

# T-SQL İLE HATA YÖNETİMİ

12

Programcıların tecrübeli olduğu en önemli durumlardan birisi şüphesiz hata yakalama ve bu hataların doğru şekilde depolanması ile yönetilmesidir. Veritabanı gibi karmaşık yapılardan oluşan bir ortamda hataların merkezi şekilde yönetilmesi gerekir. SQL Server, bu konuda gelişmiş özelliklere sahiptir. Kod blokları içerisinde oluşacak bir hata yönetilebileceği gibi, programcı tarafından yeni bir hata mesajı oluşturularak belirlenen durumlarda T-SQL programları içerisinde bu hata fırlatılabilir.

Aynı zamanda, bir hata yönetimi tablosunda depolanacak hata bilgilerine ait veriler, veritabanı katmanında merkezi yönetimi sağlayabileceği gibi, bu hatalar işletim sistemi loglarına kaydedilerek sistem yöneticisine de hata verilerini raporlanabilir.

## HATA MESAJLARI

Programlar içerisinde oluşan hataların düzenlenerek hatalar sonucu programların kilitlenmesi, işlemlerin gerçekleşmemesi gibi sorunların önüne geçmek için hatanın olduğu kod blokları içerisinde bu düzenlemelerin yapılması gerekir.

Örneğin, veritabanına bir kayıt eklerken, **INT** veri tipinde beklenen kaydın **VARCHAR** veri tipinde gönderilmesi gerçekleşme olasılıkları yüksek bir hatadır. Bu duruma hazırlıklı olmak için program içerisinde ya da veri girişi sırasında koşullar oluşturularak veriler filtrelenmelidir.

Bir veri ekleme işlemi gerçekleştirerek oluşabilecek hatayı inceleyelim.

**Production.Product** tablosundaki bir çok alan, veri girişi sırasında belirtilmesi gereken, boş geçilemez, zorunlu alanlardır. Tüm zorunlu alanları belirtmeden, sadece iki sütun değerini belirtelim.

---

```
INSERT INTO Production.Product (Name, ProductNumber)
VALUES ('Test Ürün', 'AR-5388');
```

---

Sorgu sonucunda oluşacak hata aşağıdaki gibidir.

```
Msg 515, Level 16, State 2, Line 2
Cannot insert the value NULL into column 'SafetyStockLevel', table
'AdventureWorks2012.Production.Product'; column does not allow
nulls. INSERT fails.
```

SQL Server, bu hata mesajı ile **NULL** geçilemez alanların **NULL** bırakılarak bir kayıt ekleme işlemi gerçekleştirilmek istendiğini bildirir. Hata mesajı açıklaması da buna göre yazılmıştır.

Hata mesajında, açıklama kısmından daha önemli olan kısım, Msg 515 yazılı, hata mesajının ID değerinin belirtildiği kısımdır.

Bu uyarı, **NULL** veri ekleme ile ilgili hataların 515 hata mesaj ID değeri ile tanımlandığını ve veritabanında hata mesajları tablosunda bu şekilde tutulduğunu belirtir.

Hata mesajına bakarsak, belirli bir şablon kullanıldığı aşıkardır. Peki, bu hata mesajı şablonlarına nasıl ulaşabiliriz?

Örneğin; **NULL** ile ilgili işlemlerin hata kodu olan 515 no'lu hata koduna nasıl ulaşabileceğimize bakalım.

Bir sonraki **Mesajları Görüntülemek** isimli başlıkta detaylıca incelenecek olan hata mesajlarının tutulduğu **sys.messages** sistem view'inde, bize gösterilen 515 no'lu hatayı bulalım.

SQL Server içerisindeki tüm hatalar **sys.messages** view'i içerisinde listelenir.

Bu view'i listeleyelim.

```
SELECT * FROM sys.messages;
```

	message_id	language_id	severity	is_event_logged	text
1	50001	1033	10	1	Geçerli bir ürün numarası giriniz
2	50002	1033	11	1	%d adet ürün %s kullanıcısı tarafından silindi.
3	50005	1033	16	0	Şu anki veritabanı ID değeri: %d ve veritabanı adı: ...
4	21	1033	20	0	Warning: Fatal error %d occurred at %S_DATE. No...
5	101	1033	15	0	Query not allowed in Waitfor.
6	102	1033	15	0	Incorrect syntax near '%.1s'.
7	103	1033	15	0	The %S_MSG that starts with '%.1s' is too long. Ma...

515 no'lu hata mesajını bulalım.

```
SELECT * FROM sys.messages WHERE message_id = 515;
```

	message_id	language_id	severity	is_event_logged	text
1	515	1033	16	0	Cannot insert the value NULL into column '%.1s', table '%.1s'; column does ...
2	515	1031	16	0	Der Wert NULL kann in die %1!-Spalte, %2!-Tabelle nicht eingefügt werden...
3	515	1036	16	0	Impossible d'insérer la valeur NULL dans la colonne '%1!', table '%2!'. Cette ...
4	515	1041	16	0	テーブル '%2!' の列 '%1!' に値 NULL を挿入できません。この列では NULL 値...
5	515	1030	16	0	Værdien NULL kan ikke indsættes i kolonnen '%1!', tabellen '%2!'. Kolonne...
6	515	3082	16	0	No se puede insertar el valor NULL en la columna '%1!', tabla '%2!'. La colu...
7	515	1040	16	0	Impossibile inserire il valore NULL nella colonna '%1!' della tabella '%2!'. La ...

22 kayıt listelendi. Bunlar incelendiğinde, her satırdaki verinin text sütununda farklı dillerde kayıtlar olduğu görülebilir.

İlk sırada İngilizce, sonra Almanca ve sonrasında diğer dillerle devam eden hata mesajları listeleniyor. Bu hatalardan sadece İngilizce olanını listeleyelim.

```
SELECT * FROM sys.messages WHERE message_id = 515 AND language_id = 1033;
```

	message_id	language_id	severity	is_event_logged	text
1	515	1033	16	0	Cannot insert the value NULL into column '%.1s', table '%.1s'; column does not allow nulls. %!s fails.

**language\_id** değeri, aynı hata mesajlarının farklı dillerde açıklamalara sahip olması için kullanılır.

Bize gösterilen hata kodu, dil ayarlarımızdan dolayı İngilizce idi. Ancak aynı hatanın Türkçe açıklamasına da programlama yolu ile sahip olabiliriz.

SQL Server tarafından bize gösterilen **NULL** işlem hata mesajını, 1033 **language\_id** değeriyle İngilizce açıklamalı olarak elde etmiştik.

Aynı işlemin Türkçe açıklamasını 1055 `language_id` değeriyle elde edebiliriz.

```
SELECT * FROM sys.messages WHERE message_id = 515 AND language_id = 1055;
```

	message_id	language_id	severity	is_event_logged	text
1	515	1055	16	0	NULL değeri '%2!' tablosunun '%1!' sütununa eklemeyiz; sütun null değerlere izin vermiyor. %3! başarısız.

Hata mesajı şablonu, İngilizce ve Türkçe hata mesajlarını karşılaştıralım.

### İngilizce hata şablonu;

```
Cannot insert the value NULL into column '%.*ls', table '%.*ls';
column does not allow nulls. %ls fails.
```

### İngilizce hata mesajı;

```
Cannot insert the value NULL into column 'SafetyStockLevel', table
'AdventureWorks2012.Production.Product'; column does not allow
nulls. INSERT fails.
```

### Türkçe hata şablonu;

```
NULL değeri '%2!' tablosunun '%1!' sütununa eklemeyiz; sütun null
değerlere izin vermiyor. %3! başarısız.
```

### Türkçe hata mesajı;

```
NULL değeri 'AdventureWorks2012.Production.Product' tablosunun
'SafetyStockLevel' sütununa eklemeyiz; sütun null değerlere izin
vermiyor. INSERT başarısız.
```

Bir programcının T-SQL programları geliştirirken hata mesajlarının mantığını kavraması gerekir. Bu hata mesajlarını, sadece SQL Server kullanmaz. Bazı durumlarda, büyük projelerin hatalarını yönetmek ve hata testleri yapmak zor olabilir. Bazen bir hatayı test etmek için farklı olasılıklar hesaplamak ve hesaplanmış hatalar oluşturmak da gerekebilir. Bu tür durumlarda kendi hata yönetim katmanınızı oluşturmanız gerekecektir.

`sys.messages`'deki hata kodlarını, dil kodları ile birlikte kullanarak, gerçekleşecek hatalarda, istediğiniz istemciye, istediğiniz dili kullanarak hata mesajlarını gösterebilirsiniz.

Hata şablonlarında bulunan %s, %1!, %2!, %3! ya da benzeri parametreler yer tutucu olarak kullanılır. Şablon içerisinde belirlenen bu yerlere ilgili hatanın tanımlayıcı bilgileri yerleştirilerek kullanıcıya gösterilir.

## MESAJLARI GÖRÜNTÜLEMEK

Hata yönetimi kavramına en iyi örnek SQL Server'ın kendi nesneleri ve işlemleri için kullandığı hataların açıklamalarıyla birlikte yer aldığı `sys.messages` sistem view'idir.

`sys.messages` sistem view'i, sistem prosedürleri, fonksiyonlar ve diğer SQL Server hatalarının açıklamaları ve dil gibi bilgileri içeren veri kaynağıdır. Bu view'deki hata mesajları SQL Server'ın gelişmesiyle birlikte sürekli artacaktır. Şu an `sys.messages`'de toplam 230.000 civarında hata mesajı vardır.

Bu hatalar, çalışma ve işlemler sırasında meydana gelen hataların kullanıcıya gösterilmesi için kullanılır.

---

```
SELECT * FROM sys.messages;
```

---

	message_id	language_id	severity	is_event_logged	text
1	50001	1033	10	1	Geçerli bir ürün numarası giriniz
2	50002	1033	11	1	%d adet ürün %s kullanıcısı tarafından silindi.
3	50005	1033	16	0	Şu anki veritabanı ID değeri: %d ve veritabanı adı: ...
4	21	1033	20	0	Warning: Fatal error %d occurred at %S_DATE. No...
5	101	1033	15	0	Query not allowed in Waitfor.
6	102	1033	15	0	Incorrect syntax near '%s'.
7	103	1033	15	0	The %S_MSG that starts with '%s' is too long. Ma...

**Sys.Messages View İçerisindeki Sütunlar:**

### Message\_ID

- **0 - 49.999:** Sistem hata mesajı kodları için ayrılmıştır.
- **50.000:** `RAISERROR` fonksiyonu ile üretilen anlık hata mesajları için ayrılmıştır.
- **50.001 - ... :** Kullanıcı tanımlı mesajlar için ayrılmıştır.

### Language\_ID

Sistemdeki dil grup kodu. Hatanın hangi dilde olduğunu gösterir. (1033 = İngilizce)

## Severity

- **1 - 10:** Kullanıcıdan kaynaklanan bilgi içerikli hatalardır.
- **11 - 16:** Kullanıcının **TRY/CATCH** bloğu ile yönetebileceği hatalardır.
- **17:** Disk ya da diğer kaynakların tükendiği durumlarda oluşur. **TempDB**'nin dolu olması gibi. **TRY/CATCH** blokları ile yakalanabilir.
- **18:** Kritik, dahili ve sistem yöneticisini ilgilendiren hatadır.
- **19:** **WITH LOG** özelliğinin kullanılması gerekir. Hata NT ya da Windows Event Log'da gösterilecektir. **TRY/CATCH** bloğu ile yakalanabilir.
- **20 - 25:** Tehlikeli hatalardır. Kullanıcı bağlantısı sonlandırılır. **WITH LOG** uygulanması gerekir. Event Log'da görüntülenebilir.

## Is\_Event\_Logged

- **0 - 1:** Bu türden bir hata oluştuğunda loglanıp loglanmayacağını belirler.

## Text

Hata mesajıdır. %s ve %d ile dışarıdan parametreler verilebilir.

# YENİ MESAJ EKLEMEK

Hata mesajlarının bulunduğu **sys.messages** sistem view'ine programcılar tarafından da yeni hata mesajları eklenebilir. Bu durum, SQL Server'ın hata mesajlarında daha esnek olabilmesini sağlar.

SQL Server, ilk 50.000 hata mesajını kendisi belirler ve kendisi için ayırmıştır. Kullanıcılar 50.001'den itibaren yeni bir hata mesajı ekleyebilir.

Sisteme kayıtlı hata mesajları 50.000 değerine kadar olan değerleri kullanmasa da özel olarak SQL Server'a ayrılmıştır. Kullanılan hata mesajları sondan başa doğru listelenerek şu şekilde görülebilir.

```
SELECT * FROM SYS.Messages ORDER BY Message_ID DESC;
```

	message_id	language_id	severity	is_event_logged	text
1	50005	1033	16	0	Şu anki veritabanı ID değeri: %d ve veritabanı adı: %s.
2	50002	1033	11	1	%d adet ürün %s kullanıcısı tarafından silindi.
3	50001	1033	10	1	Geçerli bir ürün numarası giriniz
4	49913	1033	10	0	The server could not load DCOM. Software Usage Metrics cannot be started without DCOM.
5	49913	1031	10	0	DCOM konnte vom Server nicht geladen werden. Softwarenutzungsmetriken können ohne...
6	49913	1036	10	0	Le serveur n'a pas pu charger DCOM. Impossible de démarrer les mesures d'utilisation des l...
7	49913	1041	10	0	サーバーは DCOM をロードできませんでした。DCOM がないと、ソフトウェアの使用状況メトリッ...

Yeni bir hata mesajı eklemek için kullanılan genel yapı aşağıdaki gibidir.

---

```
sp_addmessage @msgnum = 'mesaj_kod',
    @severity = 'seviye',
    @msgtext = 'mesaj',
    @with_log = 'true' | 'false',
    @lang = 'dil_kod',
    @replace = ''
```

---

Söz dizimindeki parametrelerin bazıları şu anlama gelir.

- **@msgnum:** 50001'den itibaren bir tam sayı.
- **@lang:** Hata mesajının dil kodu. `sys.syslanguages` ya da `master.dbo.syslanguages` ile ulaşılabilir.
- **@with\_log:** `True` ya da `False` değeri alır. `True` ise, hata kodu `RAISERROR` ile çağrıldığında, SQL Server'ın üzerinde çalıştığı işletim sisteminin event view ile görülebilen sistem log'larına mesajı ve oluş zamanının eklenmesini sağlar.
- **@replace:** Yeni bir mesaj eklemek yerine mevcut bir hatanın düzenlenmesi için kullanılır. `@replace` değişkeni değeri `REPLACE` olarak bildirilmesi gerekir.

`sp_addmessage` ile yeni bir hata mesajı ekleyelim.

---

```
sp_addmessage @msgnum = '50006',
    @severity = 10,
    @msgtext = 'Geçerli bir ürün numarası giriniz',
    @with_log = 'true';
```

---

Eklenen hata mesajını görüntüleyelim.

---

```
SELECT * FROM SYS.Messages WHERE Message_ID = 50006;
```

---

	message_id	language_id	severity	is_event_logged	text
1	50006	1033	10	1	Geçerli bir ürün numarası giriniz

`sp_addmessage` sistem prosedürünü çalıştıran uygulamanın `WITH LOG` seçeneğini kullanabilmesi için `sysAdmin` server rolüne sahip olması gerekir.

## PARAMETRELİ HATA MESAJI TANIMLAMAK

Hata yönetimi bölümünün ilk açıklama kısmında ve sonrasında hata fırlatma ile ilgili konularda yüzeysel olarak işlediğimiz parametrelili hata mesajlarını detaylandıracağız.

Hata mesajları için oluşturulan şablonlar içerisinde parametre kullanılarak, dışarıdan hata mesajının belirlenen kısımlarına değer atanması gerekebilir.

Şablonlara dışarıdan parametre atama işlemi için gerekli söz dizimi;

```
'hata mesajı %p1 mesaj devamı %p2 mesaj sonu', parametre1, parametre2
```

Hata mesajlarında kullanılan bu parametrelere yer tutucu denir. Yer tutucular için, veri tiplerine göre değişen farklı işaretler kullanılır.

% karakteri	Veri Tipi Karşılığı
d veya l	Desimal
p	Pointer
s	String
o	Unsigned octal
u	Unsigned integer
x veya X	Unsigned hexadecimal

Yer tutucuları bulunan bir hata mesajı kaydedelim.

---

```
sp_addmessage @msgnum = 50002,  
              @severity = 11,  
              @msgtext = '%d adet ürün %s kullanıcısı tarafından silindi.',  
              @with_log = 'true'
```

---

Bu hata mesajına, hata fırlatma teknikleri kullanılarak içerisindeki yer tutuculara dışarıdan parametre gönderilebilir. Bu işlem ile ilgili örnekleri **Hata Fırlatmak** kısmında inceleyeceğiz.



Ayrıca, yer tutucu özelliklerinin yanında flag ve genişlik bilgisi özellikleri de vardır.

- - (*eksi*) : Sola yanaştırma.
- + (*arti*) : Signed tipinin negatif mi pozitif mi olduğunu belirtir.
- 0 : Genişlik özelliğinde belirlenmiş olan genişlik değerine ulaşılan kadar sayısal değer başına sıfır konulacağını belirtir.
- # (*pound*) : Octal ve hex değerine bağlı olan uygun ön eki (0 ya da 0x) kullanılmasını belirtir. Sadece octal ve hex değerlere uygulanır.
- ' ' : Sayısal değer pozitif ise, sol kısmını boşluk ( ' ') ile doldurur.

Bu özelliklerin yanında, parametrenin genişlik, kısa/uzun durumu ve hassasiyeti de düzenlenebilir.

- **width**: Parametre değerini tutmak için gerekli boş yer miktarını, bir integer değer ile ayarlar.
- \* (yıldız) işareti kullanılırsa genişlik otomatik olarak belirlenir.
- **Precision**: Sayısal veri için, maksimum rakam sayısını belirler.
- **Long/Short**: Parametre integer, octal ya da hex veri tipinde ise, kısa (**h**) ya da uzun (**l**) olarak ayarlanabilir.

## HATA OLUŞTURURKEN KULLANILABİLECEK ÖZELLİKLER: WITH

SQL Server, hata oluştururken kullanılabilecek bazı ek özelliklere sahiptir. Bu özellikler WITH ile birlikte kullanılır.

- LOG
- SETERROR
- NOWAIT

### WITH LOG

SQL Server'ın hata log bilgisini SQL Server log dosyasında ve **Windows Application Log** dosyasında tutmasını sağlar. Hata şiddeti 19 ve üzerinde olan hatalarda kullanılması gerekir. Hata logları maksimum 400 bayt ile sınırlıdır.

Bu özellik için, **sysAdmin** sunucu rolüne ya da **ALTER TRACE** iznine sahip olunmalıdır.

## WITH NOWAIT

İstemciye hemen mesaj gönderir.

## WITH SETERROR

**RAISERROR** komutu çalıştığında, varsayılan olarak **RAISERROR** komutunun başarılı çalışıp çalışmadığını gösteren değeri tutar. **SETERROR** özelliği bu durumu değiştirerek, @@ERROR değişkeninin üretilen hata değerini tutmasını sağlar.

## MESAJ SİLMEK

Kullanıcı tarafından tanımlanan bir mesajı silmek için, **sp\_dropmessage** sistem Stored Procedüre'ü kullanılır.

### Söz Dizimi:

---

```
sp_dropmessage mesaj_no
```

---

Oluşturduğumuz 50001 mesaj no'lu hata mesajını silelim.

---

```
sp_dropmessage 50001
```

---

## OLUŞAN SON HATANIN KODUNU YAKALAMAK: @@ERROR

Sistemde gerçekleşen bir hatanın hata kodu, @@ERROR hata kodu ortam fonksiyonu tarafından yakalanır. Bu fonksiyon, her işlemde değişen bir değere sahiptir. Yeni bir işleme geçtiğinde eski değeri kaybolur ve yeni değeri tutmaya başlar. Hata oluşmadığında 0 değerini tutar. Hata değerini kaybetmemek ve hafızada tutmak istendiğinde hata oluştuğu anda yeni bir kullanıcı tanımlı değişkene atanması gerekir. Aksi halde, yeni hata değerine sahip olduğunda eskisine ulaşamaz.

Sıfıra bölme hatasını @@ERROR ile yönetelim.

---

```

DECLARE @deadline INT, @hataKod INT;

SET @deadline = 0

SELECT DaysToManufacture / @deadline
FROM Production.Product
WHERE ProductID = 921

SET @hataKod = @@ERROR

IF @@ERROR <> 8134
BEGIN
PRINT CAST(@hataKod AS VARCHAR) + ' No''lu sıfıra bölünme hatası.';
END
ELSE IF @@ERROR <> 0
BEGIN
PRINT CAST(@hataKod AS VARCHAR) + ' No''lu bilinmeyen bir hata
oluştı.';
END;

```

---

```

Msg 8134, Level 16, State 1, Line 5
Divide by zero error encountered.
8134 No'lu sıfıra bölünme hatası.

```

Hata sonucunda **IF** koşulunda 8134 no'lu hatayı arayan koşul içerisine girdi ve hata kodu ile birlikte belirttiğimiz bilgiyi ekranda gösterdi.

Bu işlem prosedürel olarak gerçekleştirilebilir. Örneğin, hata kodu ve dil kodunu alan ve sonucunda **sys.messages** içerisinde ilgili hata mesajı kaydını getiren bir prosedür kullanılabilir.

Aldığı parametreler ile hata göstermeyi sağlayacak prosedürü oluşturalım.

---

```

CREATE PROC pr_HataGoster(
    @hataKod INT,
    @dilKod INT
)
AS
BEGIN
    DECLARE @text VARCHAR(100);
    SELECT @text = Text FROM sys.messages
    WHERE message_id = @hataKod AND language_id = @dilKod;
    PRINT @text;
END;

```

---

Prosedürü test edelim.

---

```
EXEC pr_HataGoster 8134, 1055;
```

---

Sifıra bölünme hatasıyla karşılaşıldı.

- **İlk parametre (8134):** Sifıra bölme hatasını temsil eden hata mesaj kodu.
- **İkinci parametre (1055):** Dil kodunu temsil eder. Türkçe hata mesajı dil kodu.

Yukarıda oluşturduğumuz prosedürel olmayan örneği, otomatik hale getirerek prosedür ile tekrar oluşturalım.

---

```
DECLARE @deadline INT, @hataKod INT, @dilKod INT

SET @deadline = 0

SELECT DaysToManufacture / @deadline FROM Production.Product
WHERE ProductID = 921

SET @hataKod = @@ERROR;
SET @dilKod = 1055; -- Türkçe dil kodu

IF @@ERROR <> 8134
BEGIN
    -- Sifıra bölünme hatasının Türkçe açıklamasını getirir.
    EXEC pr_HataGoster @hataKod, @dilKod
END

ELSE IF @@ERROR <> 0
BEGIN
    -- Hangi hata gerçekleşirse o hatanın Türkçe açıklamasını getirir.
    EXEC pr_HataGoster @hataKod, @dilKod
END;


```

---

```
Msg 8134, Level 16, State 1, Line 5
Divide by zero error encountered.
Sifıra bölünme hatasıyla karşılaşıldı.
```

## STORED PROCEDURE İÇERİSİNDE @@ERROR KULLANIMI

Hata yönetimi gerçekleştirilmesi gereken en gerekli program parçacıklarından biri Stored Procedure'dür. Bu programcıklar içerisinde gerçekleşen işlemlerin yönetimi için @@ERROR kullanılabileceği gibi, TRY/CATCH bloğu da kullanılabilir.

@@ERROR ortam fonksiyonu kullanarak farklı Stored Procedure örnekleri gerçekleştirelim.

İş başvuruları gerçekleştiren çalışan adaylarının bilgilerinin tutulduğu **JobCandidate** tablosundan bir kayıt silme işlemi gerçekleştirmek için Stored Procedure geliştirelim. Ayrıca, hata yönetimini gerçekleştirmek için prosedür içerisinde @@ERROR fonksiyonunu kullanalım.

---

```
CREATE PROCEDURE HumanResources.pr_DeleteCandidate(
    @CanID INT
)
AS
DELETE FROM HumanResources.JobCandidate WHERE JobCandidateID = @CanID;
IF @@ERROR <> 0
BEGIN
    -- Başarısız olduğunu göstermek için 99 döndürür.
    PRINT N'Aday silme işleminde bir hata oluştu.';
    RETURN 99;
END
ELSE
BEGIN
    -- Başarılı olduğunu göstermek için 0 döndürür.
    PRINT N'İş adayı silindi.';
    RETURN 0;
END;
```

---

Prosedürü çağırarak **CandidateID** değeri 2 olan kaydı silelim.

```
(1 row(s) affected)
İş adayı silindi.
```

Prosedür başarıyla çalıştığı için herhangi bir hata vermedi ve ekranda işlemin başarılı olduğuna dair bilgi gösterdi. Prosedür, başarılı işlem yaptığını belirtmek için geriye 0 değeri döndürdü.

Ancak prosedür içerisinde bir hata olsaydı ekranda aşağıdaki hata bildirimi gösterilecekti.

Aday silme işleminde bir hata oluştu.

Prosedürün hata verdiğini bildirmek için **RETURN** ile geriye 99 değerini döndürecekti.

## HATA FIRLATMAK

Hata mesajlarına kaydedilen ya da anlık olarak gerçekleşen hataların fırlatılabilmesi için tetiklenmesi gerekir. Bu fırlatma işlemini gerçekleştiren iki özellik vardır. Eski sürümlerden beri desteklenen **RAISERROR** ve SQL Server 2012 ile gelen yenilikler arasında olan **THROW** ifadesidir.

### RAISERROR İFADESİ

Hata mesajlarının devreye girebilmesi için tetiklenmesi yani hatanın fırlatılması gerekir. Bunu gerçekleştiren ifadelerden birisi **RAISERROR** ifadesidir.

**RAISERROR** iki farklı amaç ile kullanılabilir. Bunlar;

- Sistemde var olan hata mesajları ile bir hata meydana getirmek.
- Anlık oluşturulan bir hata mesajı ile bir hata meydana getirmek.

#### Söz Dizimi:

---

```
RAISERROR (mesaj_kod, seviye, durum) [WITH LOG]
```

---

**RAISERROR** parametre açıklamaları;

- **mesaj\_kod**: sys.messages'de yer alan message\_id sütun değerine eşittir.
- **seviye**: Hata mesajının kritiklik seviyesini belirtir. 0-25 arasında değer alabilir.
- **durum**: Bir hata mesajı birden fazla yerde olduğunda, bu yerleri birbirinden ayırt etmek için kullanılır. 1- 127 arasında bir değer alabilir.
- **WITH LOG**: Oluşan hatanın loglara yazılmaması işaretlenerek tanımlanmış bile olsa loglara yazılmasını sağlar.

Anlık olarak bir hata üretelim.

---

```
RAISERROR('Mevcut bir ürünü eklemeye çalışıyorsunuz.', 10, 1);
```

---

```
Mevcut bir ürünü eklemeye çalışıyorsunuz.
```

Ekranda görüntülenen hata mesajının kırmızı olmadığını fark etmiş olmalısınız. Bunun nedeni hata seviyesinin yüksek olmamasıdır.

10 hata seviyesi çok kritik olmadığını belirtirken, aynı hata fırlatma işlemini seviye 16 ile gerçekleştirdiğimizde sonuç farklı olacaktır.

---

```
RAISERROR('Mevcut bir ürünü eklemeye çalışıyorsunuz.', 16, 1);
```

---

```
Msg 50000, Level 16, State 1, Line 1
Mevcut bir ürünü eklemeye çalışıyorsunuz.
```

Artık fırlattığımız hata daha kritik bir hata olarak algılanabilir.

Daha işlevsel ve dinamik değer üreten bir anlık hata oluşturarak fırlatalım. Bu hata mesajında, dinamik olarak değişkenlerden alınacak veritabanı ID ve veritabanı adı bilgilerini hata mesajı ile birlikte fırlatalım.

---

```
DECLARE @DBID INT;
DECLARE @DBNAME NVARCHAR(128);

SET @DBID = DB_ID();
SET @DBNAME = DB_NAME();

RAISERROR
    (N'Su anki veritabanı ID değeri: %d ve veritabanı adı: %s.',
    10, --Şiddet.
    1, --Durum.
    @DBID, --İlk argüman.
    @DBNAME); --İkinci argüman.
```

---

```
Şu anki veritabanı ID değeri: 7 ve veritabanı adı: AdventureWorks2012.
```

Hata mesajında %d ve %s argümanları bir yer tutucu olarak kullanılır. **RAISERROR** içerisinde @DBID ve @DBNAME değişken değerleri, çalışma zamanında alınarak bu yer tutucular ile gösterilecektir.

Yer tutuculara bir yenisini eklemek de kolaydır.

---

```
DECLARE @DBID INT;
DECLARE @DBNAME NVARCHAR(128);

SET @DBID = DB_ID();
SET @DBNAME = DB_NAME();

RAISERROR
    (N'Su anki veritabanı ID değeri: %d ve veritabanı adı: %s. Coder : %s',
    10, --Şiddet.
```

```
1, --Durum.
@DBID, -- İlk argüman.
@DBNAME, -- İkinci argüman.
'Cihan Özhan'); -- Üçüncü argüman.
```

---

Şu anki veritabanı ID değeri: 7 ve veritabanı adı: AdventureWorks2012. Coder : Cihan Özhan

Yer tutucular ile ilgili detaylı anlatımı bu bölümdeki **Parametrelili Hata Mesajları Tanımlamak** isimli kısımda bulabilirsiniz.

Yukarıdaki örneğin aynısını gerçek hata mesajı oluşturma yöntemi ile tekrar yapalım. `sys.messages` içerisine bir hata mesajı ekleyelim. Bu hata mesajını kullanarak yeni bir hata fırlatalım.

---

```
EXECUTE sp_addmessage
    50007,
    10,
    N'Su anki veritabanı ID değeri: %d ve veritabanı adı: %s.';

DECLARE @DBID INT;
SET @DBID = DB_ID();

DECLARE @DBNAME NVARCHAR(128);
SET @DBNAME = DB_NAME();

RAISERROR (50007, 10, 1, @DBID, @DBNAME);
```

---

Şu anki veritabanı ID değeri: 7 ve veritabanı adı: AdventureWorks2012.

Eğer hata mesajının kırmızı yazı ile kritik olarak gösterilmesi isteniyorsa **RAISERROR** ikinci parametresi (seviye) 10 ve üzeri olması gerekir.

## THROW İFADESİ

**RAISERROR** ile benzer özelliklere ve aynı amaca sahiptir. **THROW**, hata fırlatmak için kullanılır.

**THROW** ile anlık bir hata oluşturalım.

---

```
THROW 50001, 'Ürün ekleme sırasında bir hata meydana geldi.', 5;
```

---

```
Msg 50001, Level 16, State 5, Line 1
Ürün ekleme sırasında bir hata meydana geldi.
```



**THROW** ifadesini gerçek veri ve tablo üzerinde çalıştırmak için örnek bir tablo oluşturalım.

---

```
USE tempdb;
GO
CREATE TABLE dbo.Deneme_Tablo
(
    sutun_1 int NOT NULL PRIMARY KEY,
    sutun_2 int NULL
);
```

---

Oluşturulan **Deneme\_Tablo** isimli tabloya veri ekleme işlemi gerçekleştirirken oluşacak bir **PRIMARY KEY** hatasını fırlatacağız. **PRIMARY KEY** olan bir sütuna aynı değere sahip veri eklenemez. Bu durumda bir hata fırlatılması gerekecektir.

---

```
BEGIN TRY
    TRUNCATE TABLE dbo.Deneme_Tablo;
    INSERT dbo.Deneme_Tablo VALUES(1, 1);
    PRINT 'İlk Ekleme Sonrası';
    -- Msg 2627, Level 14, State 1 - PRIMARY KEY kısıtlama ihlali
    INSERT dbo.Deneme_Tablo VALUES(1, 1);
    PRINT 'İkinci Ekleme Sonrası';
END TRY
BEGIN CATCH
    PRINT 'Gerekirse burada istisna işlenebilir ve fırlatılabilir.';
    THROW;
END CATCH;
```

---

```
(1 row(s) affected)
İlk Ekleme Sonrası

(0 row(s) affected)
Gerekirse burada istisna işlenebilir ve fırlatılabilir.
Msg 2627, Level 14, State 1, Line 9
Violation of PRIMARY KEY constraint 'PK__Deneme_T__CCD8D9D094F12A42'.
Cannot insert duplicate key in object 'dbo.Deneme_Tablo'. The duplicate key value is (1).
```

Oluşturulan bloklar içerisinde iki farklı **INSERT** işlemi gerçekleştiriliyor. Ancak, sorgu çalıştığında sadece ilk **INSERT** işlemi başarıyla sonuçlanacaktır. İkinci **INSERT** işlemi, **sutun\_1** sütununa aynı veriyi eklemeye çalışacağı için **PRIMARY KEY** kısıtlama ihlali hatası meydana gelecek ve bu hata **THROW** ile fırlatılacaktır.

Tabloya eklenen tek kaydı listeleyelim.

---

```
SELECT * FROM Deneme_Tablo;
```

---

	sutun_1	sutun_2
1	1	1

## HATA KONTROLÜ VE TRY-CATCH

Eski versiyonlarda, SQL Server programcıları içerisinde hata yönetimi gerçekleştirmek için @@ERROR fonksiyonu kullanılırdı. @@ERROR fonksiyonu bir blok içerisinde, oluşan bir hata sonucunda hatanın değerini tutar. Bir sonraki ifadeye geçtiğinde @@ERROR fonksiyonunun değeri değişecektir. Sonraki ifade de hata olmazsa 0, hata olursa hata kodunu tutmaya başlayacaktır. Yani, @@ERROR fonksiyonu sürekli farklı ifadeler için dinamik bir hata değişkenidir. Çalışan ifadeye hata oluşup oluşmadığına bakar, oluşmadıysa 0 değeri ile sonraki ifadeye geçer ve artık önceki ifadenin değeri kaybolmuştur.

Bu nedenle, @@ERROR fonksiyonunun aldığı hata değerini anında başka bir kullanıcı tanımlı değişkene atamak gerekir.

İş bu kadar sıkıntılıken, çözüm olarak .NET programlama dillerinde sıkça kullanılan **TRY-CATCH** bloğunun T-SQL'e eklenmesiyle daha kullanışlı bir hata yönetimi işlemi gerçekleştirilebilir oldu.

### Söz Dizimi:

---

```
BEGIN TRY
    { t-sql bloğu }
END TRY
BEGIN CATCH
    { t-sql ifadeleri }
END CATCH;
```

---

**TRY** ve **CATCH** bloklarının **BEGIN** ve **END** olarak açılış ve kapanışlarının ayrı ayrı gerçekleştiğini görüyoruz. Eğer **TRY** bloğu kullanıldıysa **CATCH** bloğu da kullanılması gerekir. Tek başına **TRY** bloğu tanımlanamaz.

Programlarda çok karşılaşılan hatalardan biri olan sıfıra bölünme hatasını inceleyelim.

---

```

DECLARE @sayi1 INT = 5
DECLARE @sayi2 INT = 0
DECLARE @sonuc INT

BEGIN TRY
    SET @sonuc = @sayi1 / @sayi2
END TRY
BEGIN CATCH
    PRINT CAST(@@ERROR AS VARCHAR) + ' no lu hata oluřtu'
END CATCH;

```

---

**8134 no lu hata oluřtu**

8134 no'lu hata sıfıra bölünme hatasını temsil eder. Bu hata mesajı numarasını **sys.messages** içerisinde Türkçe açıklaması ile birlikte bulabiliriz.

---

```
SELECT * FROM sys.messages WHERE message_id = 8134 AND language_id = 1055;
```

---

	message_id	language_id	severity	is_event_logged	text
1	8134	1055	16	0	Sıfıra bölünme hatasıyla karşılařıldı.

Yukarıdaki 0'a bölme hatasının detaylarını bir Stored Procedure yardımı ile alabiliriz.

Anlık olarak hata bilgilerini getirecek bir prosedür oluřturalım.

---

```

CREATE PROCEDURE pr_HataBilgisiGetir
AS
SELECT
    ERROR_NUMBER() AS ErrorNumber,
    ERROR_SEVERITY() AS ErrorSeverity,
    ERROR_STATE() AS ErrorState,
    ERROR_PROCEDURE() AS ErrorProcedure,
    ERROR_LINE() AS ErrorLine,
    ERROR_MESSAGE() AS ErrorMessage;

```

---

Prosedörü, hata oluřacak kod bloğuna yerleřtirelim.

---

```

DECLARE @sayi1 INT = 5
DECLARE @sayi2 INT = 0
DECLARE @sonuc INT

```

```
BEGIN TRY
    SET @sonuc = @sayi1 / @sayi2
END TRY
BEGIN CATCH
    EXECUTE pr_HataBilgisiGetir;
END CATCH;
```

	ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
1	8134	16	1	NULL	6	Divide by zero error encountered.

Hata ile ilgili teknik bilgilerin listelenmesi için prosedürü kullandık.

Prosedürde yer alan fonksiyonların açıklamaları şu şekildedir.

- **ERROR\_NUMBER()** : Hatanın `sys.messages` içerisindeki hata kodu.
- **ERROR\_MESSAGE()** : Hata mesajı metni.
- **ERROR\_SEVERITY()** : Hatanın dışa dönük kritiklik durumu.
- **ERROR\_STATE()** : Hatanın sisteme dönük kritiklik seviyesi.
- **ERROR\_PROCEDURE()** : Hataya neden olan prosedür.
- **ERROR\_LINE()** : Hataya neden olan satır.