

Improve Entity Framework Performance



 **Bulk Insert**

 **Bulk Delete**

 **Bulk Update**

 **Bulk Merge**

LEARN MORE

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

Linq-to-Entities Query

Here, you will learn how to write LINQ-to-Entities queries and get the result in Entity Framework 6.x as well as in Entity Framework Core. Visit [LINQ Tutorials](#) to learn LINQ step by step.

The `DbSet` class is derived from `IQueryable`. So, we can use [LINQ](#) for querying against `DbSet`, which will be converted to an SQL query. EF API executes this SQL query to the underlying database, gets the flat result set, converts it into appropriate entity objects and returns it as a query result.

The following are some of the standard query operators (or extension methods) that can be used with LINQ-to-Entities queries.

LINQ Extension Methods
First()
FirstOrDefault()
Single()
SingleOrDefault()
ToList()
Count()
Min()
Max()
Last()

```
LastOrDefault()
```

```
Average()
```

Find()

In addition to LINQ extension methods, we can use the `Find()` method of `DbSet` to search the entity based on the primary key value.

Let's assume that `SchoolDbEntities` is our `DbContext` class and `Students` is the `DbSet` property.

```
var ctx = new SchoolDBEntities();  
var student = ctx.Students.Find(1);
```

In the above example, `ctx.Student.Find(1)` returns a student record whose `StudentId` is 1 in the database. If no record is found, then it returns null. The above query will execute the following SQL query.

```
SELECT  
[Extent1].[StudentID] AS [StudentID],  
[Extent1].[StudentName] AS [StudentName],  
[Extent1].[StandardId] AS [StandardId]  
FROM [dbo].[Student] AS [Extent1]  
WHERE [Extent1].[StudentId] = @p0',N'@p0 int',@p0=1  
go
```

First/FirstOrDefault

If you want to get a single student object, when there are many students, whose name is "Bill" in the database, then use `First` or `FirstOrDefault`, as shown below:

LINQ Query Syntax:

```
using (var ctx = new SchoolDBEntities())  
{  
    var student = (from s in ctx.Students  
                    where s.StudentName == "Bill"  
                    select s).FirstOrDefault<Student>();  
}
```

LINQ Method Syntax:

```
using (var ctx = new SchoolDBEntities())
{
    var student = ctx.Students
        .Where(s => s.StudentName == "Bill")
        .FirstOrDefault<Student>();
}
```

EF 6 executes the following SQL query in the database for the above LINQ query.

```
SELECT TOP (1)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
WHERE 'Bill' = [Extent1].[StudentName]
```

EF Core executes the following query in the database.

```
SELECT TOP (1)
[s].[StudentId], [s].[DoB], [s].[FirstName], [s].[GradeId],
[s].[LastName], [s].[MiddleName]
FROM [Students] AS [s]
WHERE [s].[FirstName] = N'Bill'
```

Parameterized Query

EF builds and executes a parameterized query in the database if the LINQ-to-Entities query uses parameters, such as below.

```
using (var ctx = new SchoolDBEntities())
{
    string name = "Bill";
    var student = ctx.Students
        .Where(s => s.StudentName == name)
        .FirstOrDefault<Student>();
}
```

The above query will result into the following SQL query in EF 6.

```

SELECT TOP (1)
[Extent1].[StudentId] AS [StudentId],
[Extent1].[Name] AS [Name]
FROM [dbo].[Student] AS [Extent1]
WHERE ([Extent1].[Name] = @p__linq__0) OR (([Extent1].[Name] IS NULL)
AND (@p__linq__0 IS NULL))',N'@p__linq__0
nvarchar(4000)',@p__linq__0=N'Bill'

```

The difference between `First` and `FirstOrDefault` is that `First()` will throw an exception if there is no result data for the supplied criteria, whereas `FirstOrDefault()` returns a default value (null) if there is no result data.

ToList

The `ToList` method returns the collection result. If you want to list all the students with the same name then use `ToList()`:

```

using (var ctx = new SchoolDBEntities())
{
    var studentList = ctx.Students.Where(s => s.StudentName == "Bill").ToList();
}

```

We may also use `ToArray`, `ToDictionary` or `ToLookup`. The above query would result in the following database query:

```

SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
WHERE 'Bill' = [Extent1].[StudentName]
go

```

GroupBy

Use the `group by` operator or `GroupBy` extension method to get the result based on the group by the particular property of an entity.

The following example gets the results grouped by each `Standard`. Use the `foreach` loop to iterate the group.

LINQ Query Syntax:

```
using (var ctx = new SchoolDBEntities())
{
    var students = from s in ctx.Students
                   group s by s.StandardId into studentsByStandard
                   select studentsByStandard;

    foreach (var groupItem in students)
    {
        Console.WriteLine(groupItem.Key);

        foreach (var stud in groupItem)
        {
            Console.WriteLine(stud.StudentId);
        }
    }
}
```

LINQ Method Syntax:

```
using (var ctx = new SchoolDBEntities())
{
    var students = ctx.Students.GroupBy(s => s.StandardId);

    foreach (var groupItem in students)
    {
        Console.WriteLine(groupItem.Key);

        foreach (var stud in groupItem)
        {
            Console.WriteLine(stud.StudentId);
        }
    }
}
```

The above query will execute the following database query:

```
SELECT
[Project2].[C1] AS [C1],
[Project2].[StandardId] AS [StandardId],
[Project2].[C2] AS [C2],
[Project2].[StudentID] AS [StudentID],
[Project2].[StudentName] AS [StudentName],
[Project2].[StandardId1] AS [StandardId1]
FROM ( SELECT
    [Distinct1].[StandardId] AS [StandardId],
    1 AS [C1],
    [Extent2].[StudentID] AS [StudentID],
    [Extent2].[StudentName] AS [StudentName],
    [Extent2].[StandardId] AS [StandardId1],
    CASE WHEN ([Extent2].[StudentID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS
[C2]
FROM (SELECT DISTINCT
    [Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1] ) AS [Distinct1]
LEFT OUTER JOIN [dbo].[Student] AS [Extent2] ON ([Distinct1].[StandardId] =
[Extent2].[StandardId]) OR (([Distinct1].[StandardId] IS NULL) AND ([Extent2].
[StandardId] IS NULL))
) AS [Project2]
ORDER BY [Project2].[StandardId] ASC, [Project2].[C2] ASC
go
```

OrderBy

Use the `OrderBy` operator with ascending/descending keywords in LINQ query syntax to get the sorted entity list.

```
using (var ctx = new SchoolDBEntities())
{
    var students = from s in ctx.Students
                    orderby s.StudentName ascending
                    select s;
}
```

Use the `OrderBy` or `OrderByDescending` method to get the sorted entity list.

```
using (var ctx = new SchoolDBEntities())
{
    var students = ctx.Students.OrderBy(s => s.StudentName).ToList();
    // or descending order
    var descStudents = ctx.Students.OrderByDescending(s =>
s.StudentName).ToList();
}
```

The above query will execute the following database query:

```
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
ORDER BY [Extent1].[StudentName] ASC
go
```

Anonymous Object Result

LINQ-to-Entities queries do not always have to return entity objects. We may choose some of the properties of an entity as a result.

The following query returns a list of anonymous objects which contains `StudentId` and `StudentName` properties.

LINQ Query Syntax:

```
using (var ctx = new SchoolDBEntities())
{
    var anonymousObjResult = from s in ctx.Students
                             where s.StandardId == 1
                             select new {
                                 Id = st.StudentId,
                                 Name = st.StudentName
                             };

    foreach (var obj in anonymousObjResult)
    {
        Console.Write(obj.Name);
    }
}
```

LINQ Method Syntax:

```
using (var ctx = new SchoolDBEntities())
{
    var anonymousObjResult = ctx.Students
        .Where(st => st.Standard == 1)
        .Select(st => new {
            Id = st.StudentId,
            Name = st.StudentName });

    foreach (var obj in anonymousObjResult)
    {
        Console.Write(obj.Name);
    }
}
```

The above query will execute the following database query:

```
SELECT
[s].[StudentID] AS [Id], [s].[StudentName] AS [Name]
FROM [Student] AS [s]
WHERE [s].[StandardId] = 1
go
```


The projectionResult in the above query will be the anonymous type, because there is no class/entity which has these properties. So, the compiler will mark it as anonymous.

Nested queries

You can also execute nested LINQ-to-entity queries as shown below:

```
using (SchoolDBEntities context = new SchoolDBEntities())
{
    var nestedQuery = from s in context.Students
        from c in s.Courses
        where s.StandardId == 1
        select new { s.StudentName, c };

    var result = nestedQuery.ToList();
}
```



Index	StudentName	c (Course)
[0]	New Student	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6}
[1]	New Student	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6}
[2]	New Student	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6}
[3]	Bill	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94}
[4]	Bill	{System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94}

The nested query shown above will result in an anonymous list with a `StudentName` and `Course` object.

```
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Join1].[CourseId1] AS [CourseId],
[Join1].[CourseName] AS [CourseName],
[Join1].[Location] AS [Location],
[Join1].[TeacherId] AS [TeacherId]
FROM [dbo].[Student] AS [Extent1]
INNER JOIN (SELECT [Extent2].[StudentId] AS [StudentId],
    [Extent3].[CourseId] AS [CourseId1], [Extent3].[CourseName] AS [CourseName],
    [Extent3].[Location] AS [Location], [Extent3].[TeacherId] AS [TeacherId]
FROM [dbo].[StudentCourse] AS [Extent2]
INNER JOIN [dbo].[Course] AS [Extent3]
    ON [Extent3].[CourseId] = [Extent2].[CourseId] ) AS [Join1]
    ON [Extent1].[StudentID] = [Join1].[StudentId]
WHERE 1 = [Extent1].[StandardId]
go
```

In this way, you can do a projection of the result, as you want the data to be.

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

[< 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 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](#)

© 2020 EntityFrameworkTutorial.net. All Rights Reserved.