< Previous

Next >

Configure Many-to-Many Relationships in Entity Framework Core

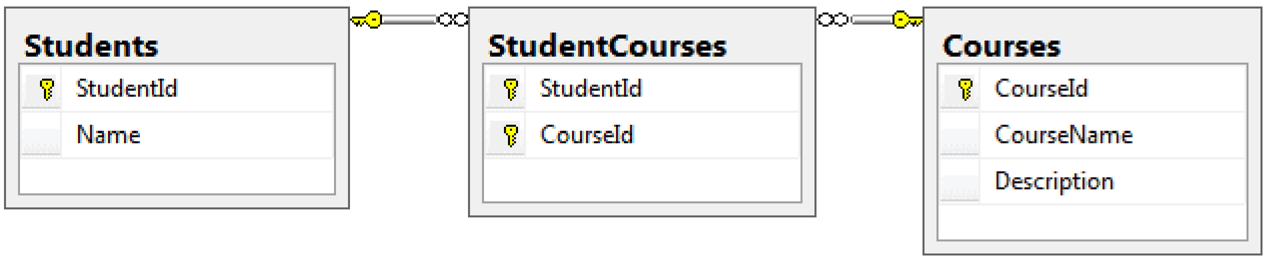
Here you will learn how to configure many-to-many relationships between two entities using Fluent API in Entity Framework Core.

Let's implement a many-to-many relationship between the following `Student` and `Course` entities, where one student can enroll for many courses and, in the same way, one course can be joined by many students.

```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public string Description { get; set; }
}
```

The many-to-many relationship in the database is represented by a joining table which includes the foreign keys of both tables. Also, these foreign keys are composite primary keys.



Convention

There are no default conventions available in Entity Framework Core which automatically configure a many-to-many relationship. You must configure it using Fluent API.

Fluent API

In the Entity Framework 6.x or prior, EF API used to create the joining table for many-to-many relationships. We need not to create a joining entity for a joining table (however, we can of course create a joining entity explicitly in EF 6).

In Entity Framework Core, this has not been implemented yet. We must create a joining entity class for a joining table. The joining entity for the above `Student` and `Course` entities should include a foreign key property and a reference navigation property for each entity.

The steps for configuring many-to-many relationships would the following:

1. Define a new joining entity class which includes the foreign key property and the reference navigation property for each entity.
2. Define a one-to-many relationship between other two entities and the joining entity, by including a collection navigation property in entities at both sides (`Student` and `Course`, in this case).
3. Configure both the foreign keys in the joining entity as a composite key using Fluent API.

So, first of all, define the joining entity `StudentCourse`, as shown below.

```
public class StudentCourse
{
    public int StudentId { get; set; }
    public Student Student { get; set; }

    public int CourseId { get; set; }
    public Course Course { get; set; }
}
```

The above joining entity `StudentCourse` includes reference navigation properties `Student` and `Course` and their foreign key properties `StudentId` and `CourseId` respectively (foreign key properties follow the convention).

Now, we also need to configure two separate one-to-many relationships between `Student` -> `StudentCourse` and `Course` -> `StudentCourse` entities. We can do it by just following the convention for one-to-many relationships, as shown below.

```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }

    public IList<StudentCourse> StudentCourses { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public string Description { get; set; }

    public IList<StudentCourse> StudentCourses { get; set; }
}
```

As you can see above, the `Student` and `Course` entities now include a collection navigation property of `StudentCourse` type. The `StudentCourse` entity already includes the foreign key property and navigation property for both, `Student` and `Course`. This makes it a fully defined one-to-many relationship between `Student` & `StudentCourse` and `Course` & `StudentCourse`.

Now, the foreign keys must be the composite primary key in the joining table. This can only be configured using Fluent API, as below.

```
public class SchoolContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
                   optionsBuilder.UseSqlServer("Server=.\\SQLEXPRESS;Database=EFCore-
SchoolDB;Trusted_Connection=True");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
             modelBuilder.Entity<StudentCourse>().HasKey(sc => new { sc.StudentId,
sc.CourseId });
    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Course> Courses { get; set; }
    public DbSet<StudentCourse> StudentCourses { get; set; }
}
```

In the above code, `modelBuilder.Entity<StudentCourse>().HasKey(sc => new { sc.StudentId, sc.CourseId })` configures `StudentId` and `CourseId` as the composite key.

This is how you can configure many-to-many relationships if entities follow the conventions for one-to-many relationships with the joining entity. Suppose that the foreign key property names do not follow the convention (e.g. SID instead of StudentId and CID instead of CourseId), then you can configure it using Fluent API, as shown below.

```
modelBuilder.Entity<StudentCourse>().HasKey(sc => new { sc.SId, sc.CId });

modelBuilder.Entity<StudentCourse>()
    .HasOne<Student>(sc => sc.Student)
    .WithMany(s => s.StudentCourses)
    .HasForeignKey(sc => sc.SId);


modelBuilder.Entity<StudentCourse>()
    .HasOne<Course>(sc => sc.Course)
    .WithMany(s => s.StudentCourses)
    .HasForeignKey(sc => sc.CId);
```

Note: EF team will include a feature where we don't need to create a joining entity for many-to-many relationships in future. Track this issue on GitHub.

Previous

Next

TUTORIALS

- EF Basics
- EF Core
- EF 6 DB-First
- EF 6 Code-First

updates, tips & tricks about Entity Framework to your inbox.

Email address GO

We respect your privacy.

HOME PRIVACY POLICY ADVERTISE WITH US