Previous Next

Working with Disconnected Entity Graph in Entity Framework Core

In the previous chapter, you learned how the `ChangeTracker` automatically changes the `EntityState` of each entity in the connected scenario. Here, you will learn about the behaviours of different methods on the root entity and child entities of the disconnected entity graph in Entity Framework Core.

Entity Framework Core provides the following different methods, which not only attach an entity to a context, but also change the `EntityState` of each entity in a disconnected entity graph:

- Attach()
- Entry()
- Add()
- Update()
- Remove()

Let's see how the above methods change the `EntityState` of each entity in an entity graph in Entity Framework Core 2.x.

Attach()

The `DbContext.Attach()` and `DbSet.Attach()` methods attach the specified disconnected entity graph and start tracking it. They return an instance of `EntityEntry`, which is used to assign the appropriate `EntityState`.

The following example demonstrates the behaviour of the `DbContext.Attach()` method on the `EntityState` of each entity in a graph.

```
public static void Main()
{
    var stud = new Student() { //Root entity (empty key)
        Name = "Bill",
        Address = new StudentAddress()  //Child entity (with key value)
        {
            StudentAddressId = 1,
            City = "Seattle",
            Country = "USA"
        },
        StudentCourses = new List<StudentCourse>() {
                new StudentCourse(){   Course = new Course(){ CourseName = "Machine
Language" } },//Child entity (empty key)
              new StudentCourse(){  Course = new Course(){  CourseId = 2 } } //Child
entity (with key value)
        }
    };

    var context = new SchoolContext();
    context.Attach(stud).State = EntityState.Added;

    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: StudentAddress, State: Unchanged
Entity: StudentCourse, State: Added
Entity: Course, State: Added
Entity: StudentCourse, State: Added
Entity: Course, State: Unchanged
```

In the above example, `stud` is an instance of the `Student` entity graph which includes references of `StudentAddress` and `StudentCourse` entities. `context.Attach(stud).State = EntityState.Added` attaches the `stud` entity graph to a context and sets Added state to it.

The `Attach()` method sets Added `EntityState` to the root entity (in this case `Student`) irrespective of whether it contains the Key value or not. If a child entity contains the key value, then it will be marked as Unchanged, otherwise it will be marked as Added. The output of the above example shows that the `Student` entity has Added `EntityState`, the child entities with non-empty key values have Unchanged `EntityState` and the ones with empty key values have Added state.

The following table lists the behaviour of the `Attach()` method when setting a different `EntityState` to a disconnected entity graph.

Attach()	Root entity with Key value	Root Entity with Empty or CLR default value	Child Entity with Key value	Child Entity with empty or CLR default value
context.Attach(entityGraph).State = EntityState.Added	Added	Added	Unchanged	Added
context.Attach(entityGraph).State = EntityState.Modified	Modified	Exception	Unchanged	Added
context.Attach(entityGraph).State = EntityState.Deleted	Deleted	Exception	Unchanged	Added

Entry()

The `DbContext.Entry()` method behaves differently in Entity Framework Core compared with the previous EF 6.x. Consider the following example:

```
var student = new Student() { //Root entity (empty key)
    Name = "Bill",
    Address = new StudentAddress()  //Child entity (with key value)
    {
        StudentAddressId = 1,
        City = "Seattle",
        Country = "USA"
    },
    StudentCourses = new List<StudentCourse>() {
                    new StudentCourse(){   Course = new Course(){ CourseName="Machine Language" } },//Child entity (empty key)
                new StudentCourse(){   Course = new Course(){   CourseId=2 } } //Child entity (with key value)
        }
};

var context = new SchoolContext();
context.Entry(student).State = EntityState.Modified;

DisplayStates(context.ChangeTracker.Entries());
```

Output:

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

In the above example, `context.Entry(student).State = EntityState.Modified` attaches an entity to a context and applies the specified `EntityState` (in this case, Modified) to the root entity, irrespective of whether it contains a Key property value or not. It ignores all the child entities in a graph and does not attach or set their `EntityState`.

The following table lists different behaviours of the `DbContext.Entry()` method.

Set EntityState using Entry()	Root entity with Key value	Root Entity with Empty or CLR default value	Child Entities with/out Key value

Set EntityState using Entry()	Root entity with Key value	Root Entity with Empty or CLR default value	Child Entities with/out Key value
context.Entry(entityGraph).State = EntityState.Added	Added	Added	Ignored
context.Entry(entityGraph).State = EntityState.Modified	Modified	Modified	Ignored
context.Entry(entityGraph).State = EntityState.Deleted	Deleted	Deleted	Ignored

Add()

The `DbContext.Add` and `DbSet.Add` methods attach an entity graph to a context and set Added `EntityState` to a root and child entities, irrespective of whether they have key values or not.

```
var student = new Student() { //Root entity (with key value)
    StudentId = 1,
    Name = "Bill",
    Address = new StudentAddress()  //Child entity (with key value)
    {
        StudentAddressId = 1,
        City = "Seattle",
        Country = "USA"
    },
    StudentCourses = new List<StudentCourse>() {
                    new StudentCourse(){   Course = new Course(){ CourseName="Machine
Language" } },//Child entity (empty key)
                new StudentCourse(){   Course = new Course(){   CourseId=2 } } //Child
entity (with key value)
        }
};

var context = new SchoolContext();
context.Students.Add(student);

DisplayStates(context.ChangeTracker.Entries());
```

Output:

```
Entity: Student, State: Added
Entity: StudentAddress, State: Added
Entity: StudentCourse, State: Added
Entity: Course, State: Added
Entity: StudentCourse, State: Added
Entity: Course, State: Added
```

The following table lists possible EntityState of each entity in a graph using the `DbContext.Add` or `DbSet.Add` methods.

Method	Root entity with/out Key value	Child Entities with/out Key value
DbContext.Add(entityGraph) or DbSet.Add(entityGraph)	Added	Added

Update()

The `DbContext.Update()` and `DbSet.Update()` methods attach an entity graph to a context and set the `EntityState` of each entity in a graph depending on whether it contains a key property value or not. Consider the following example.

```
var student = new Student() { //Root entity (with key value)
    StudentId = 1,
    Name = "Bill",
    Address = new StudentAddress()  //Child entity (with key value)
    {
        StudentAddressId = 1,
        City = "Seattle",
        Country = "USA"
    },
    StudentCourses = new List<StudentCourse>() {
                new StudentCourse(){   Course = new Course(){ CourseName="Machine
Language" } },//Child entity (empty key)
               new StudentCourse(){   Course = new Course(){   CourseId=2 } } //Child
entity (with key value)
        }
};

var context = new SchoolContext();
context.Update(student);

DisplayStates(context.ChangeTracker.Entries());
```

Output:

```
Entity: Student, State: Modified
Entity: StudentAddress, State: Modified
Entity: StudentCourse, State: Added
Entity: Course, State: Added
Entity: StudentCourse, State: Added
Entity: Course, State: Modified
```

In the above example, the `Update()` method applies the Modified state to the entities which contain non-empty key property values and the Added state to those which contain empty or default CLR key values, irrespective of whether they are a root entity or a child entity.

	Root entity with Key value	Root Entity with Empty or CLR default value	Child Entities with Key value	Child Entities with Empty Key value
Update()				

Update()	Root entity with Key value	Root Entity with Empty or CLR default value	Child Entities with Key value	Child Entities with Empty Key value
DbContext.Update(entityGraph) or DbSet.Update(entityGraph)	Modified	Added	Modified	Added

Remove()

The `DbContext.Remove()` and `DbSet.Remove()` methods set the Deleted `EntityState` to the root entity.

```
var student = new Student() { //Root entity (with key value)
    StudentId = 1,
    Name = "Bill",
    Address = new StudentAddress()  //Child entity (with key value)
    {
        StudentAddressId = 1,
        City = "Seattle",
        Country = "USA"
    },
    StudentCourses = new List<StudentCourse>() {
                new StudentCourse(){   Course = new Course(){ CourseName="Machine Language" } },//Child entity (empty key)
              new StudentCourse(){   Course = new Course(){   CourseId=2 } } //Child entity (with key value)
        }
};

var context = new SchoolContext();
context.Remove(student);

DisplayStates(context.ChangeTracker.Entries());
```

Output:

```
Entity: Student, State: Deleted
Entity: StudentAddress, State: Unchanged
Entity: StudentCourse, State: Added
Entity: Course, State: Added
Entity: StudentCourse, State: Added
Entity: Course, State: Unchanged
```

The following table lists the behaviour of the `Remove()` method on the `EntityState` of each entity.

Remove()	Root entity with Key value	Root Entity with Empty or CLR default value	Child Entities with Key value	Child Entities with Empty Key value
DbContext.Remove(entityGraph) or DbSet.Remove(entityGraph)	Deleted	Exception	Unchanged	Added

Thus, be careful while using the above methods in EF Core.

Learn about the `ChangeTracker.TrackGraph()` method to deal with each entity in an entity graph 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

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.