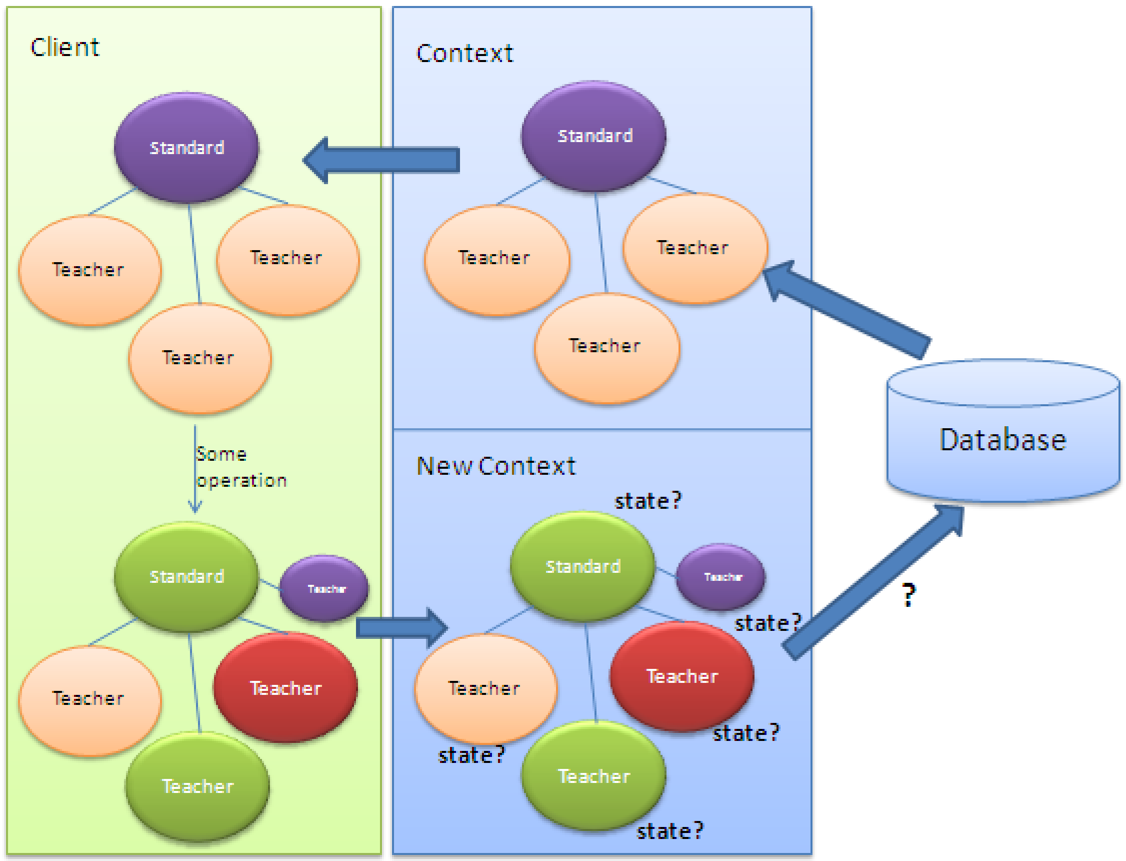
< Previous Next >

Saving a Disconnected Entity Graph in EF 6

In the previous chapter, you learned how to save a disconnected entity. Here, you will learn how to save a disconnected entity graph.

Updating an entity graph in the disconnected scenario is a complex task and needs careful design consideration.

The problem in updating a disconnected entity graph is that the context doesn't know what operation was performed on it at the client side. As per the following figure, the new context doesn't know the state of each entity:



Entity Graph in Disconnected Scenario

We need to identify the states of each entity in the entity graph before calling the `SaveChages()` method. There are different patterns to identify the entity state, which we need to consider in designing the data layer with Entity Framework. Here we are going to use the key property to identify the entity state.

Using Key Property

You can use the key (PrimaryKey) property of each entity to determine its state. If a value of the key property is the default value of CLR data type, then consider it as a new entity. Otherwise consider it an existing entity. For example, if the data type of a key property is `int` and the value of the property is zero, then it is new entity and so, EF needs to execute the INSERT command. If a value is non-zero, then it is an existing entity and EF needs to execute the UPDATE command.

The following example demonstrates saving the `Student` entity graph with the related `Standard` and `Course` entities:

```
var student = new Student() { //Root entity (empty key)
        StudentName = "Bill",
        Standard = new Standard()   //Child entity (with key value)
                    {
                        StandardId = 1,
                        StandardName = "Grade 1"
                    },
        Courses = new List<Course>() {
            new Course(){  CourseName = "Machine Language" }, //Child entity (empty key)
            new Course(){  CourseId = 2 } //Child entity (with key value)
        }
    };

using (var context = new SchoolDBEntities())
{
    //mark standard based on StandardId
    context.Entry(student).State = student.StudentId == 0 ? EntityState.Added : EntityState.Modified;

          context.Entry(student.Standard).State   =   student.Standard.StandardId   ==   0   ?   EntityState.Added   :
EntityState.Modified;

    foreach (var course in student.Courses)
        context.Entry(course).State = course.CourseId == 0 ? EntityState.Added : EntityState.Modified;

    context.SaveChanges();
}
```

In the above example, the `student` entity graph includes the related `Standard` and `Course` entities. The context sets an appropriate state to each entity based on the key property value. If it is zero, then it sets the Added state, otherwise it sets the Modified state. The `SaveChanges()` method will execute the appropriate INSERT or UPDATE command to the database for each entity.

Download EF 6 DB-First Demo Project from Github

Previous

Next

feedback@entityframeworktutorial.net

TUTORIALS

- EF Basics
- EF Core
- 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.

HOME PRIVACY POLICY ADVERTISE WITH US