Previous Next

Update Data in Disconnected Scenario in Entity Framework Core

EF Core API builds and execute UPDATE statement in the database for the entities whose `EntityState` is Modified. In the connected scenario, the `DbContext` keeps track of all entities so it knows which are modified and hence automatically sets `EntityState` to Modified.

In the disconnected scenario such as in a web application, the `DbContext` is not aware of the entities because entities were modified out of the scope of the current `DbContext` instance. So, first we need to attach the disconnected entities to a `DbContext` instance with Modified `EntityState`.

The following table lists the `DbContext` and `DbSet` methods to update entities:

DbContext Methods	DbSet Methods	Description
DbContext.Update	DbSet.Update	Attach an entity to DbContext with Modified state.
DbContext.UpdateRange	DbSet.UpdateRange	Attach collection of entities to DbContext with Modified state.

The following example demonstrates updating a disconnected entity.

```
// Disconnected Student entity
var stud = new Student(){ StudentId = 1, Name = "Bill" };

stud.Name = "Steve";

using (var context = new SchoolContext())
{
    context.Update<Student>(stud);

    // or the followings are also valid
    // context.Students.Update(stud);
    // context.Attach<Student>(stud).State = EntityState.Modified;
    // context.Entry<Student>(stud).State = EntityState.Modified;

    context.SaveChanges();
}
```

In the above example, consider the `stud` is an existing `Student` entity object because it has a valid Key property value (StudentId = 1). Entity Framework Core introduced the `DbContext.Update()` method which attaches the specified entity to a context and sets its `EntityState` to Modified. Alternatively, you can also use the `DbSet.Update()` method (`context.Students.Update(stud)`) to do the same thing.

The above example executes the following UPDATE statement in the database.

```
exec sp_executesql N'SET NOCOUNT ON;
UPDATE [Students] SET [Name] = @p0
WHERE [StudentId] = @p1;
SELECT @@ROWCOUNT;
',N'@p1 int,@p0 nvarchar(4000)',@p1=1,@p0=N'Steve'
go
```

Update Multiple Entities

Use the `DbContext.UpdateRange` or `DbSet.UpdateRange` method to attach a collection or array of entities to the `DbContext` and set their `EntityState` to Modified in one go.

```
var modifiedStudent1 = new Student()
{
    StudentId = 1,
    Name = "Bill"
};

var modifiedStudent2 = new Student()
{
    StudentId = 3,
    Name = "Steve"
};

var modifiedStudent3 = new Student()
{
    StudentId = 3,
    Name = "James"
};

IList<Student> modifiedStudents = new List<Student>()
{
    modifiedStudent1,
    modifiedStudent2,
    modifiedStudent3
};

using (var context = new SchoolContext())
{
    context.UpdateRange(modifiedStudents);

    // or the followings are also valid
    //context.UpdateRange(modifiedStudent1, modifiedStudent2, modifiedStudent3);
    //context.Students.UpdateRange(modifiedStudents);
                //context.Students.UpdateRange(modifiedStudent1,    modifiedStudent2,
modifiedStudent3);

    context.SaveChanges();
}
```

As you can see, the `UpdateRange` method has two overloads. One overload takes a collection of entities and the second overload takes `object[]` as a parameter. The `DbSet.UpdateRange` method works in the same way as the `DbContext.UpdateRange` method.

EF Core improves the performance by building an UPDATE statement for all the entities in the above example and executes it in a **single database round trip**.

```
exec sp_executesql N'SET NOCOUNT ON;
UPDATE [Students] SET [Name] = @p0
WHERE [StudentId] = @p1;
SELECT @@ROWCOUNT;

UPDATE [Students] SET [Name] = @p2
WHERE [StudentId] = @p3;
SELECT @@ROWCOUNT;

UPDATE [Students] SET [Name] = @p4
WHERE [StudentId] = @p5;
SELECT @@ROWCOUNT;

',N'@p1    int,@p0    nvarchar(4000),@p3    int,@p2    nvarchar(4000),@p5    int,@p4
nvarchar(4000)',
@p1=1,@p0=N'Bill',@p3=2,@p2=N'Steve',@p5=3,@p4=N'James'
go
```

Change in EntityState

The `Update` method sets the `EntityState` based on the value of the key property. If the root or child entity's key property is empty, null or default value of the specified data type then the `Update()` method considers it a new entity and sets its `EntityState` to Added in Entity Framework Core 2.x.

```
public static void Main()
{
    var newStudent = new Student()
    {
        Name = "Bill"
    };

    var modifiedStudent = new Student()
    {
        StudentId = 1,
        Name = "Steve"
    };

    using (var context = new SchoolContext())
    {
        context.Update<Student>(newStudent);
        context.Update<Student>(modifiedStudent);

        DisplayStates(context.ChangeTracker.Entries());
    }
}
private static void DisplayStates(IEnumerable<EntityEntry> entries)
{
    foreach (var entry in entries)
    {
        Console.WriteLine($"Entity: {entry.Entity.GetType().Name},
                  State: {entry.State.ToString()} ");
    }
}
```

Output:

```
Entity: Student, State: Added
Entity: Student, State: Modified
```

In the above example, `newStudent` does not have a Key property value (StudentId). So, the `Update()` method will mark it as Added, whereas modifiedStudent has a value, so it will be marked as Modified.

Exception:

The `Update` and `UpdateRange` methods throw an `InvalidOperationException` if an instance of `DbContext` is already tracking an entity with the same key property value. Consider the following example:

```
var student = new Student()
{
    StudentId = 1,
    Name = "Steve"
};

using (var context = new SchoolContext())
{
    // loads entity in a conext whose StudentId is 1
    context.Students.First<Student>(s => s.StudentId == 1);

    // throws an exception as it already tracking entity with StudentId=1
    context.Update<Student>(student);

    context.SaveChanges();
}
```

In the above example, a `context` object loads the `Student` entity whose `StudentId` is 1 and starts tracking it. So, attaching an entity with the same key value will throw the following exception:

The instance of entity type 'Student' cannot be tracked because another instance with the same key value for {'StudentId'} is already being tracked. When attaching existing entities, ensure that only one entity instance with a given key value is attached. Consider using 'DbContextOptionsBuilder.EnableSensitiveDataLogging' to see the conflicting key values.

Learn to delete records in the disconnected scenario in the next chapter.

Previous

Next

ENTITYFRAMEWORKTUTORIAL

Learn Entity Framework using simple yet practical examples on EntityFrameworkTutorial.net for free. Learn Entity Framework DB-First, Code-First and EF Core step by step. While using this site, you agree to have read and accepted our terms of use and privacy policy.

feedback@entityframeworktutorial.net

TUTORIALS

- EF 6 DB-First
- EF 6 Code-First

E-MAIL LIST

Subscribe to EntityFrameworkTutorial email list and get EF 6 and EF Core Cheat Sheets, latest updates, tips & tricks about Entity Framework to your inbox.

Email address GO

We respect your privacy.
