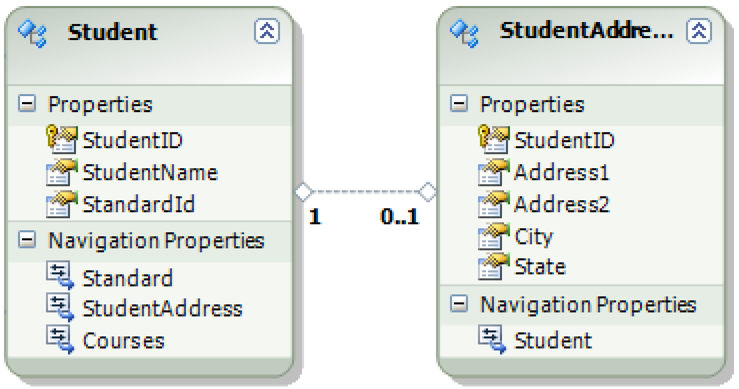
Insert, Update, and Delete with Entity Data Model

Add One-to-One Relationship Entity Graph using DBContext

We will see how to add new Student and StudentAddress entities which has One-to-One relationship that results in new rows in Student and StudentAddress table.

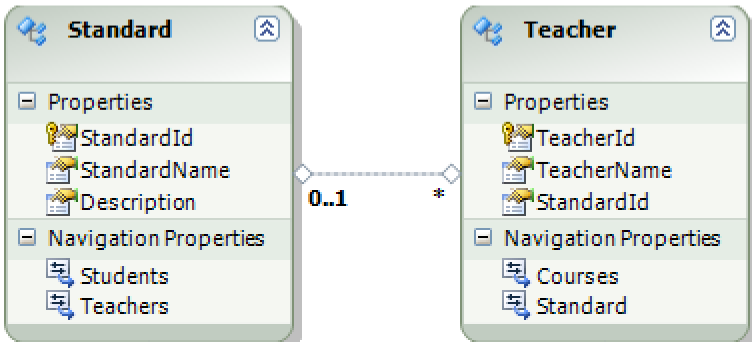


[Student and StudentAddress has One-to-One relationship]

```
// create new student entity object
var student = new Student();

// Assign student name
student.StudentName = "New Student1";

// Create new StudentAddress entity and assign it to student entity
student.StudentAddress = new StudentAddress() { Address1 = "Student1's
Address1",
        Address2 = "Student1's Address2", City = "Student1's City",
        State = "Student1's State" };

//create DBContext object
using (var dbCtx = new SchoolDBEntities())
{
    //Add student object into Student's EntitySet
    dbCtx.Students.Add(student);
    // call SaveChanges method to save student & StudentAddress into database
    dbCtx.SaveChanges();
}
```

Add One-to-Many Relationship Entity Graph using DBContext

We will see how to add new Standard and Teacher entities which has One-to-Many relationship which results in single entry in 'Standard' database table and multiple entry in 'Teacher' table.



[Standard and Teacher has One-to-Many relationship]

```
//Create new standard
var standard = new Standard();
standard.StandardName = "Standard1";

//create three new teachers
var teacher1 = new Teacher();
teacher1.TeacherName = "New Teacher1";

var teacher2 = new Teacher();
teacher2.TeacherName = "New Teacher2";

var teacher3 = new Teacher();
teacher3.TeacherName = "New Teacher3";

//add teachers for new standard
standard.Teachers.Add(teacher1);
standard.Teachers.Add(teacher2);
standard.Teachers.Add(teacher3);

using (var dbCtx = new SchoolDBEntities())
{
    //add standard entity into standards entitySet
    dbCtx.Standards.Add(standard);
    //Save whole entity graph to the database
    dbCtx.SaveChanges();
}
```

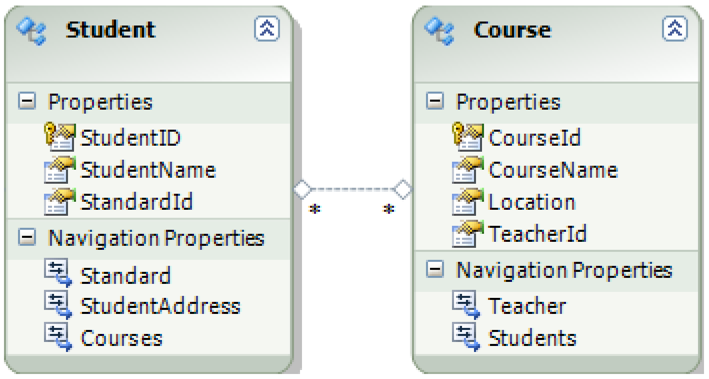
Alternatively, we can also add Standard entity into DBContext.Entry and mark it as Added which result in same insert queries as above:

dbCtx.Entry(standard).State = System.Data.EntityState.Added;

Thus, we have just added Standard entity into DBContext and it has inserted Standard as well as Teachers information into respective database tables. We don't need to add Teacher entity into DBContext separately because Standard and Teacher entities has One-to-Many relationship.

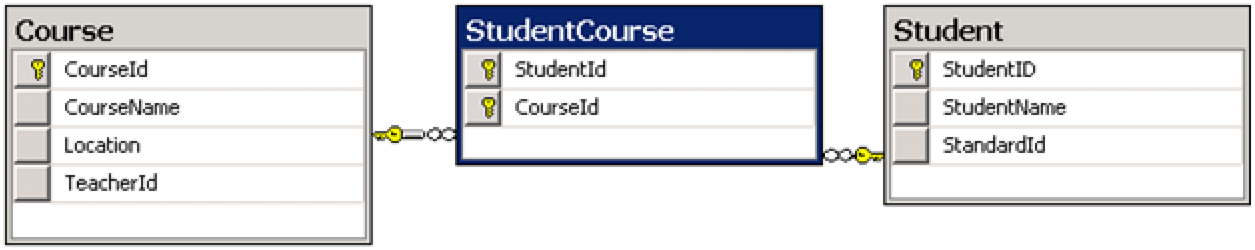
Add Many-to-Many Relationship Entity Graph using DBContext

Student and Course has Many-to-Many relationship which results in insert new rows in Student and StudentCourse tables.



[Student and Course has Many-to-Many relationship]

If you see database design, actually there are three tables participates in Many-to-Many relationship between Student and Course, Student, Course and StudentCourse tables. StudentCourse table consist StudentID and CourseId where both StudentId and CourseId is composite key (combined primary key).



Now let's see code to add these entities into DBContext:

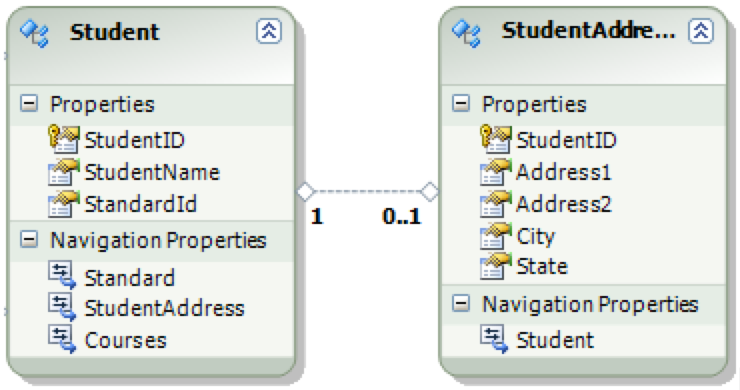
```
//Create student entity
var student1 = new Student();
student1.StudentName = "New Student2";

//Create course entities
var course1 = new Course();
course1.CourseName = "New Course1";
course1.Location = "City1";

var course2 = new Course();
course2.CourseName = "New Course2";
course2.Location = "City2";
```

```
var course3 = new Course();
course3.CourseName = "New Course3";
course3.Location = "City1";

// add multiple courses for student entity
student1.Courses.Add(course1);
student1.Courses.Add(course2);
student1.Courses.Add(course3);

using (var dbCtx = new SchoolDBEntities())
{
    //add student into DBContext
    dbCtx.Students.Add(student1);
    //call SaveChanges
    dbCtx.SaveChanges();
}
```

SaveChanges results in seven inserts query, 1 for student, 3 for Course and 3 for StudentCourse table.

Update One-to-One Entity using DBContext

We will see how to update existing Student and StudentAddress entities in disconnected scenario which has One-to-One relationship.



[Student and StudentAddress has One-to-One relationship]

In disconnected scenario, DBContext doesn't know whether child entity is added, modified or deleted. So DBContext has to find whether StudentAddress entity is newly added or modified or deleted at the time on SaveChanges.

Consider following scenario in disconnected mode for One-to-One entity relationship:

1. User might have added new StudentAddress: Originally there was no StudentAddress is associated with Student.
2. User might have modified existing StudentAddress: StudentAddress has been changed than original

3. User might have deleted existing StudentAddress: Original StudentAddress exist but user has deleted that in disconnected scenario.

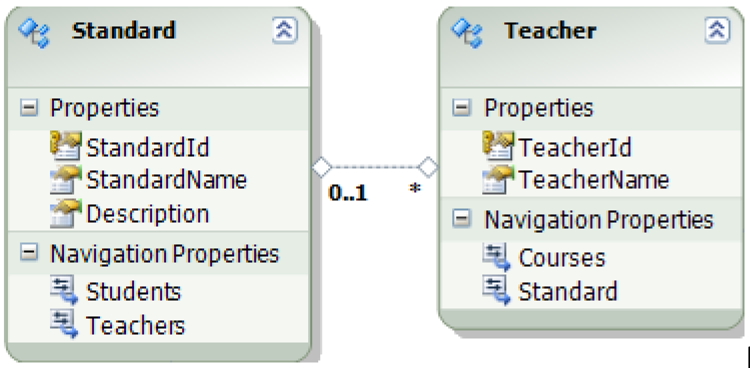
Below code snippet shows how you can handle above scenario and update one-to-one entities:

```
using (var dbCtx = new SchoolDBEntities())
{
    //Get existing StudentAddress for database for the student
    StudentAddress existingStudentAddress =
dbCtx.StudentAddresses.AsNoTracking()
    .Where(addr => addr.StudentID ==
stud.StudentID).FirstOrDefault<StudentAddress>();

    //Mark Student entity as modified
    dbCtx.Entry(stud).State = System.Data.EntityState.Modified;

    //Find whether StudentAddress is modified
    //if existing StudentAddress is not null then delete existing address
    if (existingStudentAddress != null)
        dbCtx.Entry<StudentAddress>(existingStudentAddress).State =
                                        System.Data.EntityState.Deleted;

    //if new StudentAddress is not null then add new StudentAddress.
    //This takes care modified address also.
    if (stud.StudentAddress != null)
        dbCtx.StudentAddresses.Add(stud.StudentAddress);

    dbCtx.SaveChanges();
}
```

As you can see in the above code, first we get the existing StudentAddress from database. (Please note we use AsNoTracking so that DBContext doesn't track entities.) Second, we mark Student entity as modified and then if existing StudentAddress is not null then we delete existing StudentAddress and then if new StudentAddress is not null then we add it in StudentAddress entity set. So this way we are handling all scenarios, add, modified and deleted.

Update One-to-Many Entity using DBContext

We will see how to update existing Standard and Teacher entities in disconnected scenario which has One-to-Many relationship.



[Standard and Teacher has One-to-Many relationship]

The same way as we did it in the case of One-to-One entity relationship, here also we have to find which teacher entities are added, modified or removed from the collection property of Standard entity in disconnected scenario.

Consider following scenario in disconnected mode for One-to-Many entity relationship:

- User might have added new Teachers in the collection
- User might have modified existing Teachers of the collection
- User might have removed existing Teachers from the collection

So you have to find entity state of each entities in the collection.

Below code snippet shows how you can handle above scenario and update One-to-Many entities:

```
using (var dbCtx = new SchoolDBEntities())
{
    //1- Get fresh data from database
    var existingStudent = dbCtx.Students.AsNoTracking().Include(s =>
s.Standard).Include(s => s.Standard.Teachers).Where(s => s.StudentName ==
"updated student").FirstOrDefault<Student>();

    var existingTeachers =
existingStudent.Standard.Teachers.ToList<Teacher>();

    var updatedTeachers = teachers.ToList<Teacher>();

    //2- Find newly added teachers by updatedTeachers (teacher came from
client sided) - existingTeacher = newly added teacher
    var addedTeachers = updatedTeachers.Except(existingTeachers, tchr =>
tchr.TeacherId);

    //3- Find deleted teachers by existing teachers - updatedTeachers =
deleted teachers
    var deletedTeachers = existingTeachers.Except(updatedTeachers, tchr =>
tchr.TeacherId);

    //4- Find modified teachers by updatedTeachers - addedTeachers = modified
teachers
```

```
    var modifiedTeacher = updatedTeachers.Except(addedTeachers, tchr =>
tchr.TeacherId);

    //5- Mark all added teachers entity state to Added
    addedTeachers.ToList<Teacher>().ForEach(tchr => dbCtx.Entry(tchr).State =
System.Data.EntityState.Added);

    //6- Mark all deleted teacher entity state to Deleted
    deletedTeachers.ToList<Teacher>().ForEach(tchr => dbCtx.Entry(tchr).State
= System.Data.EntityState.Deleted);


    //7- Apply modified teachers current property values to existing property
values
    foreach(Teacher teacher in modifiedTeacher)
    {
        //8- Find existing teacher by id from fresh database teachers
        var existingTeacher = dbCtx.Teachers.Find(teacher.TeacherId);

        if (existingTeacher != null)
        {
            //9- Get DBEntityEntry object for each existing teacher entity
            var teacherEntry = dbCtx.Entry(existingTeacher);
            //10- overwrite all property current values from modified
teachers' entity values,
            //so that it will have all modified values and mark entity as
modified
            teacherEntry.CurrentValues.SetValues(teacher);
        }

    }
    //11- Save all above changed entities to the database
    dbCtx.SaveChanges();
}
```

We did following steps in above code to handle all scenarios:

1. Get fresh existing data from database
2. Find newly added teachers by updatedTeachers (teacher came from client sided) - existingTeacher = newly added teacher
3. Find deleted teachers by existing teachers - updatedTeachers = deleted teachers
4. Find modified teachers by updatedTeachers - addedTeachers = modified teachers
5. Mark all added teachers entity state to Added
6. Mark all deleted teacher entity state to Deleted
7. Apply modified teachers current property values to existing property values
8. Find existing teacher by id from fresh database teachers
9. Get DBEntityEntry object for each existing teacher entity
10. Overwrite all property current values from modified teachers' entity values, so that it will have all modified values and mark entity as modified
11. Save all above changed entities to the database

Note: we have used following extension method to compare values from two lists and return entities from one list which is not present into second list. Comparison happening based on passed function:

```
public static IEnumerable<T> Except<T, TKey>(this IEnumerable<T> items,
IEnumerable<T> other,

Func<T, TKey> getKey)
{
    return from item in items
            join otherItem in other on getKey(item)
            equals getKey(otherItem) into tempItems
            from temp in tempItems.DefaultIfEmpty()
            where ReferenceEquals(null, temp) || temp.Equals(default(T))
            select item;

}
```

So this way you can update One-to-Many relationship entities.