


Improve Entity Framework Performance



 Bulk Insert

 Bulk Delete

 Bulk Update

 Bulk Merge

[LEARN MORE](#)

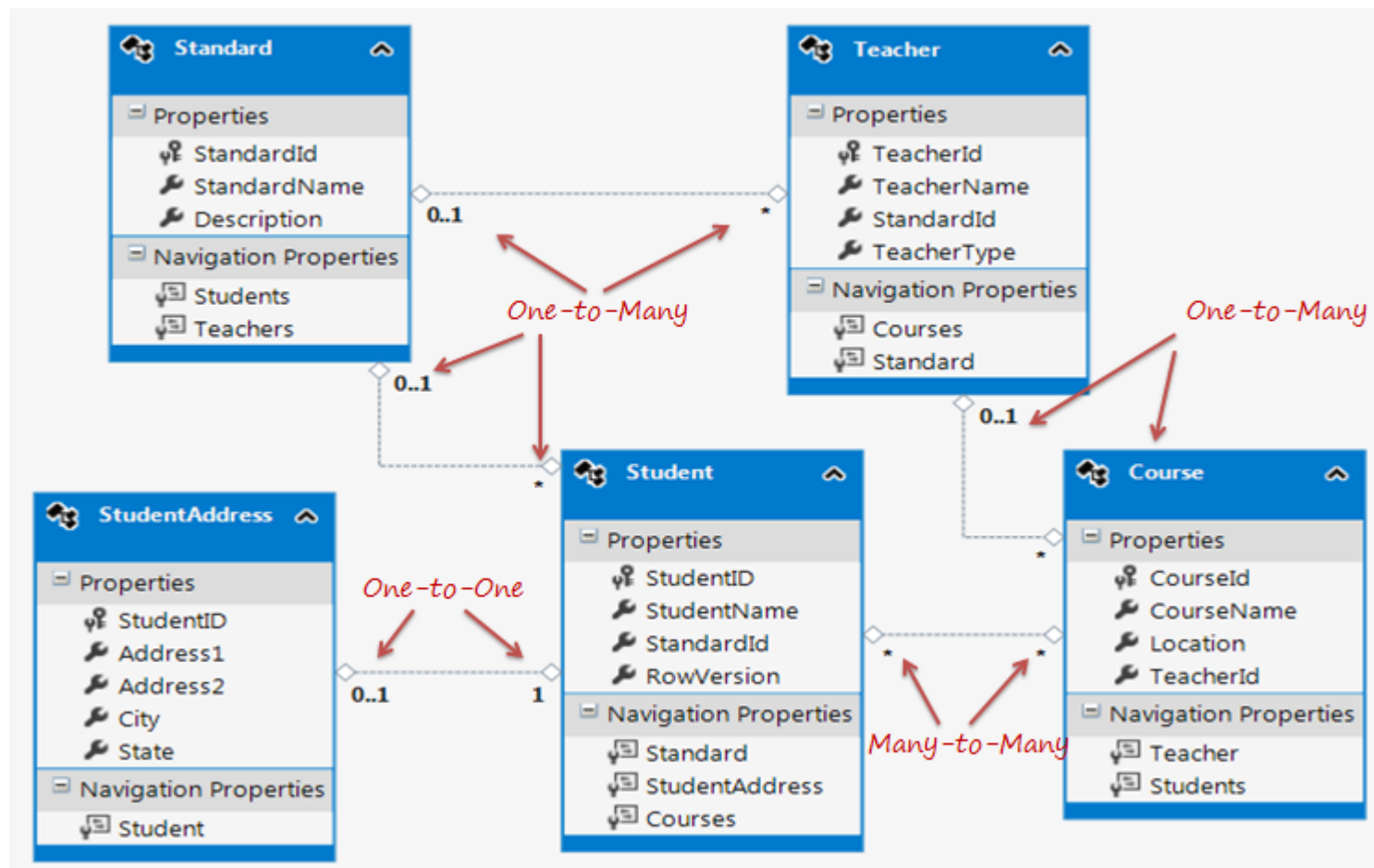
[< Previous](#)[Next >](#)

Relationships between Entities in Entity Framework 6

Here, you will learn how entity framework manages the relationships between entities.

Entity framework supports three types of relationships, same as database: 1) One-to-One 2) One-to-Many, and 3) Many-to-Many.

We have created an Entity Data Model for the SchoolDB database in the [Create Entity Data Model](#) chapter. The following figure shows the visual designer for that EDM with all the entities and relationships among them.



Let's see how each relationship (association) is being managed by entity framework.

One-to-One Relationship

As you can see in the above figure, `Student` and `StudentAddress` have a One-to-One relationship (zero or one). A student can have only one or zero addresses. Entity framework adds the `Student` [reference navigation property](#) into the `StudentAddress` entity and the `StudentAddress` navigation entity into the `Student` entity. Also, the `StudentAddress` entity has both `StudentId` property as `PrimaryKey` and `ForeignKey`, which makes it a one-to-one relationship.

```
public partial class Student
{
    public Student()
    {
        this.Courses = new HashSet<Course>();
    }

    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public Nullable<int> StandardId { get; set; }
    public byte[] RowVersion { get; set; }

    public virtual StudentAddress StudentAddress { get; set; }
}

public partial class StudentAddress
{
    public int StudentID { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string State { get; set; }

    public virtual Student Student { get; set; }
}
```

In the above example, the `StudentId` property needs to be `PrimaryKey` as well as `ForeignKey`. This can be configured using Fluent API in the `OnModelCreating` method of the context class.

One-to-Many Relationship

The `Standard` and `Teacher` entities have a One-to-Many relationship marked by multiplicity where 1 is for One and * is for Many. This means that `Standard` can have many Teachers whereas `Teacher` can associate with only one `Standard`.

To represent this, the `Standard` entity has the [collection navigation property](#) `Teachers` (please notice that it's plural), which indicates that one `Standard` can have a collection of Teachers (many teachers). And the `Teacher` entity has a `Standard` navigation property (reference property), which indicates that `Teacher` is associated with one `Standard`. Also, it contains the `StandardId` foreign key (PK in `Standard` entity). This makes it a One-to-Many relationship.

```
public partial class Standard
{
    public Standard()
    {
        this.Teachers = new HashSet<Teacher>();
    }

    public int StandardId { get; set; }
    public string StandardName { get; set; }
    public string Description { get; set; }

    public virtual ICollection<Teacher> Teachers { get; set; }
}

public partial class Teacher
{
    public Teacher()
    {
        this.Courses = new HashSet<Course>();
    }

    public int TeacherId { get; set; }
    public string TeacherName { get; set; }
    public Nullable<int> TeacherType { get; set; }

    public Nullable<int> StandardId { get; set; }
    public virtual Standard Standard { get; set; }
}
```

Many-to-Many Relationship

The `Student` and `Course` have a Many-to-Many relationship marked by * multiplicity. It means one `Student` can enroll for many Courses and also, one `Course` can be taught to many Students.

The database includes the `StudentCourse` joining table which includes the primary key of both the tables (`Student` and `Course` tables). Entity Framework represents many-to-many relationships by not having the entity set (`DbSet` property) for the joining table in the CSDL and visual designer. Instead it manages this through mapping.

As you can see in the above figure, the `Student` entity includes the [collection navigation property](#) `Courses` and `Course` entity includes the [collection navigation property](#) `Students` to represent a many-to-many relationship between them.

The following code snippet shows the `Student` and `Course` entity classes.

```
public partial class Student
{
    public Student()
    {
        this.Courses = new HashSet<Course>();
    }

    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public Nullable<int> StandardId { get; set; }
    public byte[] RowVersion { get; set; }

    public virtual ICollection<Course> Courses { get; set; }
}

public partial class Course
{
    public Course()
    {
        this.Students = new HashSet<Student>();
    }

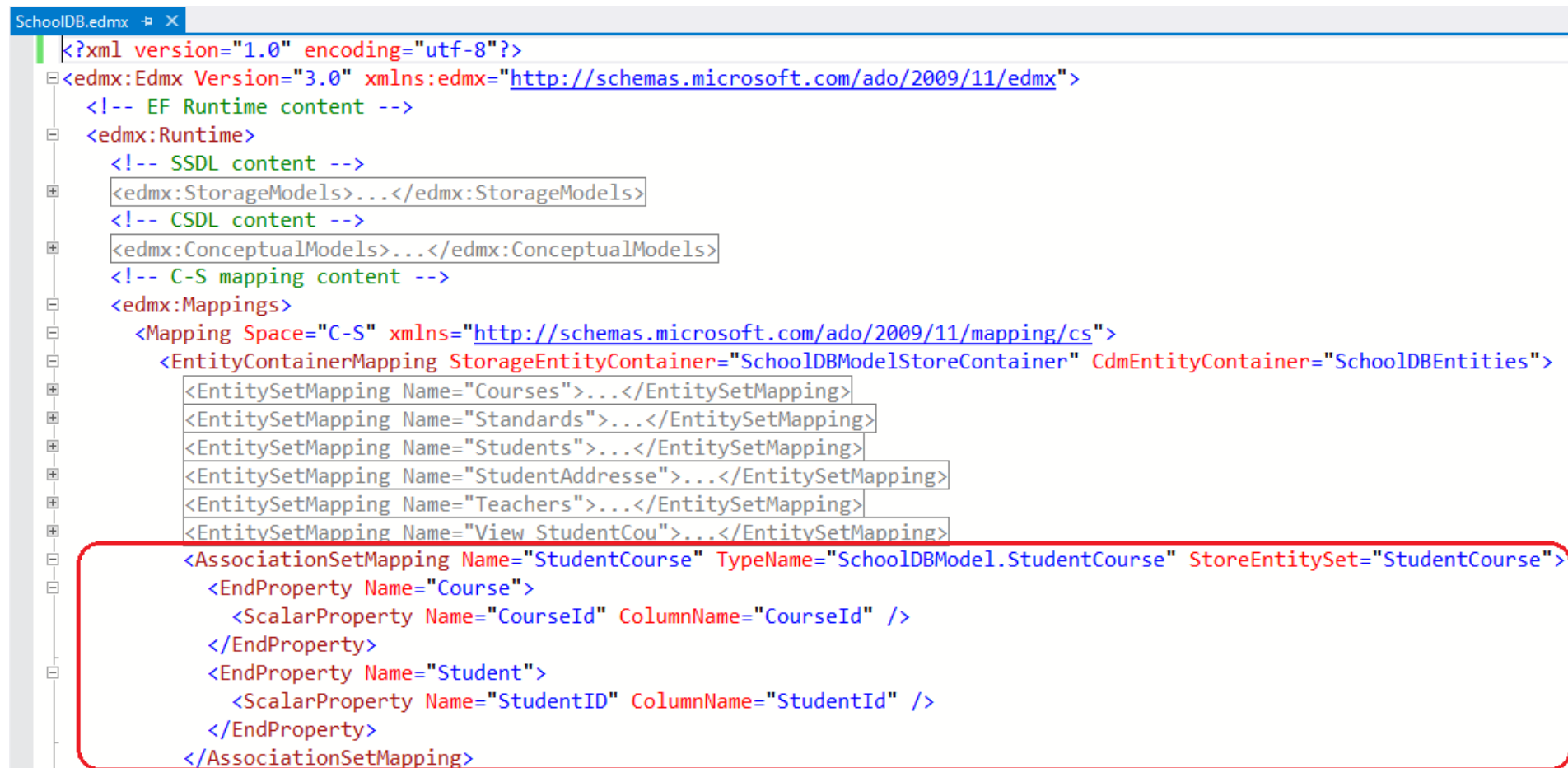
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public System.Data.Entity.Spatial.DbGeography Location { get; set; }

    public virtual ICollection<Student> Students { get; set; }
}
```

Note: Entity framework supports many-to-many relationships only when the joining table (`StudentCourse` in this case) does **NOT** include any columns other than PKs of both tables. If the join tables contain additional columns, such as `DateCreated`, then the EDM creates an entity for the middle table as well and you will have to manage CRUD

operations for many-to-many entities manually.

Open EDM in XML view. You can see that SSDL (storage schema) has the `StudentCourse` entityset, but CSDL doesn't have it. Instead, it's being mapped in the navigation property of the `Student` and `Course` entities. MSL (C-S Mapping) has mapping between `Student` and `Course` put into the `StudentCourse` table in the `<AssociationSetMapping/>` section.



The screenshot shows the XML view of a SchoolDB.edmx file. The XML is structured as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>...</edmx:StorageModels>
    <!-- CSDL content -->
    <edmx:ConceptualModels>...</edmx:ConceptualModels>
    <!-- C-S mapping content -->
    <edmx:Mappings>
      <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">
        <EntityContainerMapping StorageEntityContainer="SchoolDBModelStoreContainer" CdmEntityContainer="SchoolDBEntities">
          <EntitySetMapping Name="Courses">...</EntitySetMapping>
          <EntitySetMapping Name="Standards">...</EntitySetMapping>
          <EntitySetMapping Name="Students">...</EntitySetMapping>
          <EntitySetMapping Name="StudentAdresse">...</EntitySetMapping>
          <EntitySetMapping Name="Teachers">...</EntitySetMapping>
          <EntitySetMapping Name="View StudentCou">...</EntitySetMapping>
          <AssociationSetMapping Name="StudentCourse" TypeName="SchoolDBModel.StudentCourse" StoreEntitySet="StudentCourse">
            <EndProperty Name="Course">
              <ScalarProperty Name="CourseId" ColumnName="CourseId" />
            </EndProperty>
            <EndProperty Name="Student">
              <ScalarProperty Name="StudentID" ColumnName="StudentId" />
            </EndProperty>
          </AssociationSetMapping>
        </EntityContainerMapping>
      </Mapping>
    </edmx:Mappings>
  </edmx:Runtime>
</edmx:Edmx>
```

The `<AssociationSetMapping Name="StudentCourse" ...>` section is highlighted with a red box, indicating the mapping between the `Student` and `Course` entities to the `StudentCourse` table.

Thus, a many-to-many relationship is being managed by C-S mapping in EDM. So, when you add a `Student` in a `Course` or a `Course` in a `Student` entity and when you save it, it will then insert the PK of the added student and course in the `StudentCourse` table. So, this mapping not only enables a convenient association directly between the two entities, but also manages querying, inserts, and updates across these joins.

[Download EF 6 DB-First Demo Project from Github](#)

Managing spare parts and materials mas data is a significant endeavor.

[< 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 Basics](#)

> [EF 6 DB-First](#)

> [EF Core](#)

> [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.

We respect your privacy.

[HOME](#) [PRIVACY POLICY](#) [ADVERTISE WITH US](#)

© 2020 EntityFrameworkTutorial.net. All Rights Reserved.