SQL CURSOR'LARI

10

Cursor'ler, veri kümesini ele alarak her seferinde bir kayıt üzerinde işlem yapabilmeyi sağlayan yöntemdir. Kullanım olarak bir metin editörüne benzer. Metin editöründe imleç (*cursor*) hangi satırda ise, o satır üzerinde işlem yapabilirsiniz. Cursor'lar da aynı şekilde bir veri kümesi üzerinde ileri ve geri giderek satırlar üzerinde işlem yapmayı sağlar.

Cursor, varsayılan olarak sadece ileri doğru işlem yapar. Geriye doğru işlem yapabiliyor olsa da, ana kullanım yöntemi ileriye doğrudur. Ayrıca sadece ileri doğru çalışan Cursor ile, kaydırma yapabilen (ileride göreceğiz) Cursor'lar arasında önemli performans ve hız farkı vardır.

En hızlı Cursor, sadece ileri doğru okuma işlemi yapan Cursor'dür. Sadece ileri doğru okuma yapan Cursor'leri, DOT.NET geliştiricileri için **`SQLDataReader`** nesnesine benzetebiliriz. Her ikisi de sadece ileri doğru okuma yaptıkları ve önceki satırlarla ilgilenmediği için çok hızlıdırlar.

Veritabanında bir **`SELECT`** sorgusu ile alınan kayıtlar üzerinde, döngü yapısı oluşturarak tüm satırları tek tek inceleyebilmek, üzerinde işlem yapabilmek için Cursor programlanmalıdır. Cursor'ler veritabanında saklanmazlar. Cursor'ler kitap içerisinde işlenen Stored Procedure'ler içerisinde kullanılmalıdır.

ISO Söz Dizimi:

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
 FOR select_statement
 [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
[;]
```

Transact-SQL Genişletilmiş Söz Dizimi:

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
 [ FORWARD_ONLY | SCROLL ]
 [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
 [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
 [ TYPE_WARNING ]
 FOR select_statement
 [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[;]
```

Cursor İçerisindeki SELECT Sorgusunun Farkları

Cursor içerisinde oluşturulan **SELECT** sorgusu her ne kadar aynı söz dizimine sahip olsa da Cursor'lere özel bazı farklılıkları vardır.

- Cursor ve sonuç kümesi bildirim sırasında isimlendirilir, daha sonra bu isimler kullanılır.
- Cursor bildirimi uygulamadan ayrı olarak yapılır.
- Cursor, kendi içerisinde açık ve kapalı olma durumuna sahiptir. Siz kapatana kadar Cursor açık kalır.
- Siz silmeden Cursor hafızadan silinmez.
- Cursor, işlevsel yeteneklere sahip olabilmesi için bazı özel komutları içerir.

Cursor'ler Neden Kullanılır?

Cursor'ler veritabanında sürekli kullanmanız gereken bir özellik olmasa da, gerektiği durumlarda birçok faydalı işleve sahiptir. Az da olsa kullanmanız gereken faydalı bir özelliktir.

Cursor'ler genel olarak şu amaç ile kullanılır;

- Bir sorgu sonucunda (resultset) bulunan kayıtlar içerisinde gezinmek, önceki ya da sonraki kayda gitmek, ilk ya da son kayda gitmek istenildiğinde kullanılır.
- Sorgu sonucu dönen değerleri incelemeye tabi tutarak, üzerinde değişiklik yapılması gereken satırlarda güncelleme işlemi yapmak.

- Diğer kullanıcılar tarafından yapılan değişikliklerin görünürlük seviyesini ayarlamak için kullanılır. (Görünürlük seviyesi konusunu daha sonra inceleyeceğiz)
- Trigger ya da Stored Procedure'lerin bir sorgu sonucuna satır satır erişmesini sağlamak.

Cursor'un Ömrü

Cursor birden fazla parçadan oluşur. Cursor'un sistem üzerindeki yaşam döngüsünü öğrenmeden ömrünü kavramak güçtür. Bu nedenle, Cursor ömrünü anlayabilmek için bir Cursor oluşturalım.

Bir Cursor geliştirme kısımları şu şekildedir;

- Bildirim
- Açılış
- Kullanım/Yönlendirme
- Kapanış
- Hafızada Ayrılan Belleği Boşaltmak

Cursor'lerin çalışma modelini kavramak için karmaşık değil, basit bir sorgu ile ilk örneği gerçekleştirmeliyiz.

Ürünler tablosunda **`ProductID`** ve **`Name`** sütunlarını Cursor ile listeleyelim.

```
DECLARE @ProductID INT
DECLARE @Name  VARCHAR(255)
DECLARE ProductCursor CURSOR FOR
   SELECT ProductID, Name FROM Production.Product
OPEN ProductCursor
FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' - ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
CLOSE ProductCursor
DEALLOCATE ProductCursor
```

Sorgu çalıştırıldığında aşağıdaki gibi bir listeleme gerçekleşmiş olmalı.

```
1   -  Adjustable Race
2   -  Bearing Ball
3   -  BB Ball Bearing
4   -  Headset Ball Bearings
316  -  Blade
317  -  LL Crankarm
318  -  ML Crankarm
319  -  HL Crankarm
320  -  Chainring Bolts
321  -  Chainring Nut
322  -  Chainring
```

Cursor başarılı bir şekilde çalıştı. Ancak yukarıdaki komutları henüz incelemedik. Şimdi, oluşturduğumuz Cursor'ü yukarıda belirttiğimiz geliştirme kısımlarına göre sınıflandırarak inceleyelim.

Bildirim

Bir Cursor'ün geliştirme kısımlarında ilk sırada bildirim kısmı olduğunu belirtmiştik. Cursor içerisinde kullanılacak değişkenlerin tanımlandığı, yani bildirimlerinin gerçekleştiği bölümdür. Bu bildirimler, Cursor ile **FETCH** edilecek kayıtların hafızada tutularak Cursor'ün üzerinde bulunduğu kayıtları temsil eder. Cursor satır satır okuduğu veriyi bu değişkenlere aktararak üzerinde işlem yapılabilmesini sağlar.

Cursor'ün kendisi de değişkenler ile birlikte burada tanımlanır. Değişkenler ile Cursor'ün tanımlanması arasındaki tek tanım farkı Cursor'ün `@` işareti ile başlayan bir isimlendirmeye sahip olmamasıdır. Cursor isminden sonra **CURSOR FOR** kullanılarak Cursor'ün kullanacağı veriyi temsil eden sorgu oluşturulur.

```
DECLARE @ProductID INT
DECLARE @Name  VARCHAR(255)
DECLARE ProductCursor CURSOR FOR
   SELECT ProductID, Name FROM Production.Product;
```

Açılış

Cursor oluşturulduktan sonra veri okuma işlemine hazır hale gelmesi için **OPEN** komutu ile açılması gerekir. Açılış kısmı, Cursor'ün **OPEN** ile açıldığı kısmı temsil eder. Açılan Cursor, kaynak tüketimine başlamış demektir. Kapanış kısmında anlattığımız gibi Cursor işlemi sonunda açık Cursor kapatılmalıdır.

```
OPEN ProductCursor
```

Kullanım / Yönlendirme

Cursor işleminin yönetim kısmıdır. Bir Cursor ile yapılması gereken işlemler, verinin yönetilmesi işlemleri bu kısımda gerçekleştirilir. Bildirim kısmında bildirimi yapılan değişkenlere bu bölümde **WHILE** döngüsü ile elde edilen kayıt verileri aktarılır. Ekrana yazdırma ya da benzeri işlemler de bu kısımda gerçekleştirilir.

Aşağıdaki sorguda, **ProductCursor** isimli Cursor sorunsuz çalışıyor ise **WHILE** döngüsü içerisine girilir ve elde edilen veriler PRINT ile ekranda listelenir.

İlk kaydın yakalanması için bir **FETCH** işlemi gerçekleştirilir. İlk **FETCH** ile bir değer bulunur ise, içerisindeki veriler **FETCH**'deki değişkenlere aktarılır. Bu işlemden sonra **WHILE** döngüsü gerçekleşir.

```
FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + '  -  ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
```

WHILE döngüsüne girmeden önce Cursor'de veri olduğunu nasıl anlıyoruz?

@@FETCH_STATUS global değişken değeri, her satır getirildiğinde güncellenir. Eğer değer **0** (*sıfır*) ise **FETCH** işlemi başarılıdır ve **WHILE** döngüsüne girilir. Daha sonra **FETCH** işlemi döngü halinde gerçekleşir.

@@FETCH_STATUS isimli global değişken **FETCH** işleminin sonucunu bize 3 değer ile bildirir.

- **0 : FETCH** işlemi başarılıdır. Kayıt vardır.

- **1** : Kayıt bulunamadı.
- **2** : **FETCH** başarısız. Bu hata, son kaydın ötesinde ya da ilk kaydın da öncesinde bulunulduğunu belirtir.

Kapanış

Cursor açılışı gerçekleştikten sonra **CLOSE** ile kapatılmadığı sürece sürekli açık kalacaktır. Açık kaldığı takdirde Cursor'ün kullanıldığı yerlerde birçok hata ile karşılaşılabileceği gibi, sürekli açık kalan Cursor sistem kaynağını gereksiz şekilde tüketecektir. Cursor yönetimi açısından, açık olması gereken özel durumlar haricinde mutlaka kapatılması gerekir. Kapanış işlemiyle birlikte Cursor ile ilgili kilitler serbest bırakılır.

```
CLOSE ProductCursor
```

Hafızada Ayrılan Belleği Boşaltmak

CLOSE komutu, Cursor'ün kapanmasını sağlar. Ancak Cursor'ün kapanması demek kullandığı hafızanın boşaltılması anlamına gelmez. Hafıza boşaltma işleminin gerçekleşmesi için **CLOSE** işleminden sonra **DEALLOCATE** komutu kullanılmalıdır.

```
DEALLOCATE ProductCursor
```

Cursor Tipleri ve Özellikleri

Cursor'ler işlev ve yeteneklerine göre farklı tip ve özelliklerle ayrılırlar. Cursor'lerin özellik ve tiplerini inceleyelim.

Scope

Scope, Cursor'lerin görünebilirlik ayarını belirtir. **LOCAL** ve **GLOBAL** olarak ikiye ayrılır. En kısa tanım ile şu şekilde özetlenebilir. Bir sproc içerisinde oluşturulan Cursor **GLOBAL** scope'u içerisinde ise, bir başka sproc içerisinden bu Cursor'e erişilebilir. Ancak bu Cursor **LOCAL** scope içerisinde ise bir başka sproc içerisinden erişilemez. **GLOBAL** olarak tanımlanan bir Cursor ismi ile başka bir sproc içerisinde dahi olsa yeni bir Cursor tanımlanamaz. Etki alanı çerçevesinde benzersiz isime sahip olmalıdır. **GLOBAL** scope içerisindeki Cursor'de başka bir sproc içerisinden de erişilebilir olduğu için, farklı sproc içerisinde de olsa, aynı isimde tanımlama işlemi hataya yol açar.

SQL Server Cursor'leri varsayılan olarak **GLOBAL** özelliğindedir. SQL Server'da bir şeyin **LOCAL** ya da **GLOBAL** olması, tüm bağlantılar ya da sadece geçerli bağlantılar tarafından görülmesini ifade ederi. Yukarıda, bir sproc içerisindeki Cursor'e bir diğerinden ulaşma durumu, bu **görülme** olayını ifade eder.

Kaydırılabilirlik

Kaydırılabilirlik (*scrollability*) özelliği, SQL Server'da Cursor'lerin hareket yönünü ifade eder. Varsayılan olarak sadece ileriye doğru hareket eden Cursor'ler kaydırılabilirlik seçenekleri ile ileri ve geri hareket kabiliyetlerinin kullanılmasını sağlar.

FORWARD_ONLY Özelliği

Cursor'ün varsayılan yöntemidir. Sadece ileriye doğru hareket eder. Tek yöne sahip ve sadece ileriye doğru çalıştığı için kullanılabilecek gezinme özelliği **FETCH NEXT**'dir.

Daha önce örnek verdiğimiz DOT.NET içerisindeki **DataReader** nesnesi gibi çalışır. Hatırlarsanız, **DataReader**'da sadece ileriye doğru çalışır ve bu nedenle çok hızlı sonuç üretirdi. Ancak **DataReader**'da gösterilen bir kaydı tekrar elde etmek için geriye doğru gidemezdik. Aynı kaydı tekrar elde etmek için sorgunun tekrar çalıştırılması gerekiyordu. **FETCH NEXT** de aynı şekilde çalışır. Cursor bir sonraki kayda geçtikten sonra önceki kayda tekrar ulaşmak için Cursor'ün kapatılıp yeniden açılması gerekir.

SCROLLABLE Özelliği

Cursor'ü ileri geri hareket ettirmek için kullanılır. Kaydırma işleminin temel özelliği **FETCH** anahtar sözcüğüdür.

- **FETCH** ile kullanılan yan komutlar şu şekildedir:
- **NEXT :** Bir sonraki kayda geçer.
- **PRIOR :** Bir önceki kayda geçer.
- **FIRST :** İlk kayda geçer.
- **LAST :** Son kayda geçer.

Normal bir Cursor tanımlamasına ek olarak **SCROLL** ifadesi içerir.

Örnek:

```
CREATE PROC proc_name
AS
DECLARE @var1 INT, @var1 INT
DECLARE cursor_name CURSOR
LOCAL
SCROLL
 FOR
  SELECT ...
```

Duyarlilik Kavrami

Cursor duyarlılığı, Cursor açıldıktan sonra veritabanında gerçekleşen değişikliklerin dikkate alınıp alınmadığı konusu ile ilgilidir. Bazı durumlarda, kullanılacak Cursor'ün veritabanı değişikliklerini dikkate almamasını ve veriler değişse de çalışmasına statik veriler ile devam etmesi istenebilir. Bu durumda statik Cursor kullanılır. Statik Cursor, oluşturulduktan sonra veritabanındaki değişikliklere karşı duyarsız olacaktır. Ancak statik Cursor'ün aksine, dinamik Cursor veritabanında gerçekleşen ekleme, güncelleme, silme gibi değişiklik işlemlerine karşı duyarlı olacaktır.

Cursor'lerle Satirlari Dolaşmak: FETCH

FETCH komutu, Cursor için gerçek anlamda gezinme işini yapan komuttur. Bir Cursor'ün üzerinde, ileri ya da geri herhangi bir hareket ve konumlandırma işlemi yapılacak ise, bu işi yapacak olan komuttur.

FETCH işlemi ile kullanılan komutlarla ilgili birçok örnek yaptık. Öncelik sırasında sonlarda olan bazı komutlar için ise bu bölümde örnekler yapacağız.

Çeşitli örnekler yapmak için bir Cursor tanımlayalım.

```
DECLARE ProductCursor SCROLL CURSOR FOR
SELECT ProductID, Name FROM Production.Product WHERE ProductID < 5
```

Cursor'ü açalım.

```
OPEN productCursor
```

Cursor artık çalışıyor. Artık sırasıyla, tüm işlemleri deneyebiliriz.

FETCH NEXT

Cursor ile ilgili hazırladığımız bir çok örnekte **NEXT** komutunu kullandık. **FETCH** işleminde en çok ve ileri doğru hareket için kullanılan sorgu ifadesidir.

NEXT, sonuç kümesinde bir satır ileri girmeyi sağlar. Cursor bildirimi **FORWARD_ONLY** yapıldı ise **NEXT** ifadesi kullanılır.

```
-- Bir kayır ileri hareket ettirir.
FETCH NEXT FROM ProductCursor
```

FETCH PRIOR

NEXT ifadesinin tersi olarak çalışır. Cursor'ün bulunduğu kayıttan geriye doğru bir kayıt hareket etmeyi sağlar. Yani, "önceki" anlamına gelir.

```
-- Önceki kayda gider.
FETCH PRIOR FROM ProductCursor
```

FETCH FIRST

FIRST ifadesi ilk kayda geri dönmek için kullanılır. Cursor hangi satırda olursa olsun, **FIRST** ilk satıra dönmesini sağlar.

```
-- İlk kayda gider
FETCH FIRST FROM ProductCursor
```

FETCH LAST

LAST ifadesinin tersi olarak çalışır. Cursor'ün son kayıt üzerine gitmesini sağlar. Cursor hangi satırda olursa olsun, son kayıt üzerinde konumlanmak için **LAST** kullanılır.

```
-- Son kayda gider
FETCH LAST FROM ProductCursor
```

FETCH RELATIVE

Cursor'un bulunduğu satırdan belirli sayıda satır kadar ileri ya da geriye doğru hareket etmek için kullanılır.

```
-- Bulunulan yerden 3 kayıt ileri konumlanır.
FETCH RELATIVE 3 FROM ProductCursor

-- Bulunulan kayıttan 2 kayıt geriye konumlanır.
FETCH RELATIVE -2 FROM ProductCursor
```

FETCH ABSOLUTE

Cursor başlangıcından itibaren ne kadar satır istendiğini belirten bir **`Integer`** değer alır. Belirtilen değer pozitif ise Cursor'un başından, negatif ise sonundan itibaren verilen değer kadar satır anlamına gelir.

```
-- Baştan 2. kayda konumlanır
FETCH ABSOLUTE 2 FROM ProductCursor
```

Tüm işlemlerden sonra Cursor'ü kapatalım ve hafızadaki alanı boşaltalım.

```
CLOSE ProductCursor
DEALLOCATE ProductCursor
```

TYPE_WARNING

Cursor üzerinde yapılacak kapalı dönüştürme işlemi gerçekleşirken uyarı mesajı verir. Örneğin; Keyset-Driven Cursor tanımlandığında, eğer üzerinde unique indeks yoksa, SQL Server otomatik olarak Keyset-Driven'ı Statik Cursor'e dönüştürür. Bu tür dönüşümler kapalı dönüşüm olarak bilinir.

Bir örnek üzerinde çalışalım.

Aşağıdaki sorguyu çalıştırmadan, daha önce açık bulunan ProductCursor isimli Cursor halen açık ise kapatın ve hafızadan boşaltın.

```
DECLARE ProductCursor CURSOR
GLOBAL
SCROLL
KEYSET
```

```
TYPE_WARNING   -- Dönüştürme mesajını verecek olan özellik
FOR
SELECT ProductID, Name FROM Production.Product

DECLARE @ProductID INT
DECLARE @Name    VARCHAR(5)

OPEN ProductCursor

FETCH NEXT FROM ProductCursor INTO @ProductID, @Name

WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' ' +  @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END

CLOSE ProductCursor
DEALLOCATE ProductCursor
```

```
1 Adjus
2 Beari
3 BB Ba
4 Heads
316 Blade
317 LL Cr
318 ML Cr
319 HL Cr
320 Chain
321 Chain
322 Chain
```

Diğer Cursor örneklerimizden farklı olan tek kısım **TYPE_WARNING** tanımlaması oldu. Sorgu sonucunda bir mesaj alındı. Bu bir hata mesajı değil, sadece bilgilendirme amaçlı uyarı mesajıdır.

Cursor Tipleri

Cursor'ler çeşitli özelliklerine göre 4 farklı tipe ayrılır.

- Statik
- Keyset ile Çalıştırılan (*Keyset-Driven*)
- Dinamik
- Sadece İleri (*Forward-Only*)

Dört tip olarak ayrılan Cursor'lerin aralarındaki farklılığın sebepleri kaydırılabilirlik ve veritabanındaki değişikliklere karşı duyarlılıklarıdır.

Statik Cursor'ler

Cursor oluşturulduktan sonra hiç bir şekilde değiştirilmeyen, yani statik olarak çalışan Cursor tipidir. Statik Cursor oluşturulduğunda, **tempdb** veritabanında, geçici tablonun tutabileceği alan miktarınca kayıt tutulur.

Statik Cursor, **tempdb** üzerinde tutulur ve bir geçici tablo görevi görür. Statik Cursor içerisindeki kayıtlarda güncelleme yapılamaz. Bazı nesne modelleri güncellemeye olanak tanısa da sonuç olarak değiştirilen veri veritabanına yazılamaz.

Statik Cursor özelliğini örnek üzerinde inceleyelim.

Cursor işleminde kullanacağımız ve üzerinde değişiklik yapacağımız tabloyu oluşturalım.

```
SELECT ProductID, Name INTO ProductCursorTable FROM Production.Product;
```

Cursor bildirimini yapalım.

```
DECLARE ProductCursor CURSOR
GLOBAL   -- Cursor batch dışında da kullanılabilir.
SCROLL   -- Geriye doğru kaydırma yapılabilir.
STATIC
FOR
SELECT ProductID, Name FROM ProductCursorTable
```

Cursor için kullanılacak değişkenlerin bildirimini yapalım.

```
DECLARE @ProductID INT
DECLARE @Name   VARCHAR(30)
```

Cursor'ü açalım.

```
OPEN ProductCursor
```

Cursor ile ilk kaydı seçelim.

```
FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
```

Cursor'deki tüm kayıtları seçmek için bir döngü oluşturalım.

```
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' - ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END;
```

Cursor, sorgu sonucunda bize ilgili tablodaki kayıtları listeledi.

```
1 - Adjustable Race
879 - All-Purpose Bike Stand
712 - AWC Logo Cap
3 - BB Ball Bearing
2 - Bearing Ball
877 - Bike Wash - Dissolver
316 - Blade
843 - Cable Lock
952 - Chain
324 - Chain Stays
322 - Chainring
320 - Chainring Bolts
```

Şimdi, oluşturduğumuz tablo ve Cursor'ü kullanarak bir güncelleme işlemi yapalım ve statik Cursor'lerin nasıl çalıştığını örnek ile inceleyelim.

`ProductCursorTable` tablomuzdaki içeriği listeleyelim.

```
SELECT ProductID, Name FROM ProductCursorTable;
```

	ProductID	Name
1	1	Adjustable Race
2	879	All-Purpose Bike Stand
3	712	AWC Logo Cap
4	3	BB Ball Bearing
5	2	Bearing Ball
6	877	Bike Wash - Dissolver
7	316	Blade

Cursor'ün kaynak olarak kullandığı tabloda bir güncelleme yapalım.

```
UPDATE ProductCursorTable
SET Name = 'Update ile değiştirildi'
WHERE ProductID = 1;
```

Tablo üzerinde güncelleme işlemi başarılı.

Şimdi, oluşturduğumuz Cursor'ü tekrar çalıştırarak verinin Cursor'de değişip değişmediğini görelim.

```
DECLARE @ProductID INT
DECLARE @Name   VARCHAR(30)

FETCH FIRST FROM ProductCursor INTO @ProductID, @Name

WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' - ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
```

```
1 - Adjustable Race
879 - All-Purpose Bike Stand
712 - AWC Logo Cap
3 - BB Ball Bearing
2 - Bearing Ball
877 - Bike Wash - Dissolver
316 - Blade
843 - Cable Lock
952 - Chain
324 - Chain Stays
322 - Chainring
```

Sorgu sonucunda, tabloda yapılan değişikliğin Cursor tarafından fark edilmediğini görüyoruz. Durumu onaylamak için tabloyu tekrar listeleyelim. Tabloda değişmiş olan veri, Cursor'de değişmemiştir.

```
SELECT ProductID, Name FROM ProductCursorTable;
```

	ProductID	Name
1	1	Update ile degistirildi
2	879	All-Purpose Bike Stand
3	712	AWC Logo Cap
4	3	BB Ball Bearing
5	2	Bearing Ball
6	877	Bike Wash - Dissolver
7	316	Blade

Son işlem olarak Cursor kapatılmalı ve hafızada kapladığı yer boşaltılmalıdır.

```
CLOSE ProductCursor
DEALLOCATE ProductCursor
```

Örnekte kullanılan kodların tamamı;

```
SELECT ProductID, Name INTO ProductCursorTable FROM Production.Product
DECLARE ProductCursor CURSOR
GLOBAL
SCROLL
STATIC
FOR
SELECT ProductID, Name FROM ProductCursorTable
DECLARE @ProductID INT
DECLARE @Name   VARCHAR(30)
OPEN ProductCursor

FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' - ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
UPDATE ProductCursorTable
SET Name = 'Update ile değiştirildi'
WHERE ProductID = 1
FETCH FIRST FROM ProductCursor INTO @ProductID, @Name
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' - ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
CLOSE ProductCursor
DEALLOCATE ProductCursor
```

Yukarıdaki sorguda neler yapıldı?

- Bir tablo oluşturularak içerisine veri eklendi.
- Oluşturulan Static Cursor ile bu tablodaki veriler elde edildi. Ekranda listeleme yapıldı.
- Tablo üzerinde güncelleme işlemi yapılarak gerçek tablodaki veri gerçek anlamda değiştirildi.
- Statik Cursor'ler oluşturulduktan sonra veritabanı ile bağlantısını kestiği için, yapılan güncellemenin farkında olamadı. Bu nedenle Cursor'deki veriler değişmedi.

ANAHTAR TAKIMI İLE ÇALIŞTIRILAN CURSOR'LER

Anahtar Takımı Cursor'ler (*Keyset-Driven Cursor*), veritabanındaki tüm satırları **unique** olarak tanımlayan veri kümesi sağlayan Cursor'lerdir.

Anahtar Takımı ile Çalıştırılan Cursor'lerin Özellikleri:

- Cursor'ün kullandığı tablo üzerinde unique indeks olması gerekir.
- tempdb veritabanında veri kümes değil, sadece anahtar takımı saklanır.
- Satırlarda meydana gelen tüm değişikliklere duyarlıdırlar. Ancak Cursor oluşturulduktan sonra eklenen yeni satırlara karşı duyarlı değillerdir.

Anahtar takımı ile çalışan Cursor'ler kullanıldığında, tüm anahtarlar tempdb içerisinde saklanır. Bu anahtarlar gerçek veriye ulaşmak için, veri bulma yöntemi olarak kullanılır. Bir işaretleyici olarak da düşünülebilir. **FETCH** işlemi sırasında ulaşılacak gerçek veriye bu anahtarlar ile ulaşılır.

Şimdi bir Keyset-Driven Cursor oluşturalım.

Bu örnekte **ProductCursorTable** tablosunu ve **ProductCursor** isimli Cursor'ü tekrar kullanacağız. Bu nedenle daha önce kullandığımız **ProductCursorTable** tablosunu **DROP TABLE** ifadesi ile silin ve cursor'ü de **DEALLOCATE** ile hafızadan boşaltın.

Cursor kullanacağımız tabloyu oluşturalım.

```
SELECT ProductID, Name
INTO ProductCursorTable
FROM Production.Product WHERE ProductID > 0 AND ProductID < 300
```

Cursor tablosu üzerinde bir **Primary Key** biçiminde **Unique** indeks oluşturalım.

```
ALTER TABLE ProductCursorTable
ADD CONSTRAINT PKCursor
PRIMARY KEY(ProductID)
```

Identity sütunlarda boş olan sütunları değiştirebilmek için;

```
SET IDENTITY_INSERT ProductCursorTable ON;
```

Cursor bildirimini yapalım.

```
DECLARE ProductCursor CURSOR
GLOBAL
SCROLL
KEYSET
FOR
SELECT ProductID, Name FROM Production.Product
WHERE ProductID > 0 AND ProductID < 300
```

Cursor'de kullanılacak değişkenleri tanımlayalım.

```
DECLARE @ProductID INT
DECLARE @Name   VARCHAR(30)
```

Cursor'ü açalım.

```
OPEN ProductCursor
```

İlk kaydı bulup getirelim.

```
FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
```

Tüm kayıtları bulup getirmek için bir döngü oluşturalım.

```
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
```

```
1 Adjustable Race
2 Bearing Ball
3 BB Ball Bearing
4 Headset Ball Bearings
```

Cursor'ün kullandığı tabloda güncelleştirme yapalım.

```
UPDATE ProductCursorTable
SET Name = 'Değiştirildi'
WHERE ProductID = 1
```

Cursor'ün kullandığı tabloda silme işlemi yapalım.

```
DELETE ProductCursorTable
WHERE ProductID = 2
```

Cursor'ün kullandığı tabloda kayıt ekleme işlemi yapalım.

```
INSERT INTO ProductCursorTable(ProductID, Name)
VALUES(299,'Test Veri')
```

İlk kaydı bulalım.

```
DECLARE @ProductID INT;
DECLARE @Name   VARCHAR(30);
FETCH FIRST FROM ProductCursor INTO @ProductID, @Name
```

Bir **WHILE** döngüsü oluşturarak verilerin, değiştirme (**Insert**, **Update**, **Delete**) işlemlerinden sonraki durumlarını inceleyelim.

```
DECLARE @ProductID INT;
DECLARE @Name   VARCHAR(30);
WHILE @@FETCH_STATUS = 0
BEGIN
 PRINT CAST(@ProductID AS VARCHAR) + ' ' + @Name
 FETCH NEXT FROM ProductCursor INTO @ProductID, @Name
END
```

```
2 Bearing Ball
3 BB Ball Bearing
4 Headset Ball Bearings
```

Cursor'ü kapatalım ve hafızada kapladığı yeri boşaltalım.

```
CLOSE ProductCursor
DEALLOCATE ProductCursor
```

Cursor'lerin aralarındaki teknik farklılıkları kavramanın en iyi yolu farklı işlemler için farklı Cursor'ler programlamaktır.

Dinamik Cursor'ler

Temeldeki veride yapılan değişiklikleri proaktif olarak göstermezler. Ancak, tüm değişikliklere karşı duyarlı olmaları nedeniyle dinamik olarak adlandırılırlar.

Cursor oluşturulmasından sonra eklenen kayıtlar, güncellenen satırların Cursor'da da güncellenmesi, silinen kayıtların Cursor'den silinmesi işlemlerine karşı duyarlıdırlar. Dinamik Cursor'lerin performansı düşürdüğü durumlar olabilir. Eğer temel tablodaki kayıtlar çok geniş değilse dinamik Cursor'ler performanslı çalışmaya devam edecektir. Ancak, temel tablonun veri ve sütun genişliği fazla ise performanslı çalışmayacak ve bu durum bir sorun oluşturacaktır. Bunun nedenini anlamak için dinamik Cursor'lerin çalışmasını incelemek gerekir.

Diğer Cursor tiplerinde Cursor oluşturulduktan sonra veri hafızaya alınır ve üzerinde okuma işlemi gerçekleştirilirdi. Bu nedenle, bir veri kümesi Cursor'e alındıktan sonra tablodaki yeni kayıt ve güncellenen kayıtlardan haberi olmazdı. Ancak dinamik Cursor'ler tablo temelinde veri duyarlılığına sahiptir. Bunun sebebi, dinamik Cursor'lerin her **`FETCH`** komutu çalıştırıldığında Cursor'ü yeniden oluşturması ve içerdiği **`SELECT`** ifadesini **`WHERE`** ile birlikte tekrar çalıştırmasıdır.

Bazen sorgu işlemlerinde hız her şey demek değildir. Dinamik cursor'ler genellikle cache üzerinde çalışır. Yani, RAM (*bellek*) kullanımı yüksek bir cursor tipidir. Hatırlarsanız Keyset Cursor'ler ise **`tempdb`** üzerinden, yani disk üzerinden çalışıyorlardı. Geniş bellek kapasitesinin olduğu durumlarda RAM üzerinden çalışmak, disk'e göre daha hızlıdır. Ancak, SQL Server'a yoğun sorgu oluşturulduğunda, geniş tablo yapısı ve veriye sahip olduğu durumlarda, SQL Server'ın kullandığı bellek alanı artacağı için sistemdeki kullanılmayan bellek alanı düşecektir. Bu da RAM üzerinde çalışan dinamik Cursor'ler için büyük risktir. Gene de, küçük veri kümeleri ile çalışılıyorsa, hem veri duyarlılığı olması hem de performanslı çalışmaları nedeniyle dinamik Cursor'ler tercih edilebilirler.

Aşağıdaki şekilde tanımlanırlar.

```
DECLARE cursor_ad CURSOR
DYNAMIC  -- Cursor'e dinamik özelliği kazandırır.
SCROLL  -- Cursor'e ileri ve geri hareket özelliği kazandırır.
```

```
GLOBAL -- Cursor'e batch dışında kullanılabilme özelliği kazandırır.
FOR
select_ifadesi
```

FOR <SELECT>

Cursor bildirimlerinde gördüğünüz gibi, en önemli Cursor komutu parçasıdır. Bir Cursor için gerekli **SELECT** cümleciğinin tanımlanacağı kısımdır.

FOR UPDATE

FOR UPDATE özelliği, Cursor içerisinde belirtilen sütunların güncellenmesi için kullanılır. Sadece sütun listesinde bulunan sütunlar güncellenebilir. Sütun listesinde bulunmayanlar salt okunur kalacaktır.

FOR UPDATE için bir örnek yapalım.

Güncelleme için kullanacağımız tabloyu oluşturalım.

```
CREATE TABLE dbo.Employees(
    EmployeeID INT NOT NULL,
    Random_No  VARCHAR(50) NULL
) ON [PRIMARY]
```

Employees tablosunun içeriğini listeleyelim. Boş, yani herhangi bir kayıt eklemedik.

	EmployeeID	Random_No

WHILE döngüsü için gerekli tanımlamaları yapalım ve döngüyü çalıştıralım.

```
SET NOCOUNT ON
DECLARE @Rec_ID   AS INT
SET @Rec_ID = 1

WHILE (@Rec_ID <= 100)
BEGIN
    INSERT INTO Employees
    SELECT @Rec_ID, NULL

    IF(@Rec_ID <= 100)
    BEGIN
```

```
    SET @Rec_ID = @Rec_ID + 1
    CONTINUE
    END
    ELSE
    BEGIN
    BREAK
    END
END
SET NOCOUNT OFF
```

Yukarıdaki **WHILE** döngüsü ile birlikte, artık 100 kayıtlık bir tablomuz oluştu. 100 kayıt oluşturmak için **WHILE** döngüsü kullanıldı.

Tablodaki veriyi listelemek için;

```
SELECT EmployeeID, Random_No FROM Employees;
```

	EmployeeID	Random_No
1	1	NULL
2	2	NULL
3	3	NULL
4	4	NULL
5	5	NULL
6	6	NULL
7	7	NULL

Tabloya bir Constraint ekliyoruz.

```
ALTER TABLE dbo.Employees
ADD CONSTRAINT PK_Employees PRIMARY KEY CLUSTERED
(
    EmployeeID ASC
)ON [PRIMARY]
```

Güncelleme işlemini gerçekleştirelim. Bu kısımda, daha önce oluşturduğumuz tablonun **NULL** bırakılan **Random_No** sütununa rastgele oluşturulan değerler atayarak güncelleme gerçekleştireceğiz.

```
SET NOCOUNT ON

DECLARE
    @Employee_ID   INT,
    @Random_No  VARCHAR(50),
    @TEMP      VARCHAR(50)
DECLARE EmployeeCursor CURSOR FOR
SELECT EmployeeID, Random_No FROM Employees FOR UPDATE OF Random_No
OPEN EmployeeCursor
FETCH NEXT FROM EmployeeCursor
INTO @Employee_ID, @Random_No
WHILE (@@FETCH_STATUS = 0)
BEGIN
    SELECT @TEMP =  FLOOR(RAND()*10000000000000)
    UPDATE Employees SET Random_No = @TEMP WHERE CURRENT OF
EmployeeCursor
    FETCH NEXT FROM EmployeeCursor
    INTO @Employee_ID, @Random_No
END

CLOSE EmployeeCursor
DEALLOCATE EmployeeCursor

SET NOCOUNT OFF
```

FOR UPDATE ile eklenen **Random** verilerden sonra tablo kayıtlarını tekrar listeleyelim.

```
SELECT EmployeeID, Random_No FROM Employees;
```

	EmployeeID	Random_No
1	1	1.61579e+012
2	2	2.2216e+012
3	3	1.8678e+012
4	4	7.76989e+012
5	5	5.66375e+012
6	6	5.87996e+012
7	7	3.56572e+012

FOR UPDATE ile güncelleme işlemimiz başarıyla gerçekleştirilmiş ve tüm satırlara **Random** olarak veriler eklenmiştir.