< Previous Next >

Delete Data in Disconnected Scenario in Entity Framework Core

EF Core API builds and executes the DELETE statement in the database for the entities whose `EntityState` is Deleted. There is no difference in deleting an entity in the connected and disconnected scenario in EF Core. EF Core made it easy to delete an entity from a context which in turn will delete a record in the database using the following methods.

DbContext Methods	DbSet Methods	Description
DbContext.Remove	DbSet.Remove	Attaches the specified entity to the `DbContext` with Deleted state and starts tracking it.
DbContext.RemoveRange	DbSet.RemoveRange	Attaches a collection or array of entities to the `DbContext` with Deleted state and starts tracking them.

The following example demonstrates the different ways of deleting an entity in the disconnected scenario.

```
// entity to be deleted
var student = new Student() {
        StudentId = 1
};

using (var context = new SchoolContext())
{
    context.Remove<Student>(student);

    // or the followings are also valid
    // context.RemoveRange(student);
    //context.Students.Remove(student);
    //context.Students.RemoveRange(student);
    //context.Attach<Student>(student).State = EntityState.Deleted;
    //context.Entry<Student>(student).State = EntityState.Deleted;

    context.SaveChanges();
}
```

In the above example, a `Studnet` entity with the valid `StudentId` is removed from a context using the `Remove()` or `RemoveRange()` method. The data will be deleted from the database on `SaveChanges()`. The above example executes the following delete command in the database:

```
exec sp_executesql N'SET NOCOUNT ON;
DELETE FROM [Students]
WHERE [StudentId] = @p0;
SELECT @@ROWCOUNT;
',N'@p0 int',@p0=1
go
```

Note: The `DbContext.Remove()` and `DbContext.RemoveRange()` methods are newly introduced in EF Core to make the delete operation easy.

Exception:

If the Key value in the specified entity in the `Remove()` or `RemoveRange()` method does not exist in the corresponding database table, then EF Core will throw an exception: The following example will throw an exception.

```
var student = new Student() {
    StudentId = 50
};

using (var context = new SchoolContext()) {

    context.Remove<Student>(student);

    context.SaveChanges();
}
```

In the above example, a `Student` with `StudentId = 50` does not exist in the database. So, EF Core will throw the following `DbUpdateConcurrencyException`:

Database operation expected to affect 1 row(s) but actually affected 0 row(s). Data may have been modified or deleted since entities were loaded.

So, you need to handle the above exception appropriately or make sure that the corresponding data with id exists in the database before deleting it.

```
var student = new Student() {
    StudentId = 50
};

using (var context = new SchoolContext())
{
    try
    {
        context.Remove<Student>(deleteStudent);
        context.SaveChanges();
    }
    catch (DbUpdateConcurrencyException ex)
    {
        throw new Exception("Record does not exist in the database");
    }
    catch (Exception ex)
    {
        throw;
    }
}
```

Delete Multiple Records

You can remove multiple entities in one go by using the `DbContext.RemoveRange()` or `DbSet.RemoveRange()` method.

```
IList<Student> students = new List<Student>() {
    new Student(){ StudentId = 1 },
    new Student(){ StudentId = 2 },
    new Student(){ StudentId = 3 },
    new Student(){ StudentId = 4 }
};

using (var context = new SchoolContext())
{
    context.RemoveRange(students);

    // or
    // context.Students.RemoveRange(students);

    context.SaveChanges();
}
```

The above example will delete 4 records from the database in a **single database trip**. Thus, EF Core improved the performance.

```
exec sp_executesql N'SET NOCOUNT ON;
DELETE FROM [Students]
WHERE [StudentId] = @p0;
SELECT @@ROWCOUNT;


DELETE FROM [Students]
WHERE [StudentId] = @p1;
SELECT @@ROWCOUNT;

DELETE FROM [Students]
WHERE [StudentId] = @p2;
SELECT @@ROWCOUNT;


DELETE FROM [Students]
WHERE [StudentId] = @p3;
SELECT @@ROWCOUNT;

',N'@p0 int,@p1 int',@p0=1,@p1=2,@p2=3,@p3=4
go
```

Delete Related Data

If an entity has relationship with other entities such as one-to-one or one-to-many then deleting related data when the root entity is deleted is depends on how the relationship is configured.

For example, consider that the `Student` and `Grade` entities have a one-to-many relationship. There will be many student records for a particular `GradeId`. EF will throw a reference integrity error if we try to delete a grade which has related `Students` records in the database. To solve this issue, you can define the referential constraint action options using Fluent API. For example, you can configure a cascade delete option for the relationship, as shown below.

```
modelBuilder.Entity<Student>()
    .HasOne<Grade>(s => s.Grade)
    .WithMany(g => g.Students)
    .HasForeignKey(s => s.GradeId)
    .OnDelete(DeleteBehavior.Cascade);
```

Now, if you delete the `Grade` entity, then all the related `Student` records will also be deleted in the database.

There are other referential constraint action options available in EF Core, such as `SetNull`, `ClientSetNull`, and `Restrict`.

< Previous

Next >

ENTITYFRAMEWORKTUTORIAL

✉ 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