Параллелизм в Entity Framework

Последнее обновление: 31.10.2015

При работе с Entity Framework, когда у нас одновременно множесво пользователей имеют доступ к одинаковому набору данных и могут эти данные изменять, мы можем столкнуться с проблемой параллелизма. Например, два пользователя независимо друг от друга начнут редактировать один и тот же объект. И после сохранения объекта первым пользователем второй пользователь уже будет работать с неактуальными данными.

Рассмотрим на примере. Например, в приложении ASP.NET MVC есть стандартное действие для редактирования:

```
public ActionResult Edit(int id)
{
    Person p = db.People.Find(id);
    return View(p);
}
[HttpPost]
public ActionResult Edit(Person p)
{
    db.Entry(p).State = EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Хотя данный код прекрасно будет работать, но если оба пользователя одновременно откроют один и тот же объект на редактирование, то после сохранения изменнного объекта первым пользователем, второй пользователь уже буде работать с устаревшими данными.

Что предлагает Entity Framework для решения этой проблемы?

Прежде всего стоит сказать, что есть два типа параллелизма: оптимистичный и пессимистичный.

При пессимистичном параллелизме (pessimistic concurrency) на базу данных накладываются ограничения по доступу. Например, строки объявляются только для чтения или обновления. Пессимистичный параллелизм предполагает работу на уровне базы данных и предполагает создание сложной программной логики, которая бы отслеживала и управляла правами доступа. В Entity Framework поддержки для пессимистичного параллелизма нет.

При оптимистичном параллелизме (optimistic concurrency) допускается проблема параллельного доступа к данным со стороны разных пользователей, как в выше приведенном случае с обновлением. И для решения оптимистичного параллелизма в Entity Framework есть свои методы.

Одни из методов предполагает, что в модели мы объявляем специальное свойство с атрибутом `[Timestamp]`, которое будет отслеживать модификацию строки в таблице. Например:

```
using System.ComponentModel.DataAnnotations;
//.............................
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
```

```
    public int Age { get; set; }

    [Timestamp]
    public byte[] RowVersion { get; set; }
}
```

Атрибут Timestamp указывает, что значение свойства RowVersion будет включаться в создаваемое Entity Frameworkом SQL-выражение Where при отправке в базу данных команд на обновление и удаление. В качестве типа для свойства используется массив байтов.

Также на представлении для редактирования надо добавить скрытое поле для хранения значения новго свойства:

```
@Html.HiddenFor(model => model.RowVersion)
```

Теперь, если два пользователя одновременно начнут редактировать одну и ту же модель, то после сохранения модели первым пользователем, второй пользователь получит исключение **DbUpdateConcurrencyException** (находится в пространстве имен System.Data.Entity.Infrastructure), которое соответственно надо обработать:

```
[HttpPost]
public ActionResult Edit(Person p)
{
    try
    {
        db.Entry(p).State = EntityState.Modified;
        db.SaveChanges();
    }
    catch (DbUpdateConcurrencyException ex)
    {
        ViewBag.Error = "Объект ранее уже был изменен";
        return View(p);
    }
    return RedirectToAction("Index");
}
```

Ну и чтобы дать пользователю знать об ошибке, где-нибудь в представлении выведем `ViewBag.Error`. И теперь при одовременном редактировании одного и того же объекта второй сохраняющий пользователь получит ошибку, а его обновления не будут сохранены в базу данных.

При чем это касается не только обновления, но и удаления.

Назад **Содержание** **Вперед**