مشکل Concurrency در SQL Server و مدیریت آن در Entity Framework

یکی از مشـکلاتی که در محیط هایی که چندین کاربر به صورت همزمان از یک بانک اطلاعاتی [illegible] می کنند، مشکل Concurrency است. این مشـکل زمانی پیش می آید که دو کاربر به صورت همزمان قصد ایجاد تغییر در یک رکورد را داشته باشند، فرض کنید کاربر ۱ رکورد شماره ۴ را برای تغییر از بانـک اطلاعاتی خوانـده است، در همین زمان که کاربر ۱ در حال تغییر رکورد ها بوده و تغییرات هنوز در بانک اطلاعاتی ثبت نشـده انـد، کاربر ۲ همان رکورد را می خوانـد، کاربر ۱ تغییرات را ذخیره می کنـد و سپس کاربر ۲ تغییرات را ذخیره می کنـد. در این حالت اطلاعات کاربری که آخرین به روزرسانی را در بانک ثبت می کند به عنوان اطلاعات اصلی در بانک ثبت می شود. در حالت عادی این موضوع مشکلی ایجاد نمی کند، اما اگر نوشتن اطلاعات توسط کاربرها اولویت داشـته باشد باید برای رفع این مشـکل کاری کرد. برای مدیریت Concurrency ها در Entity Framework در مدلی که قصد مدیریت Concurrency را برای آن داریم بایـد یـک خصوصـیت تعریـف کرده و آن خصوصـیت را به عنـوان خصوصـیتی که جهت جلوگیری از Concurrency استفاده می شود مشخص کنیم. مدل زیر را در نظر بگیرید:

```
public class Customer
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

public class SampleContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }
}
```

در حالت عادی اگر کد زیر را اجرا کنیم مشکلی ایجاد نخواهد شد:

```
var context1 = new SampleContext();
var context2 = new SampleContext();

var customer1 = context1.Customers.First();
var customer2 = context2.Customers.First();

customer1.FirstName = "Mohammad";
customer2.FirstName = "Reza";

context1.SaveChanges();
context2.SaveChanges();
```

در کد بالا در دو Context جداگانه رکوردی از جدول Customers خوانده شده و خصوصیت FirstName آن تغییر داده می شود، در

کد بالا با فراخوانی متد SaveChanges هیچ مشکلی ایجاد نشده و نام Reza به عنوان نام نهایی در جدول ثبت می شود. در این حالت Concurrency رخ داده است. برای مدیریت این حالت، کلاس Customers را به صورت زیر تغییر می دهیم:

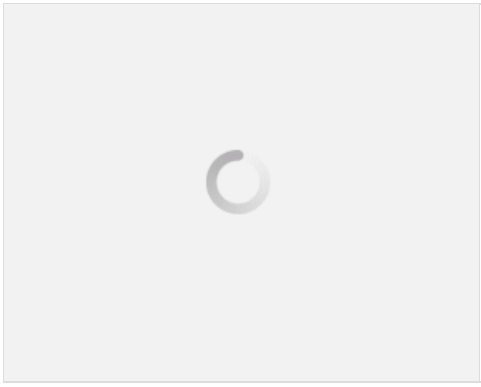
```
public class Customer
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    [Timestamp]
    public byte[] RowVersion { get; set; }
}
```

خصوصیت RowVersion که در کـد بالا مشاهده می کنید، بعد از ایجاد بانک اطلاعاتی سـتونی از نوع timestamp در جدول ایجاد خواهـد کرد و از این سـتون برای مدیریت Concurrency اسـتفاده می شود، با هر بار خوانـدن رکورد از جـدول مقدار RowVersion تغییر می کند و زمان به روز رسانی از این ستون برای مقایسه بین نسخه Entity و نسخه بانک اطلاعاتی استفاده شده و در صورت وجود تفاوت میان مقادیر بانک اطلاعاتی و RowVersion خطایی از نوع DbUpdateConcurrencyException ایجاد خواهد شد. بعد از اضافه کردن خصوصیت RowVersion کد به روزرسانی اطلاعات رو به صورت زیر تغییر می دهیم:

```
var context1 = new SampleContext();
var context2 = new SampleContext();

var customer1 = context1.Customers.First();
var customer2 = context2.Customers.First();

customer1.FirstName = "Mohammad";
customer2.FirstName = "Reza";



try
{
    context1.SaveChanges();
}
catch (DbUpdateConcurrencyException ex)
{
    throw;
}

try
{
    context2.SaveChanges();
}
catch (DbUpdateConcurrencyException ex)
{
```

```
    throw;
}
```

با اجرای کد بالا پیغام خطایی به صورت زیر دریافت می کنیم:



همچنین اگر دستورات SQL اجرا شده توسط Entity Framework را بررسـی کنیم، خواهیم دید که در زمان اجرای دستورات update ستون RowVersion نیز به در قسمت where استفاده شده است:

```
UPDATE [dbo].[Customers]
SET [FirstName] = @0
WHERE (([Id] = @1) AND ([RowVersion] = @2))
```

زمان وقوع خطا شما می توان در بخش catch عملیات مورد نظر خود را انجام دهید، برای مثال، رکورد را مجدد بخوانید و Update را انجام دهید یا مقدار جاری را نگه داری کنید که این موضوع بستگی به نحوه پیاده سازی سیستم خواهد داشت. علاوه بر ستون RowVersion می توان ستون های دیگری را نیز به عنوان ستونی که برای بررسی Concurrency استفاده می شود انتخاب کرد، برای مثال، در مدل زیر ستون FirstName به عنوان ستون Concurrency Check انتخاب شده است:

```
public class Customer
{
    public int Id { get; set; }
    [ConcurrencyCheck]
    public string FirstName { get; set; }
    public string LastName { get; set; }
    [Timestamp]
    public byte[] RowVersion { get; set; }
}
```

دستور SQL ایجاد شده در حالت بالا به صورت زیر خواهد بود:

```
UPDATE [dbo].[Customers]
SET [FirstName] = @0
WHERE ((([Id] = @1) AND ([FirstName] = @2)) AND ([RowVersion] = @3))
```

به این نکته دقت کنیـد که در زمان تعریف Entity ها، برای هر Entity تنها یک ستون Timestamp می توان تعریف کرد، اما چندین ستون رو می توان با ConcurrencyCheck مشخص کرد.

نویسنده: حسن احمدی

منبع: ITpro

نظر شما

برای ارسال نظر باید وارد شوید.

نظر

هیچ نظری ارسال نشده است! اولین نظر برای این مطلب را شما ارسال کنید...

افرادی که این مطلب را خواندند مطالب زیر را هم خوانده اند