRINT** komutu ile ekranda kaç kayıt bulunduğunu gösterecektir.

**IF** kontrolünün en çok kullanıldığı durumlardan biri de, yeni bir tablo oluştururken, bu tablonun veritabanında var olup olmadığıdır. Eğer tablo var ise silinecek ve yenisi oluşturulacak, yok ise silme işlemine gerek kalmadan sadece **CREATE** sorgusu çalışacaktır.

Deneme isimli bir tablo veritabanında var ise sil ve yenisini oluştur. Bu isimde bir tablo yok ise, sadece oluştur.

---

```
IF EXISTS (
    SELECT *
    FROM Sys.Sysobjects WHERE ID = Object_ID(N'[dbo].[Deneme]')
    AND OBJECTPROPERTY(ID, N'IsUserTable') = 1
)
DROP TABLE dbo.Deneme;
CREATE TABLE dbo.Deneme
(
    DeneID INT
);
```

---

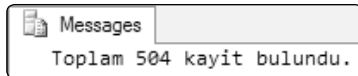
**IF** akış kontrolünde, **ELSE** kullanılarak akış kontrolünde belirtilen şart ya da şartlar sağlanmadığı takdirde çalışacak sorgular oluşturulabilir.

Ürünler tablosundaki kayıtları toplayalım. Eğer kayıt yok ise (yani **NULL** ise), ekranda '*Kayıt Yok*' yazsın, eğer kayıt var ise, bu kayıtları toplayarak, toplam sonucunu veri tipi dönüştürme işleminden sonra bir metinsel değer ile birleştirilerek ekrana yazdıralım.

---

```
DECLARE @val INT;
SELECT @val = COUNT(ProductID) FROM Production.Product
IF @val IS NULL
    PRINT 'Kayıt Yok'
ELSE
    PRINT 'Toplam ' + CAST(@val AS VARCHAR) + ' kayıt bulundu.';
```

---



## ELSE Koşulu

Bazen bir ya da iki koşuldan daha fazlası için **IF ... ELSE** kullanılması gerekebilir. Bu tür durumlarda **ELSE IF** kullanarak koşul sayısı artırılabilir.

Dışarıdan alınan sayısal bir değere göre kontrol yapan bir **IF ... ELSE** tanımlayalım.

---


```

DECLARE @val INT;
SET    @val = 1;

IF @val = 1
    PRINT 'Bir'
ELSE IF (@val = 2)
    PRINT 'İki'
ELSE IF (@val = 3)
    PRINT 'Üç'
ELSE IF (@val >= 4) OR (@val <= 8)
BEGIN
    PRINT '4 - 8 arasında bir değer';
    PRINT '...'
END
ELSE
    PRINT 'Tanımlanamadı.';

```

---

Yukarıdaki sorguda, **IF ... ELSE** akış kontrolünün birçok özelliğini görebilirsiniz. **val** değişkenine aktarılan değeri **IF** blokları içerisinde kontrol ederek şartlara uyan ilk blok içerisindeki kodları çalıştırır.  Messages Bir

- **val** değişkeninin değeri 1 olarak atandığında, ilk **IF** bloğu çalışacak ve **PRINT** ile ekrana *'Bir'* yazacaktır.
- **val** değişkeninin değeri 4 ve 8 dahil, bu sayıların arasında bir değer ise, **BEGIN ... END** blokları arasındaki iki ayrı **PRINT** komutu çalışacaktır.
- Bu şartların hiç biri sağlanmadığı takdirde, en son olarak belirtilen **ELSE** bloğu çalışır.

En son oluşturulan **ELSE IF** bloğu içerisindeki **BEGIN ... END** bloğu dikkatinizi çekmiş olmalı. Bir blok içerisinde kodlama yaparken, blok içerisindeki kod satırı birden fazla olacak ise, bu durumu veritabanı motoruna bildirmek için **BEGIN ... END** blokları içerisine alınması gerekir. **BEGIN ... END** bloklarının sorgu üzerindeki kullanımını kavrayabilmek için bu blok başlangıç ve bitiş komutlarını silerek sorguyu tekrar çalıştırın. **BEGIN** ve **END** komutlarını silerek çalıştırdığınızda, sorgu çalışmayacak ve hata üretecektir.

Bir başka **IF ... ELSE** kuralı ise, hiç bir şartın sağlanmadığı durumda çalışması için kullanılan **ELSE** komutunun sorgunun en sonunda kullanılması zorunluluğudur.

**ELSE** komutunun sırasını değiştirerek herhangi bir sorgu sırasına yerleştirdiğinizde, sorguyu çalıştırmaya gerek kalmadan, sorgunun altında, kırmızı bir hata çizgisi belirecektir. Sorguyu bu şekilde çalıştırmak isterseniz de, bir hata üretilenektir.

**IF** koşulları içerisine, bir sorgu sonucunu değer olarak göndererek koşul sorgulaması yapılabilir.

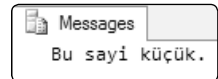
## İÇ İÇE IF KULLANIMI

Bir çok işlemi genel olarak tek bir **IF ... ELSE** bloğu ile çözebiliyor olsak da, bazı durumlarda iç içe birden fazla **IF** kullanılması gerekebilir.

Parametre olarak verilen değeri iç içe **IF ... ELSE** kullanarak sorgulayacağız. Sayının büyüklüğüne göre, ekranda 'küçük', 'büyük' ya da 'ne küçük ne büyük' gibi bir ifade yazdıracağız.

```
DECLARE @sayi INT;
SET @sayi = 5;

IF @sayi > 100
    PRINT 'Bu sayı büyük.';
ELSE
BEGIN
    IF @sayi < 10
        PRINT 'Bu sayı küçük.';
    ELSE
        PRINT 'Ne küçük ne de büyük.';
END;
```



Yukarıdaki sorguda **sayi** değişkenine, çeşitli değer atamaları yapıldığında aşağıdaki sonuçları üretir.

- 10'dan küçük bir değer atandığında, *'Bu sayı küçük'* sonucunu üretir.
- 10 ile 100 sayılarına eşit ya da bu sayılar arasında ise, *'Ne küçük ne de büyük'* sonucunu üretir.
- 100'den büyük bir değer girilirse, *'Bu sayı büyük'* sonucunu üretir.

İç içe işlemler gerçekleştirmek için ihtiyacınız kadar **IF** kullanabilirsiniz.

Yukarıdaki sorguyu biraz daha geliştirerek 3 adet **IF** içeren bir sorgu oluşturalım.

---

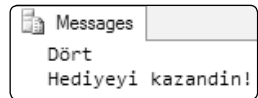
```

DECLARE @sayi INT;
SET @sayi = 4;

IF @sayi > 100
    PRINT 'Bu sayı büyük.';
ELSE
BEGIN
    IF @sayi < 10
        IF @sayi = 1
            PRINT 'Bir'
        ELSE IF (@sayi = 2)
            PRINT 'İki'
        ELSE IF (@sayi = 3)
            PRINT 'Üç'
        ELSE IF (@sayi = 4)
            BEGIN
                PRINT 'Dört'
                PRINT 'Hediyeyi kazandın!'
            END
        ELSE IF (@sayi = 5)
            PRINT 'Beş'
        ELSE
            PRINT 'Bu sayı, 5 ve 10 arasında bir değere
sahip.';
    ELSE
        PRINT 'Ne küçük ne de büyük.';
END;

```

---



**sayı** değişkenine 4 değeri atadığınızda bir hediye kazanacaksınız!

Sorgularda dikkat edilmesi gereken konu, sorgunun basitliği ve işlevselliğidir. Bu tür iç içe sorgular işinizi gördüğü ölçüde işlevsel, anlayabildiğiniz kadar basittir. İç içe sorgular kullanırken açıklama satırları kullanmaya özen göstermelisiniz. En iyi kod, kolay anlaşılabilir olanıdır.



## CASE DEYİMİ

**CASE** ifadesi, programlama dillerinde de var olan akış kontrol özelliklerinden biridir. Akış kontrolü için kullanılan, neredeyse tüm teknikler genel olarak bir diğerinin yapabildiği işlemleri gerçekleştirebilir. Tabi ki aralarında farklılıklar vardır. Performans, kod okunabilirliği, karmaşık sorgular içerisinde kullanılabilirliği gibi seçim yapılması gerektiren durumlar olabilir.

**CASE** ifadesi, iki farklı şekilde kullanılabilir. Bunlar;

- Giriş ifadesi ile
- Boolean bir deyim ile

### SÖZ DİZİMİ (GİRİŞ DEYİMİ)

---

```
CASE <giriş deyimi>
WHEN <when deyimi> THEN <sonuç deyimi>
[...n]
[ELSE <sonuç deyimi>]
END
```

---

Giriş deyimi yöntemi ile kullanımda, **WHEN** koşulunda kullanılan değer ile karşılaştırma yapılacak olan bir giriş deyimi kullanılır.

### SÖZ DİZİMİ (BOOLEAN DEYİMİ)

---

```
CASE
WHEN <Boolean Deyimi> THEN <sonuç deyimi>
[...n]
[ELSE <sonuç deyimi>]
END
```

---

Boolean deyimi yöntemi ile kullanımda, her **WHEN** koşulunda **TRUE** ya da **FALSE** değeri verecek olan bir ifade sağlanır.

Satılan bisikletleri türleri (dağ, gezi vb.) ile birlikte listeleyelim.

---

```
SELECT ProductNumber, Category =
CASE ProductLine
WHEN 'R' THEN 'Yol'
```

```

        WHEN 'M' THEN 'Dağ'
        WHEN 'T' THEN 'Gezi'
        WHEN 'S' THEN 'Diğer Satılıklar'
        ELSE 'Satılık Değil'
    END,
    Name
FROM Production.Product
ORDER BY ProductNumber;

```

	ProductNumber	Category	Name
1	AR-5381	Satılık Degil	Adjustable Race
2	BA-8327	Satılık Degil	Bearing Ball
3	BB-7421	Satılık Degil	LL Bottom Bracket
4	BB-8107	Satılık Degil	ML Bottom Bracket
5	BB-9108	Satılık Degil	HL Bottom Bracket
6	BC-M005	Dag	Mountain Bottle Cage
7	BC-R205	Yol	Road Bottle Cage

Çalışanların görev, cinsiyet ve doğum tarihlerini listeleyelim.

```

SELECT JobTitle, BirthDate, Gender, Cinsiyet =
    CASE
        WHEN Gender = 'M' THEN 'Erkek'
        WHEN Gender = 'F' THEN 'Kadın'
    END
FROM HumanResources.Employee;

```

	Job Title	BirthDate	Gender	Cinsiyet
1	Chief Executive Officer	1963-03-02	M	Erkek
2	Vice President of Engineering	1965-09-01	F	Kadin
3	Engineering Manager	1968-12-13	M	Erkek
4	Senior Tool Designer	1969-01-23	M	Erkek
5	Design Engineer	1946-10-29	F	Kadin
6	Design Engineer	1953-04-11	M	Erkek
7	Research and Development Manager	1981-03-27	M	Erkek

## WHILE DÖNGÜSÜ

**WHILE** ifadesi, tüm programlama dillerinde olduğu gibi, döngüsel olarak bir koşul deyimini test etmek için kullanılır. Sonuç **TRUE** olduğu sürece, döngünün en başına dönerek tekrar test edilir. Sonuç **FALSE** olursa döngüden çıkılır.

### Söz Dizimi:

---

```
WHILE Boolean_Deyim
    { sql_ifadesi | ifade_bloğu | BREAK | CONTINUE }
```

---

Değişken ve **WHILE** döngüsü içeren bir örnek hazırlayalım.

---

```
DECLARE @counter INT
SELECT @counter = 0

WHILE @counter < 5
BEGIN
    SELECT '@counter değeri : ' + CAST(@counter AS VARCHAR(1))
    SELECT @counter = @counter + 1
END;
```

---

	(No column name)
1	@counter değeri : 0
	(No column name)
1	@counter değeri : 1
	(No column name)
1	@counter değeri : 2
	(No column name)
1	@counter değeri : 3
	(No column name)
1	@counter değeri : 4

**WHILE** gibi döngü ifadeleri, bazen bir işlemi tekrarlayarak gerçekleştirmek için farklı amaçlarda kullanılabilir. Örneğin; bir tablo oluşturarak bu tabloya çok fazla sayıda test kaydı girilmesini istenebilir. Bu işlemi tek tek yapmak çok zaman alacaktır. Ancak bir döngü oluşturarak bir tabloya çok sayıda kayıt girilebilir.

---

```

CREATE TABLE #gecici(
    firmaID          INT NOT NULL IDENTITY(1,1),
    firma_isim       VARCHAR(20) NULL
);
WHILE (SELECT count(*) FROM #gecici) < 10
BEGIN
    INSERT #gecici VALUES ('dijibil'),('kodlab')
END;
SELECT * FROM #gecici;

```

---

	firmaID	firma_isim
1	1	dijibil
2	2	kodlab
3	3	dijibil
4	4	kodlab
5	5	dijibil
6	6	kodlab
7	7	dijibil
8	8	kodlab
9	9	dijibil
10	10	kodlab

Yukarıdaki sorgu ile birlikte **gecici** adında, geçici bir tablo oluşturulacak ve bu tabloya 10 adet kayıt girilecektir. Bu kayıtlar, **WHILE** döngüsü ile gerçekleştirilecektir.

10 yerine 100 ya da 1000 yazıldığında, **WHILE** döngüsü veri ekleme işlemini 100 ve 1000 kere gerçekleştirerek tabloya o sayıda kayıt ekleyecektir.

## BREAK KOMUTU

**WHILE** döngüsü sonlanmadan döngüden çıkmak için **BREAK** komutu kullanılır. **BREAK** komutu kullanmayı gerektirecek şekilde bir döngü yapısı oluşturulmamalı ve **BREAK** komutu kullanılmamalıdır. Ancak zorunlu hallerde bu komut kullanılmalıdır.

## CONTINUE KOMUTU

**BREAK** komutunun tam tersi bir işlem için kullanılır. Döngü içerisinde nerede olursanız olun, döngünün en başına dönmenizi sağlar.

Bir metni, ekranda istediğimiz kadar tekrar ettirerek yazdıralım.

---

```

DECLARE @counter INT, @counter1 INT;
SELECT @counter = 0, @counter1 = 3;
WHILE @counter <> @counter1
BEGIN
    SELECT CAST(@counter AS VARCHAR) + ' : ' + 'dijibil.com & kodlab.com';
    SELECT @counter = @counter + 1;
END;

```

---

	(No column name)
1	0 : dijibil.com & kodlab.com
	(No column name)
1	1 : dijibil.com & kodlab.com
	(No column name)
1	2 : dijibil.com & kodlab.com

## WAITFOR İFADESİ

Belirli bir saatte belirlenen işlemi gerçekleştirmek için kullanılır.

### Söz Dizimi:

---

```

WAITFOR
    DELAY 'zaman' | TIME 'zaman'

```

---

**WAITFOR** ifadesi, parametre olarak belirtilen sürenin dolmasını bekler. Süre dolduğu anda görevini gerçekleştirir.

## WAITFOR DELAY

**DELAY** parametresi, **WAITFOR** işlemi ile beklenecek süreyi belirler. Saat, dakika ve saniye olarak değer verilebilir. En fazla 24 saat bekleme süresine sahiptir. Süre olarak gün değeri verilemez.

### Söz Dizimi:

---

```

WAITFOR DELAY '01:00'

```

---

1 dakika bekledikten sonra **sp\_helpdb** sistem prosedürünü çalıştıralım.

```
BEGIN
    WAITFOR DELAY '00:01';
    EXECUTE sp_helpdb;
END;
```

---

## WAITFOR TIME

**TIME** parametresi bir zaman belirtmek için kullanılır. Sadece saat cinsinden değer verilebilir. En fazla 24 saat değer verilebileceği gibi gün değeri verilemez.

İlk olarak **WAITFOR** ifadesinden önceki kodlar çalışır, **WAITFOR** ifadesine geldiğinde 11:00'a kadar beklenir. Saat eşleşmesi sağlandığında **WAITFOR** ifadesinden sonraki kodlar çalışır.

### Söz Dizimi:

```
WAITFOR TIME '11:00'
```

---

Belirli bir işlemi, belirlenen süre sonunda gerçekleştirecek bir örnek yapalım.

---

```
DECLARE @counter INT, @counter1 INT;
SELECT @counter = 0, @counter1 = 3;
WHILE @counter <> @counter1
BEGIN
    WAITFOR TIME '11:00'
    SELECT CAST(@counter AS VARCHAR) + ' : ' + 'dijibil.com & kodlab.com';
    SELECT @counter = @counter + 1;
END;
```

---

## GOTO

Kod blokları arasında etiketleme yaparak akış sırasında farklı bir etikete işlem sırası atlatmak için kullanılır. Özel durumlar dışında kullanılan bir özellik değildir. Ancak bazı durumlarda kullanmak gerekebilir.

---

```
DECLARE @Counter int;
SET @Counter = 1;
WHILE @Counter < 10
BEGIN
```

```

SELECT @Counter
SET @Counter = @Counter + 1
IF @Counter = 4 GOTO Etiket_Bir
IF @Counter = 5 GOTO Etiket_Iki
END
Etiket_Bir:
    SELECT 'Etiket 1'
    GOTO Etiket_Uc;
Etiket_Iki:
    SELECT 'Etiket 2'
Etiket_Uc:
    SELECT 'Etiket 3'

```

---

	(No column name)
1	1
	(No column name)
1	2
	(No column name)
1	3
	(No column name)
1	Etiket 1
	(No column name)
1	Etiket 3

