Универзитет у Београду
Електротехнички факултет

Entity Framework object-relational mapping

Author: Nemanja Kojic, MScEE

Prerequisites

- .NET Framework
- SQL Server
- C#
- Visual Studio
- EF versions: 3.5, 4.0, 4.1, 4.3, 5.0, 6.0 (latest)

What is ORM?

- Object-Relational Mapping/Mapper
- Bridges the gap between two paradigms:
  - OO paradigm (objects, inheritance, encapsulation)
  - Relational (tables, columns, constraints, sql,...)
- Automates CRUD operations
- Translates OO actions into relational queries.
- Frameworks: DataObjects.Net, Nhibernate, OpenAccess, EntityFramework

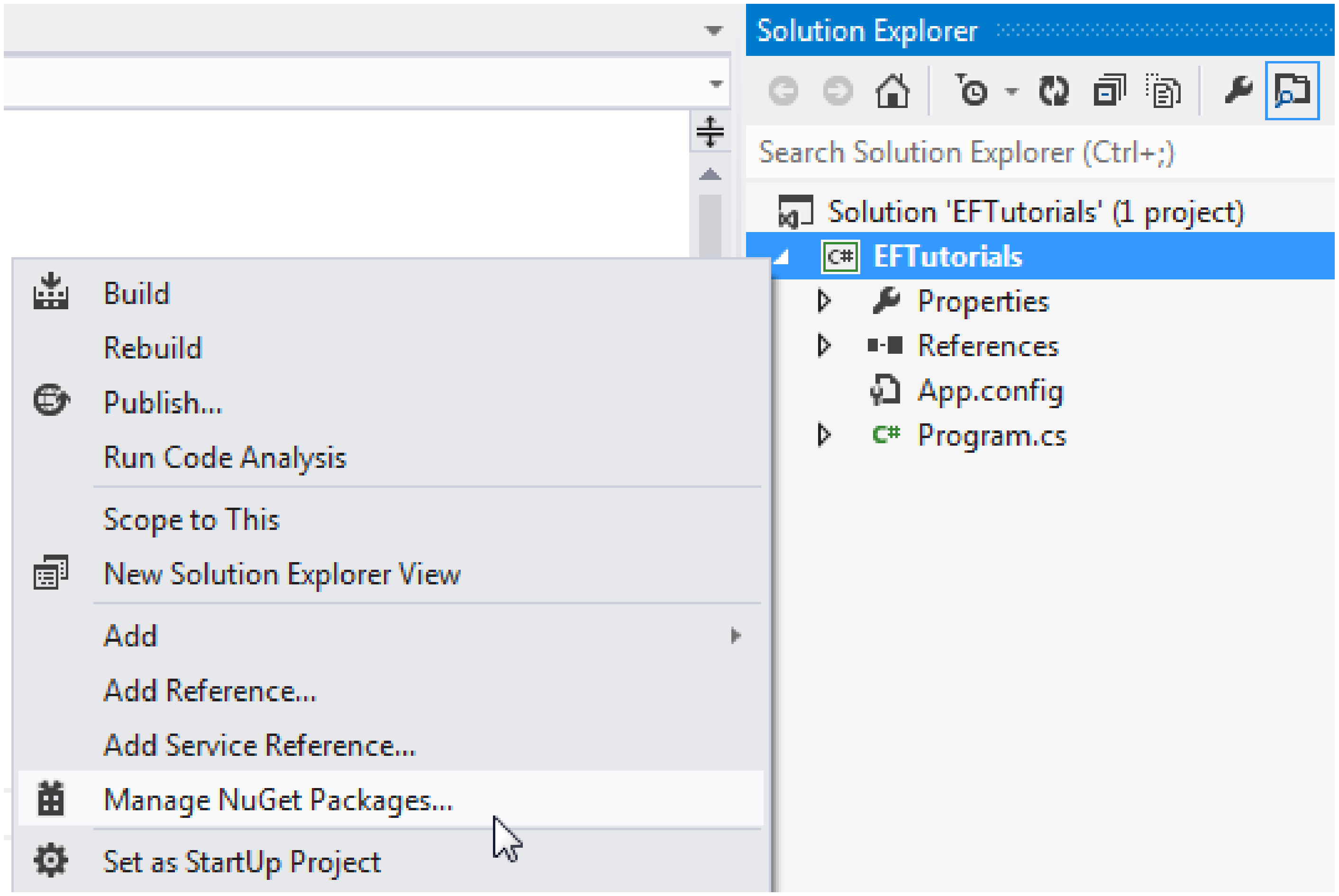
What is Entity Framework?

- Microsoft ADO .NET Entity Framework
- https://entityframework.codeplex.com
- It is an open-source ORM framework
  - Enhancement to ADO .NET
  - Automated mechanism for accessing and storing data in the database
- Enables developers to deal with objects
- No SQL, no tables, no Joins, etc.

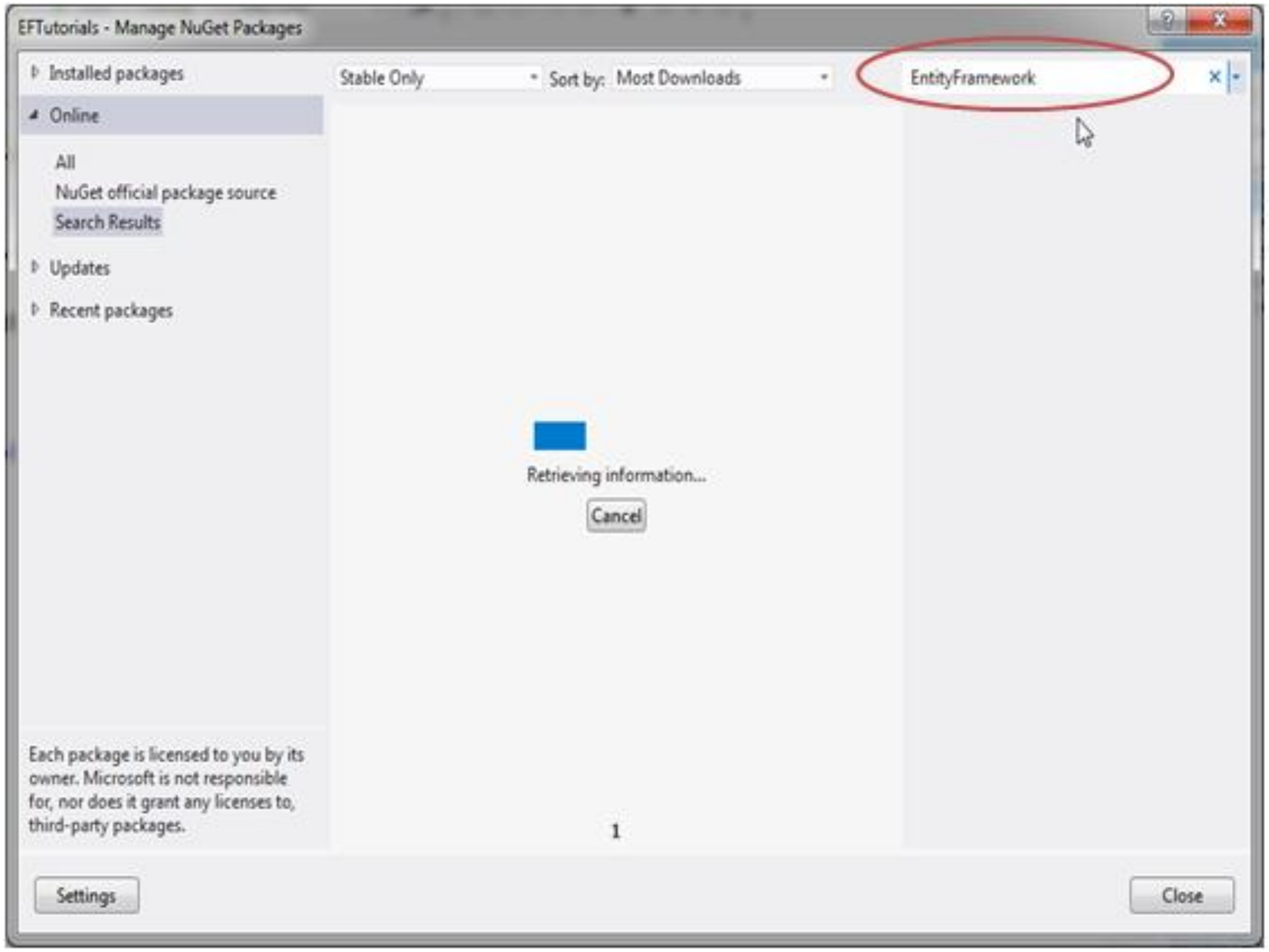
EF distribution

- EF 5.0
  - Part of EF included in NuGet package, and
  - Part of EF included in .NET Framework
- EF 6.0
  - Included in EntityFramework.dll
  - Independent of .NET Framework
- .Net Framework 4.5

Install EF via NuGet

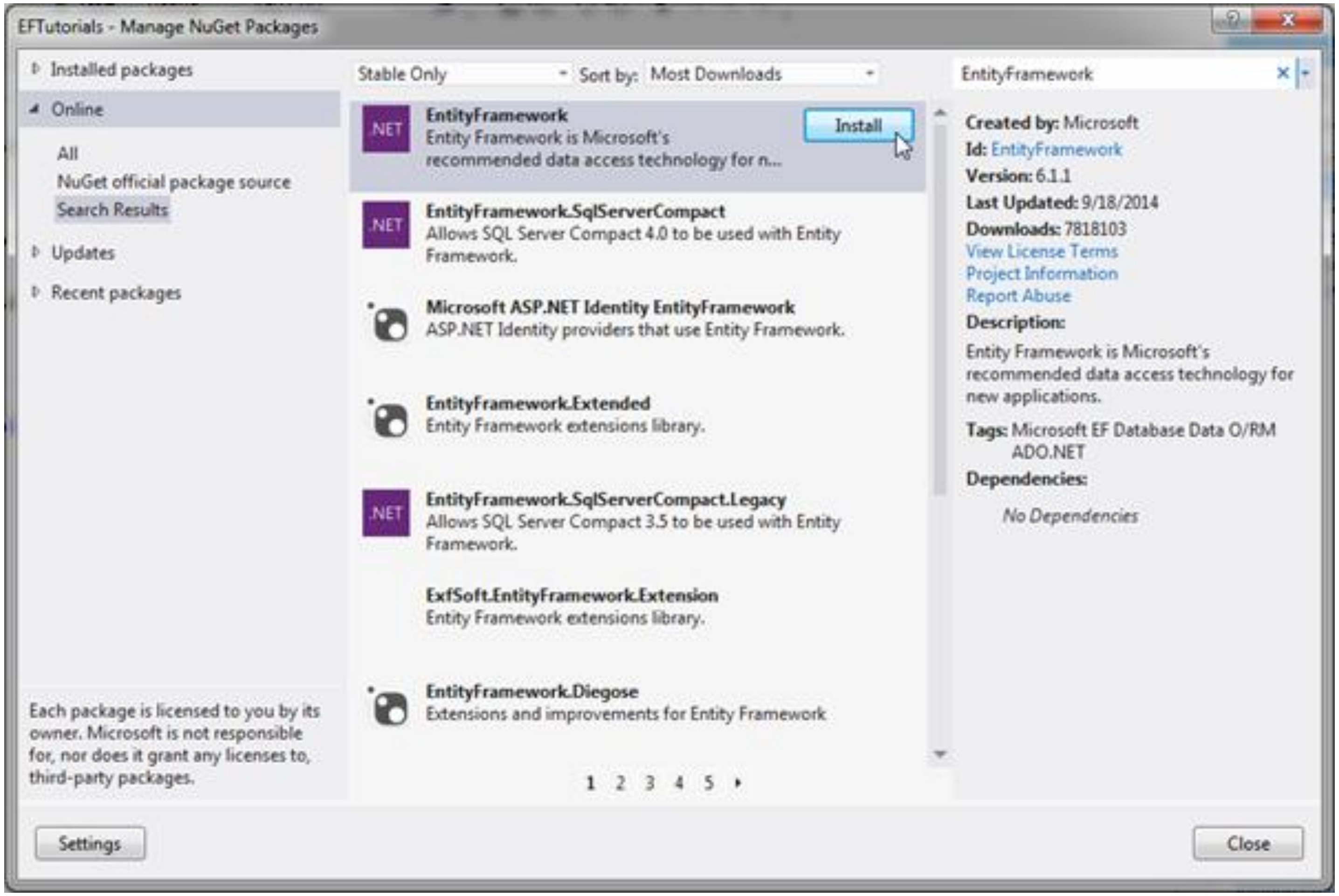


Install EF via NuGet



Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Install EF via NuGet



EFTutorials - Manage NuGet Packages

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

EntityFramework
Entity Framework is Microsoft's recommended data access technology for n...

EntityFramework.SqlServerCompact
Allows SQL Server Compact 4.0 to be used with Entity Framework.

Microsoft ASP.NET Identity EntityFramework
ASP.NET Identity providers that use Entity Framework.

EntityFramework.Extended
Entity Framework extensions library.

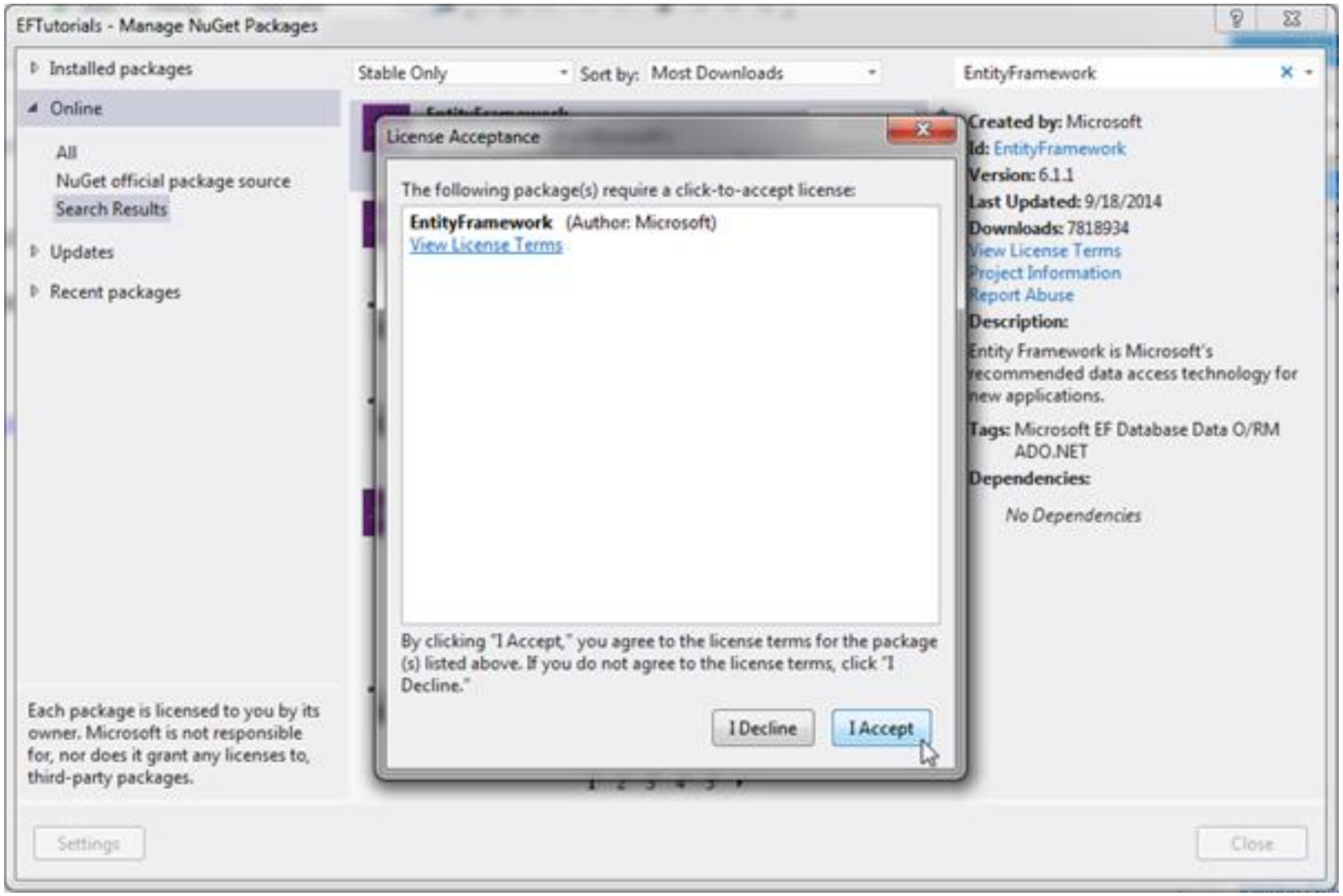
EntityFramework.SqlServerCompact.Legacy
Allows SQL Server Compact 3.5 to be used with Entity Framework.

ExfSoft.EntityFramework.Extension
Entity Framework extensions library.

EntityFramework.Diegose
Extensions and improvements for Entity Framework

Created by: Microsoft
Id: EntityFramework
Version: 6.1.1
Last Updated: 9/18/2014
Downloads: 7818103
View License Terms
Project Information
Report Abuse
Description:
Entity Framework is Microsoft's recommended data access technology for new applications.
Tags: Microsoft EF Database Data O/RM ADO.NET
Dependencies:
No Dependencies

Install EF via NuGet



License Acceptance

The following package(s) require a click-to-accept license:

EntityFramework (Author: Microsoft)
View License Terms

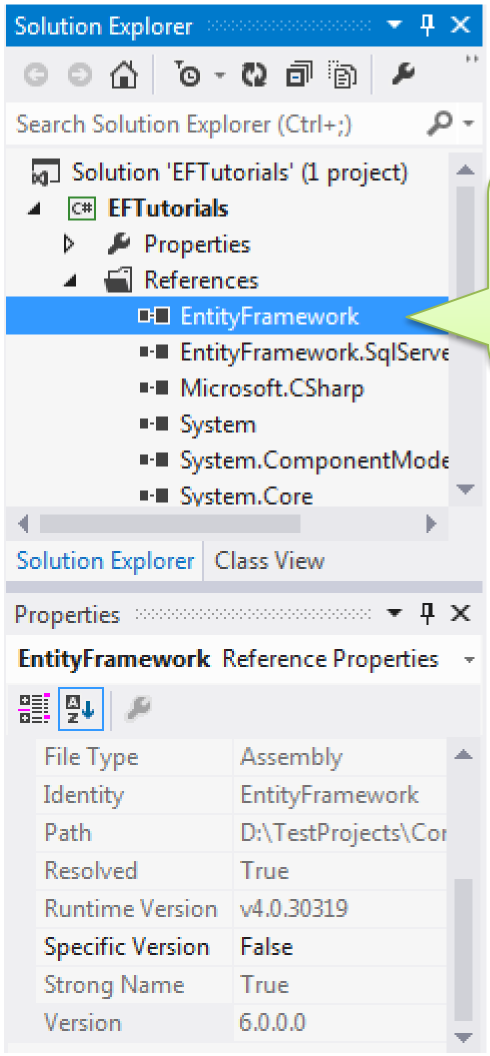
By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."

I Decline | I Accept

Created by: Microsoft
Id: EntityFramework
Version: 6.1.1
Last Updated: 9/18/2014
Downloads: 7818934
View License Terms
Project Information
Report Abuse
Description:
Entity Framework is Microsoft's recommended data access technology for new applications.
Tags: Microsoft EF Database Data O/RM ADO.NET
Dependencies:
No Dependencies

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

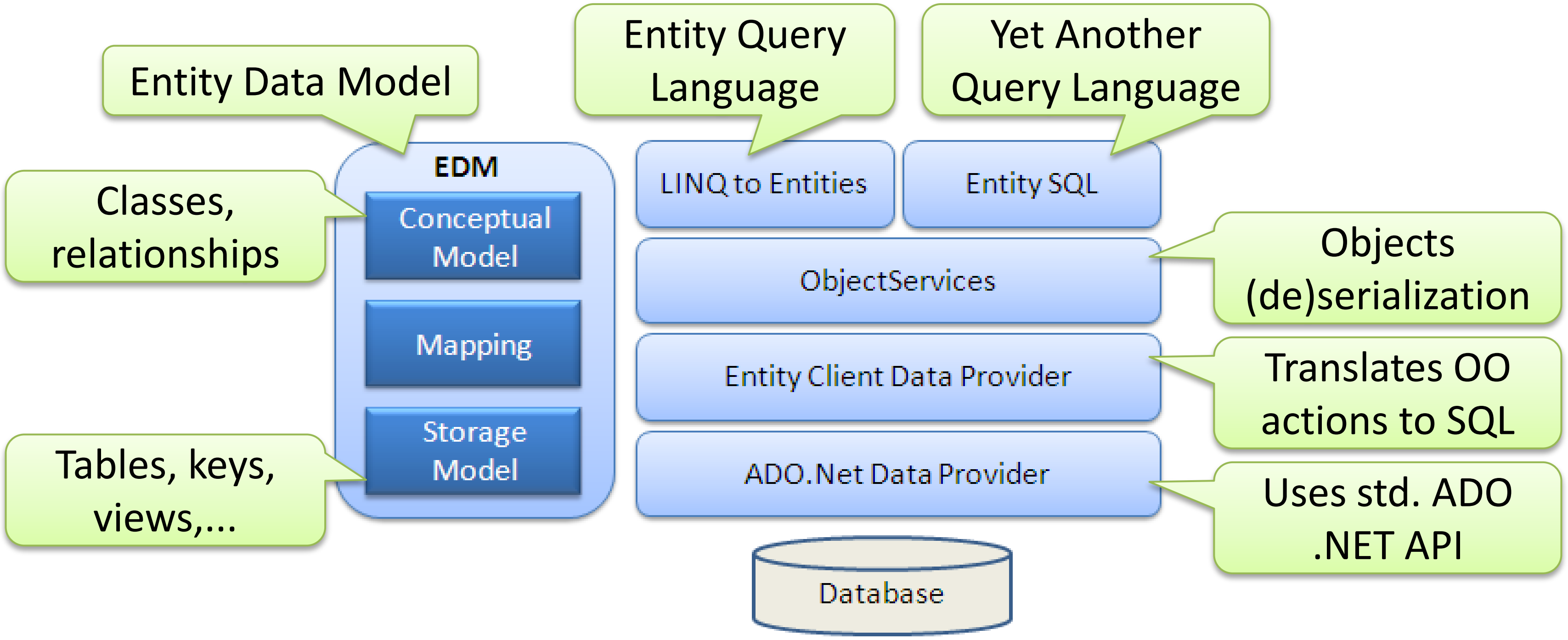
Install EF via NuGet



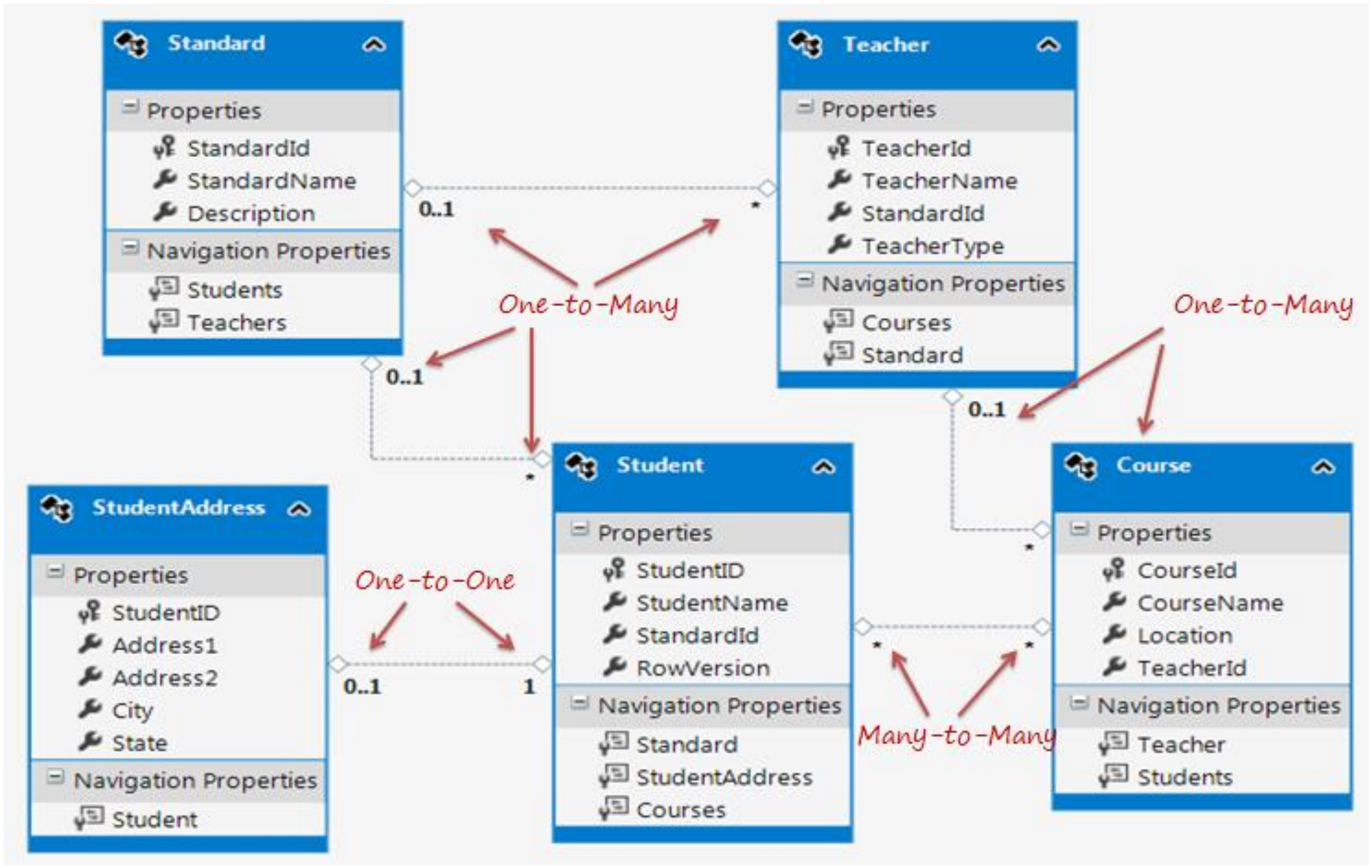
File Type	Assembly
Identity	EntityFramework
Path	D:\TestProjects\Cor
Resolved	True
Runtime Version	v4.0.30319
Specific Version	False
Strong Name	True
Version	6.0.0.0

After installation, make sure that appropriate version of EntityFramework.dll is included in the project.

EF Architecture



Entity Relationships



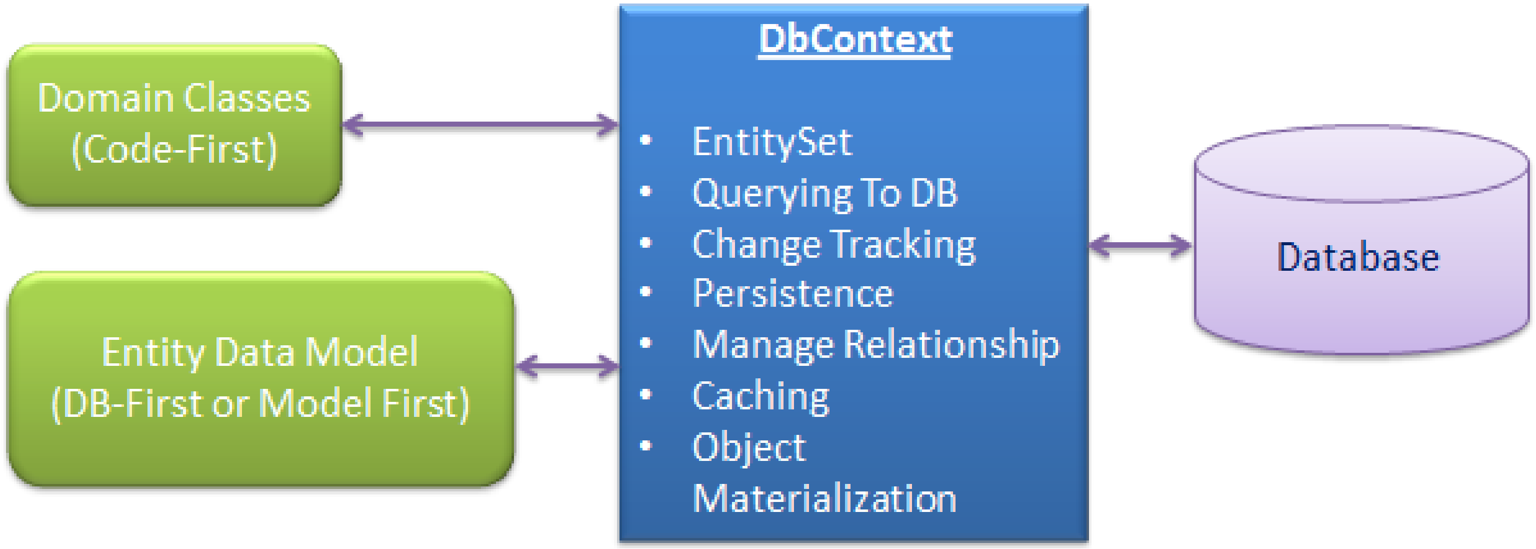
Relationship mapping

```
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 Standard Standard { get; set; }
    public virtual StudentAddress StudentAddress { get; set; }
    public virtual ICollection<Course> Courses { 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; }
 }
 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 Nullable<int> TeacherId { get; set; }

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

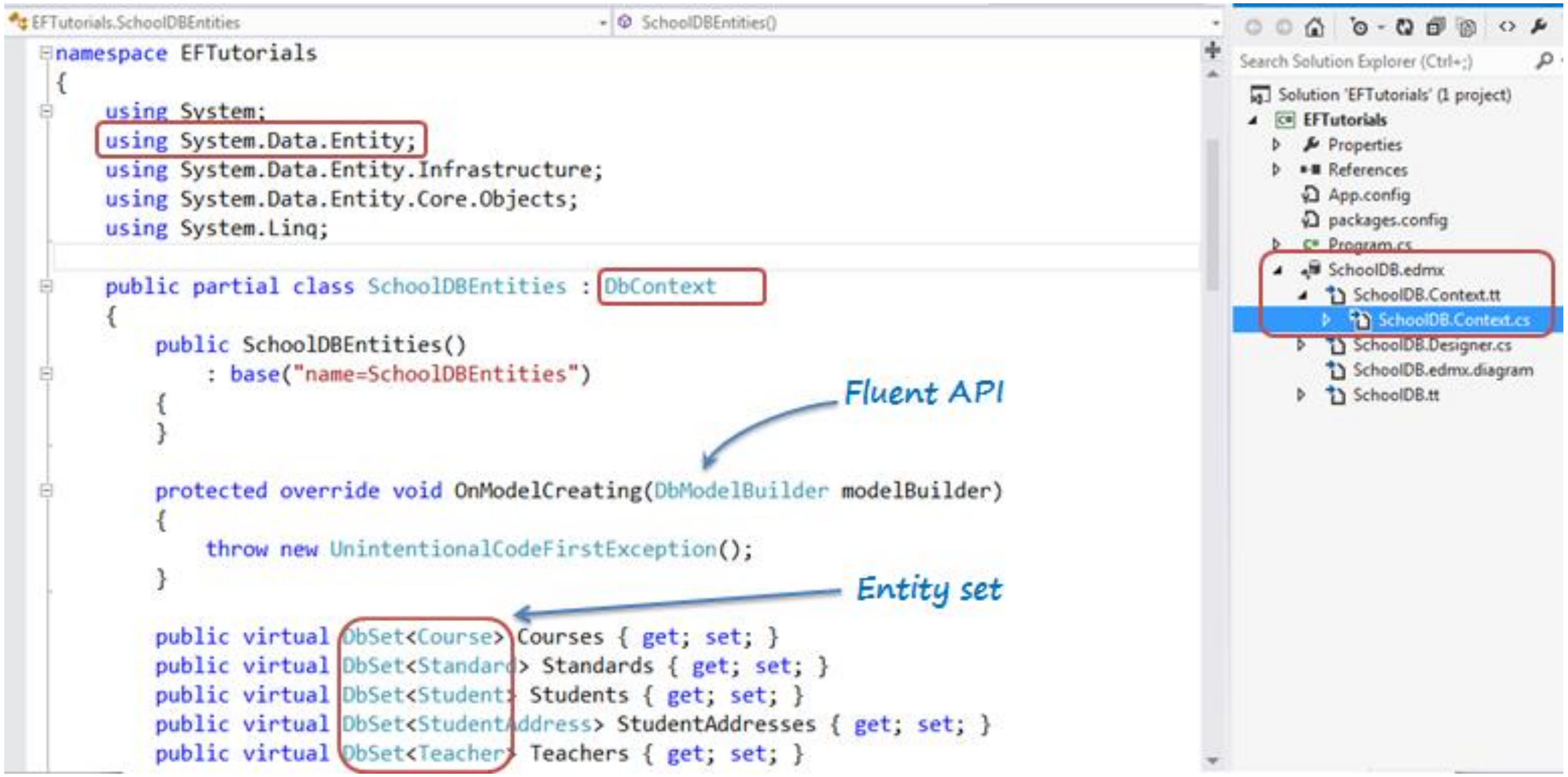
```
public partial class Standard
{
    public Standard()
    { this.Students = new HashSet<Student>();
      this.Teachers = new HashSet<Teacher>();
    }
    public int StandardId { get; set; }
    public string StandardName { get; set; }
    public string Description { get; set; }

    public virtual ICollection<Student> Students { 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> StandardId { get; set; }
    public Nullable<int> TeacherType { get; set; }

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

DbContext

- Bridge between the database and domain objects



DbContext class

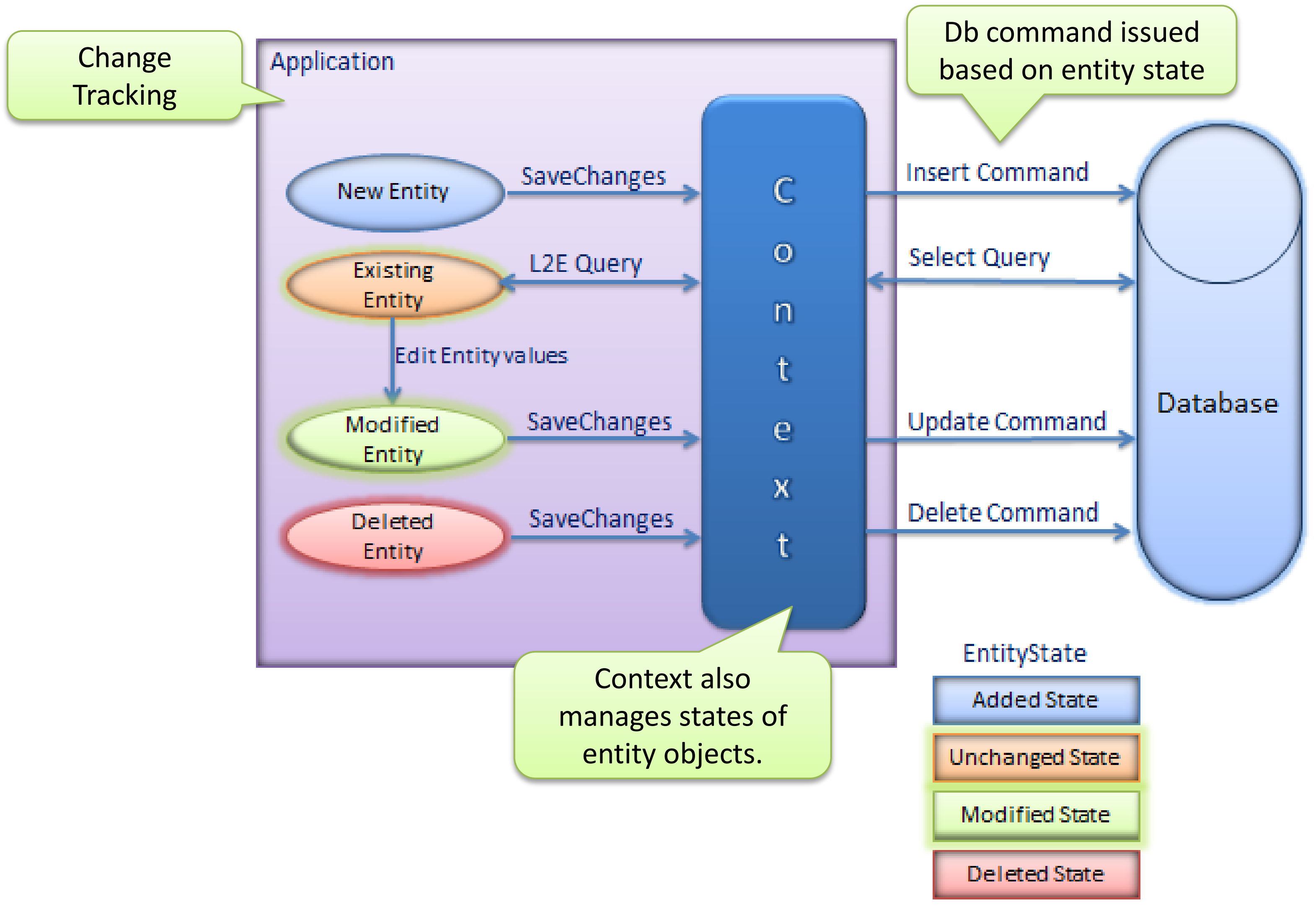


```
namespace EFTutorials
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Data.Entity.Core.Objects;
    using System.Linq;

    public partial class SchoolDBEntities : DbContext
    {
        public SchoolDBEntities()
            : base("name=SchoolDBEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Course> Courses { get; set; }
        public virtual DbSet<Standard> Standards { get; set; }
        public virtual DbSet<Student> Students { get; set; }
        public virtual DbSet<StudentAddress> StudentAddresses { get; set; }
        public virtual DbSet<Teacher> Teachers { get; set; }
```

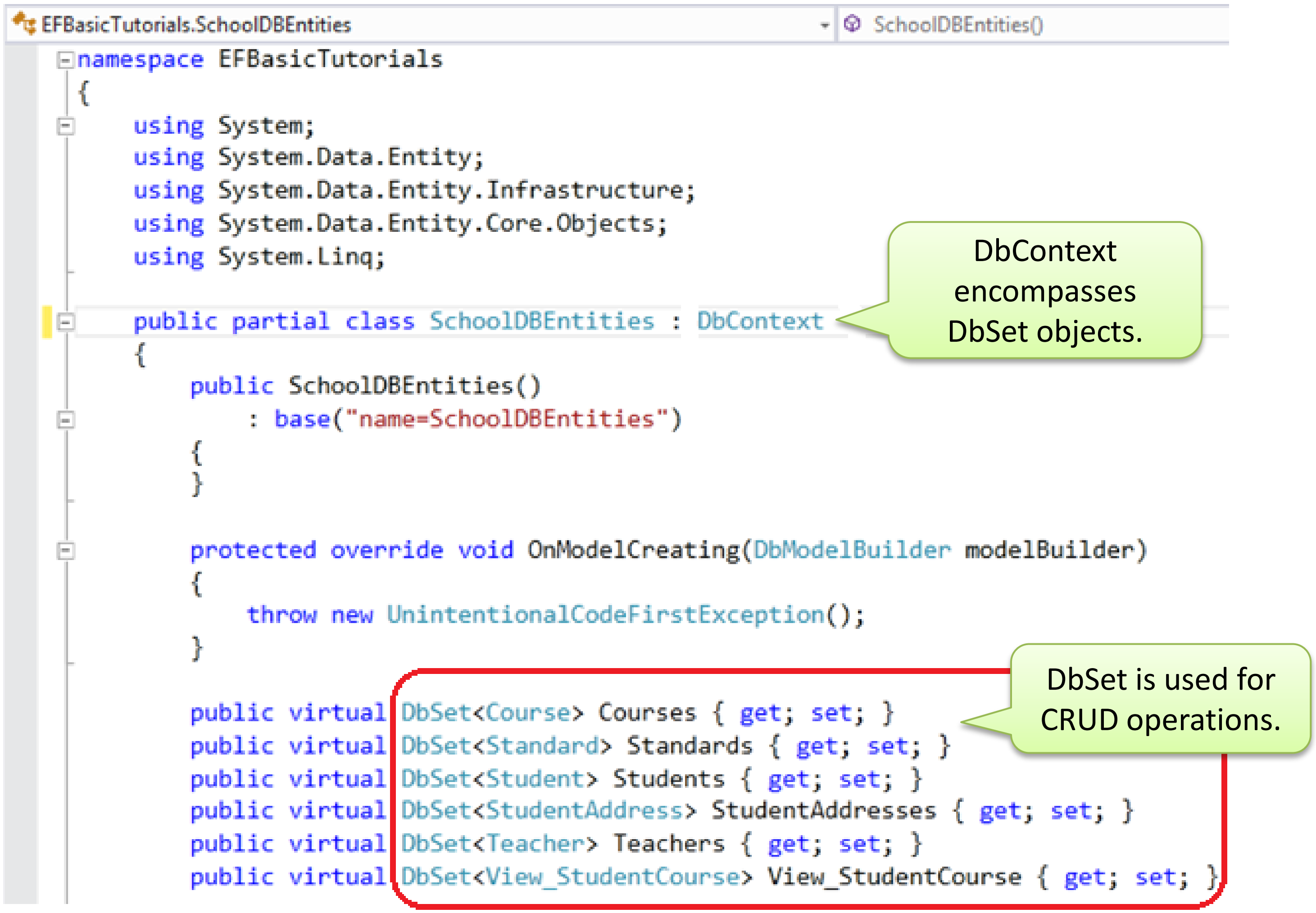
```
using (var ctx = new SchoolDBEntities())
{

    //Can perform CRUD operation using ctx here..
}
```

Intantiating DbContext

Entity Lifecycle



DbSet class



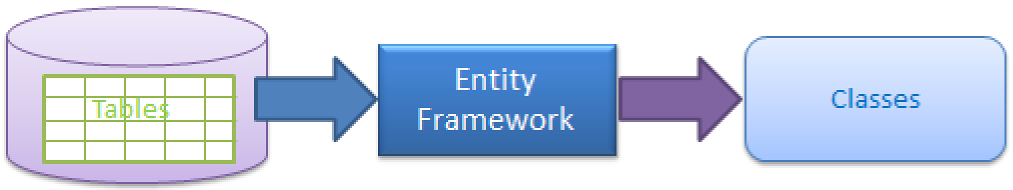
```
namespace EFBasicTutorials
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;
    using System.Data.Entity.Core.Objects;
    using System.Linq;

    public partial class SchoolDBEntities : DbContext
    {
        public SchoolDBEntities()
            : base("name=SchoolDBEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Course> Courses { get; set; }
        public virtual DbSet<Standard> Standards { get; set; }
        public virtual DbSet<Student> Students { get; set; }
        public virtual DbSet<StudentAddress> StudentAddresses { get; set; }
        public virtual DbSet<Teacher> Teachers { get; set; }
        public virtual DbSet<View_StudentCourse> View_StudentCourse { get; set; }
```

DbContext encompasses DbSet objects.
DbSet is used for CRUD operations.

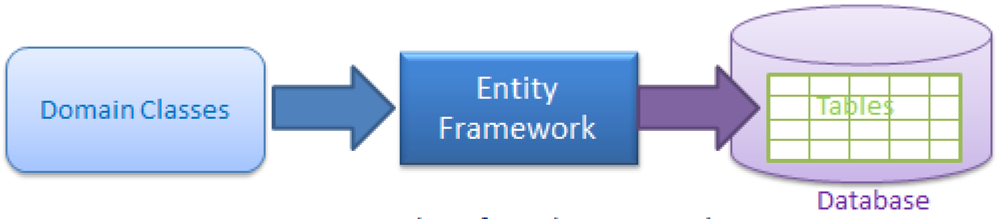
DbSet operations

Method name	Return type	Example
Add	Added entity type	dbcontext.Students.Add(studentEntity)
Attach(Entity)	Passed entity	dbcontext.Students.Attach(studentEntity);
Create	Entity	var newStudentEntity =dbcontext.Students.Create();
Find(int)	Entity type	//Find student whose StudentID is 1 Student studEntity = dbcontext.Students.Find(1);
Include	DBQuery	var studentList = dbcontext.Students.Include ("StudentAddress").ToList<Student>(); var studentList = dbcontext.Students.Include (s=>s.StudentAddress).ToList<Student>();
Remove	Removed entity	dbcontext.Students.Remove(studentEntity);
SqlQuery	DBSqlQuery	var studentEntity = dbcontext.Students.SqlQuery ("select * from student where studentid =1") .FirstOrDefault<Student>()

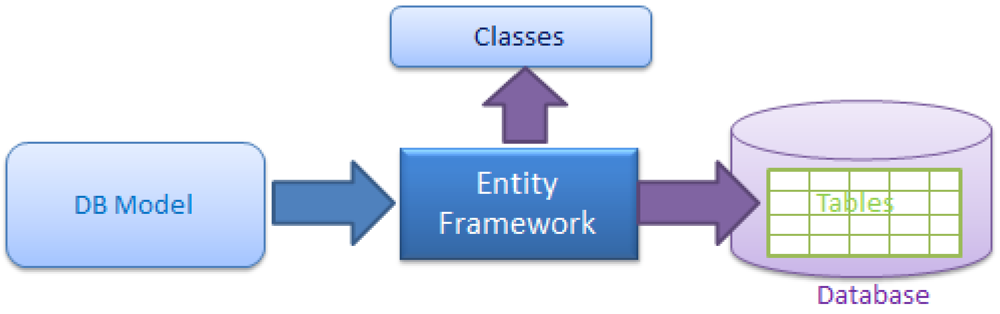
EF: development approaches



Generate Data Access Classes for Existing Database

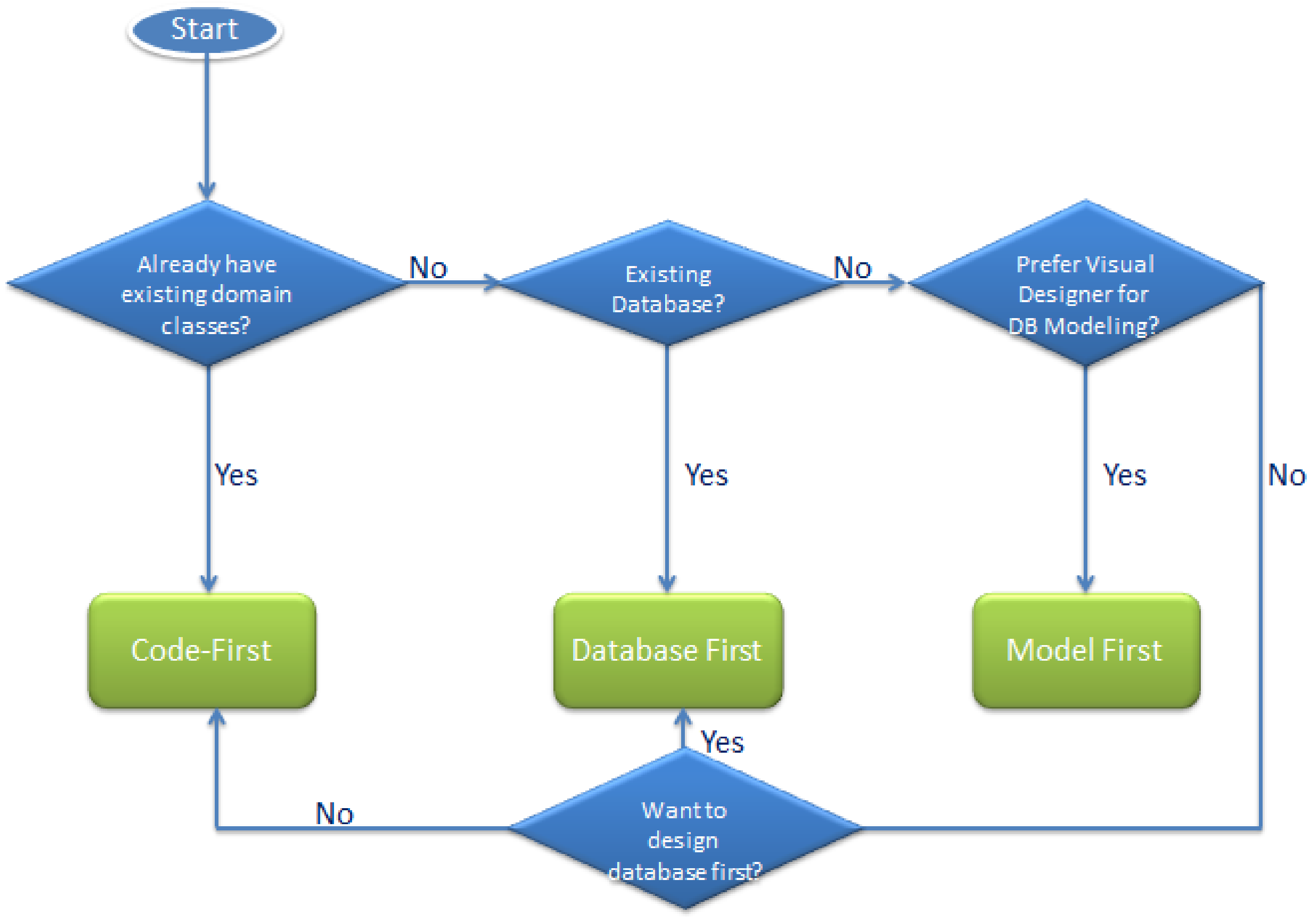


Create Database from the Domain Classes



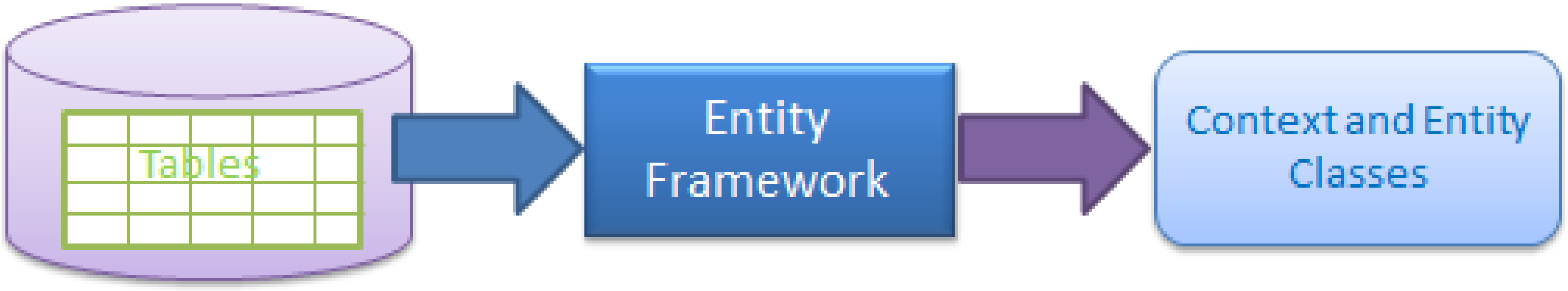
Create Database and Classes from the DB Model design

Choose Dev Approach with EF


Start
Already have existing domain classes?
No
Existing Database?
No
Prefer Visual Designer for DB Modeling?
Yes
Yes
Yes
No
Code-First
Database First
Model First
Yes
No
Want to design database first?

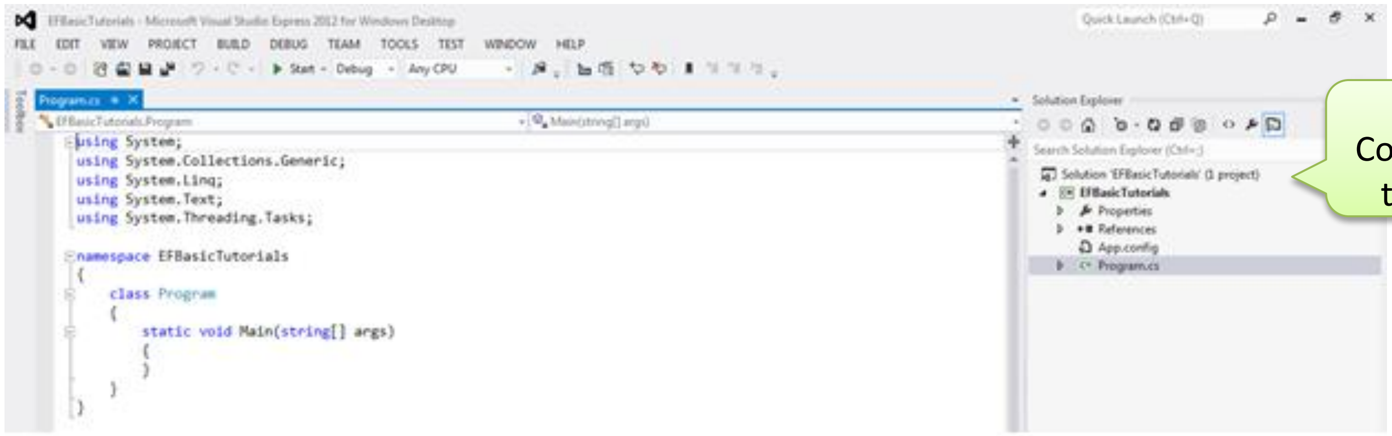
Database-First development

- Context and entity classes generated from the existing database



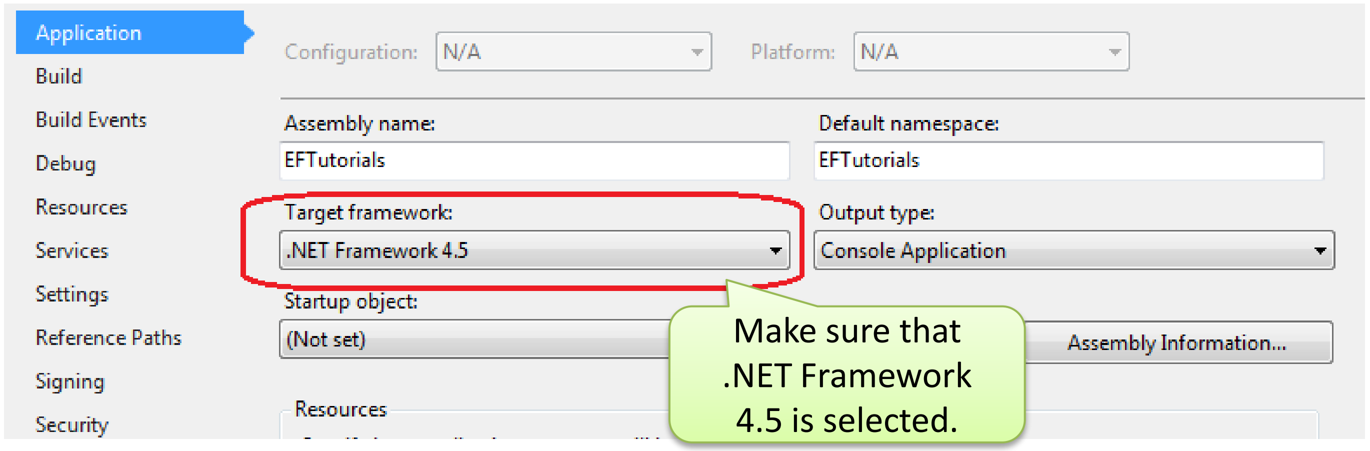
Database-First Approach

Create entity data model from DB (1)



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EFBasicTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

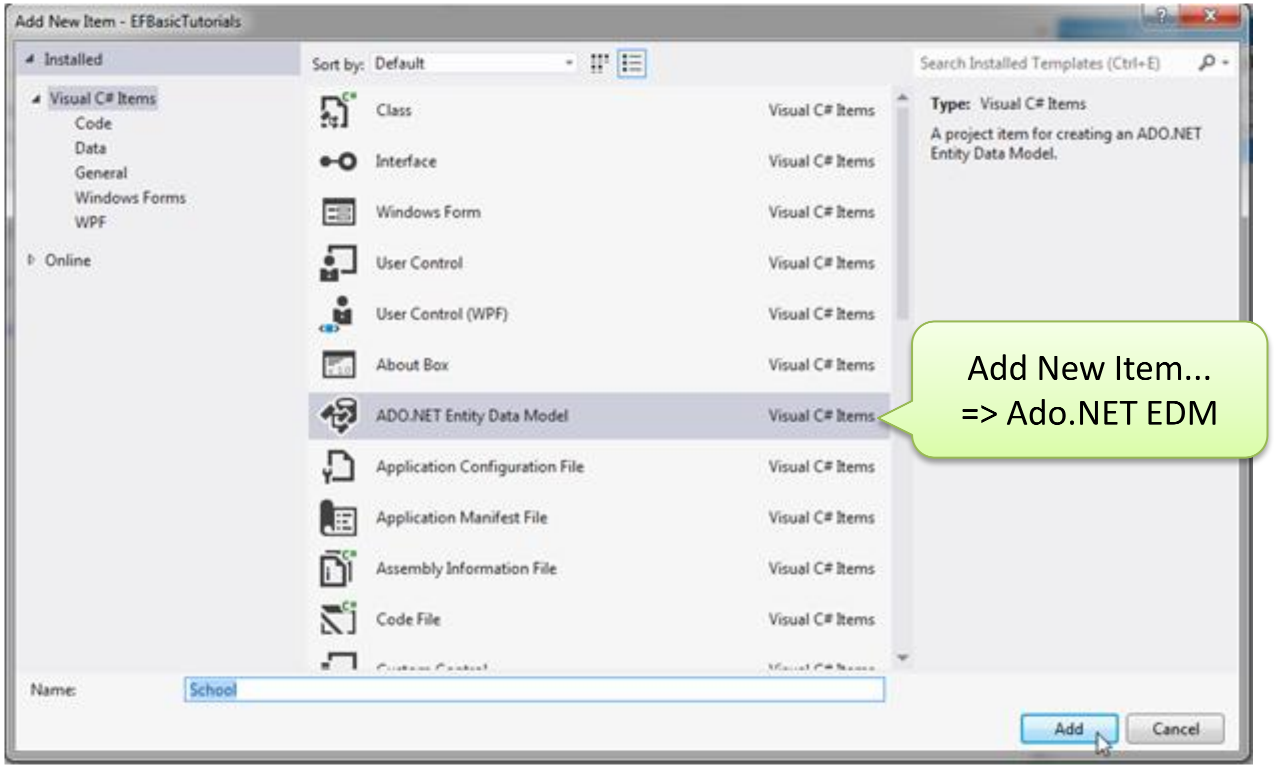
Create a new Console Project, for testing purpose.



Configuration: N/A
Platform: N/A
Assembly name: EFTutorials
Default namespace: EFTutorials
Target framework: .NET Framework 4.5
Output type: Console Application
Startup object: (Not set)
Resources

Make sure that .NET Framework 4.5 is selected.

Create entity data model from DB (2)


Add New Item - EFBasicTutorials
Installed
Visual C# Items
Code
Data
General
Windows Forms
WPF
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class
Interface
Windows Form
User Control
User Control (WPF)
About Box
ADO.NET Entity Data Model
Application Configuration File
Application Manifest File
Assembly Information File
Code File
Add
Cancel

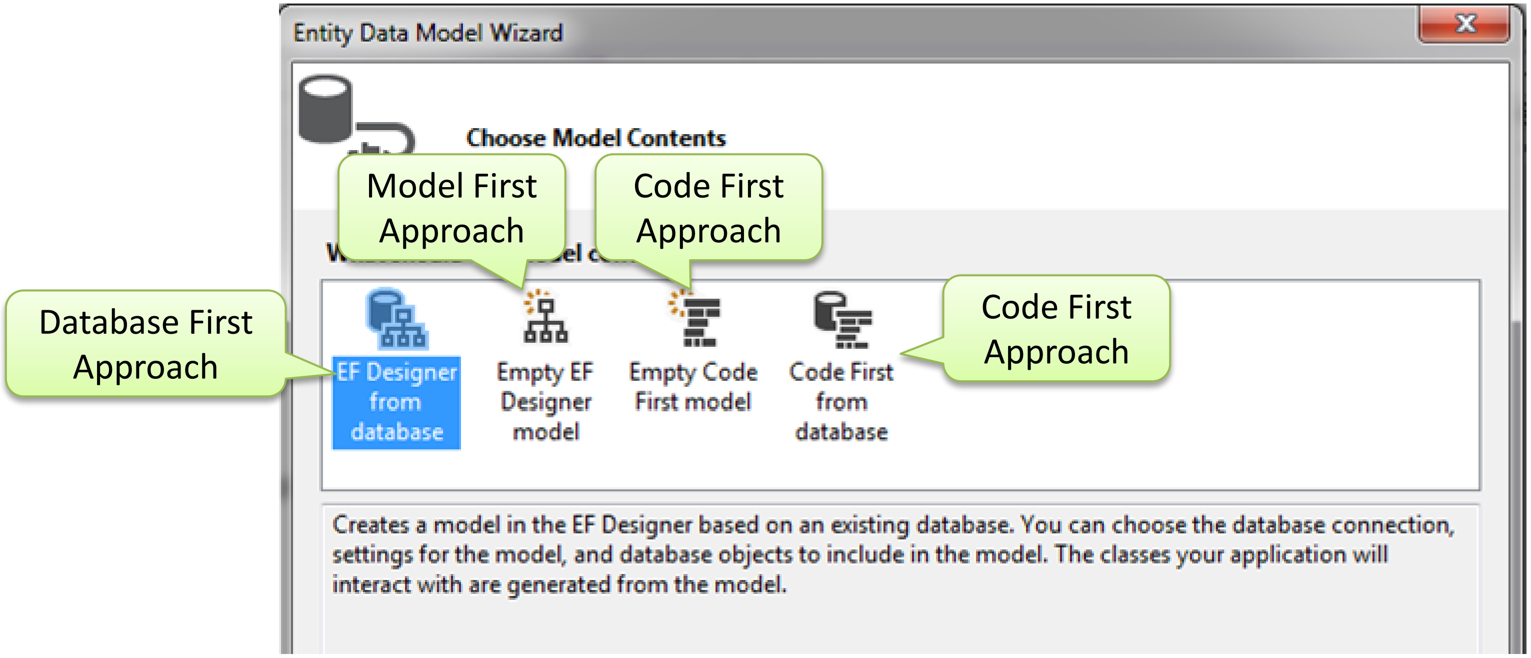
Type: Visual C# Items

A project item for creating an ADO.NET Entity Data Model.

Name: School

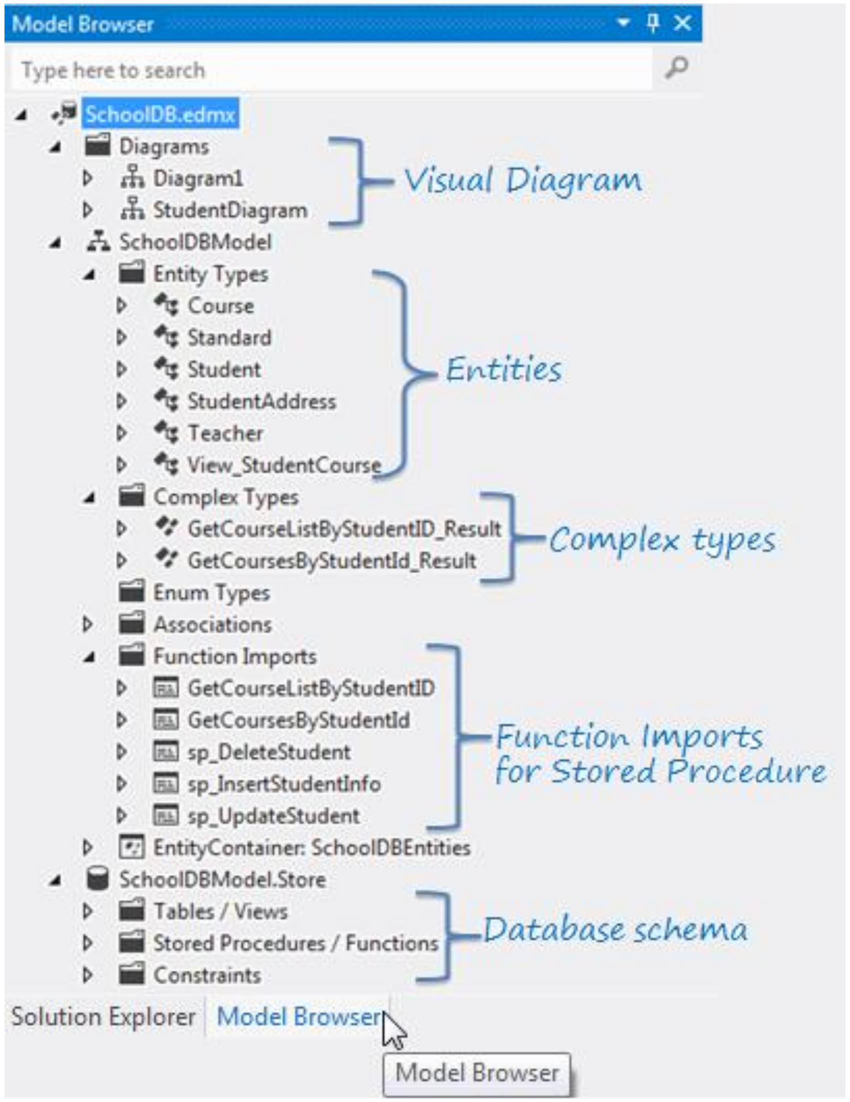
Add New Item...
=> Ado.NET EDM

Create entity data model from DB (3)



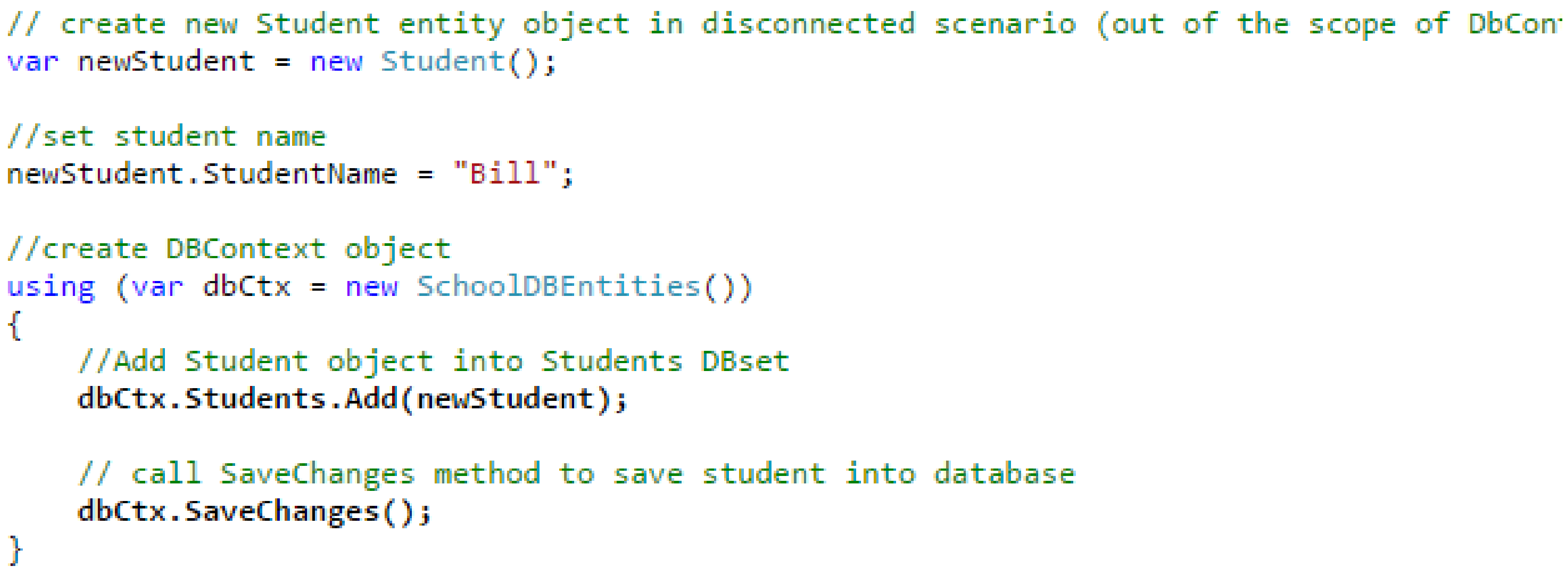
Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.

Model browser



```
Model Browser
Type here to search
◢ SchoolDB.edmx
  ◢ Diagrams
    ▷ Diagram1
    ▷ StudentDiagram
  ◢ SchoolDBModel
    ◢ Entity Types
      ▷ Course
      ▷ Standard
      ▷ Student
      ▷ StudentAddress
      ▷ Teacher
      ▷ View_StudentCourse
    ◢ Complex Types
      ▷ GetCourseListByStudentID_Result
      ▷ GetCoursesByStudentId_Result
      Enum Types
    ▷ Associations
    ◢ Function Imports
      ▷ GetCourseListByStudentID
      ▷ GetCoursesByStudentId
      ▷ sp_DeleteStudent
      ▷ sp_InsertStudentInfo
      ▷ sp_UpdateStudent
    ▷ EntityContainer: SchoolDBEntities
  ◢ SchoolDBModel.Store
    ▷ Tables / Views
    ▷ Stored Procedures / Functions
    ▷ Constraints
Solution Explorer | Model Browser
Model Browser
```

Add single entity



```
// create new Student entity object in disconnected scenario (out of the scope of DbCon
var newStudent = new Student();

//set student name
newStudent.StudentName = "Bill";

//create DBContext object
using (var dbCtx = new SchoolDBEntities())
{
    //Add Student object into Students DBset
    dbCtx.Students.Add(newStudent);

    // call SaveChanges method to save student into database
    dbCtx.SaveChanges();
}
```

Update single entity

```
Student stud;
//1. Get student from DB
using (var ctx = new SchoolDBEntities())
{
    stud = ctx.Students.Where(s => s.StudentName == "New Student1").FirstOrDefault<Student>();
}

//2. change student name in disconnected mode (out of ctx scope)
if (stud != null)
{
    stud.StudentName = "Updated Student1";
}

//save modified entity using new Context
using (var dbCtx = new SchoolDBEntities())
{
    //3. Mark entity as modified
    dbCtx.Entry(stud).State = System.Data.Entity.EntityState.Modified;

    //4. call SaveChanges
    dbCtx.SaveChanges();
}
```

Delete single entity

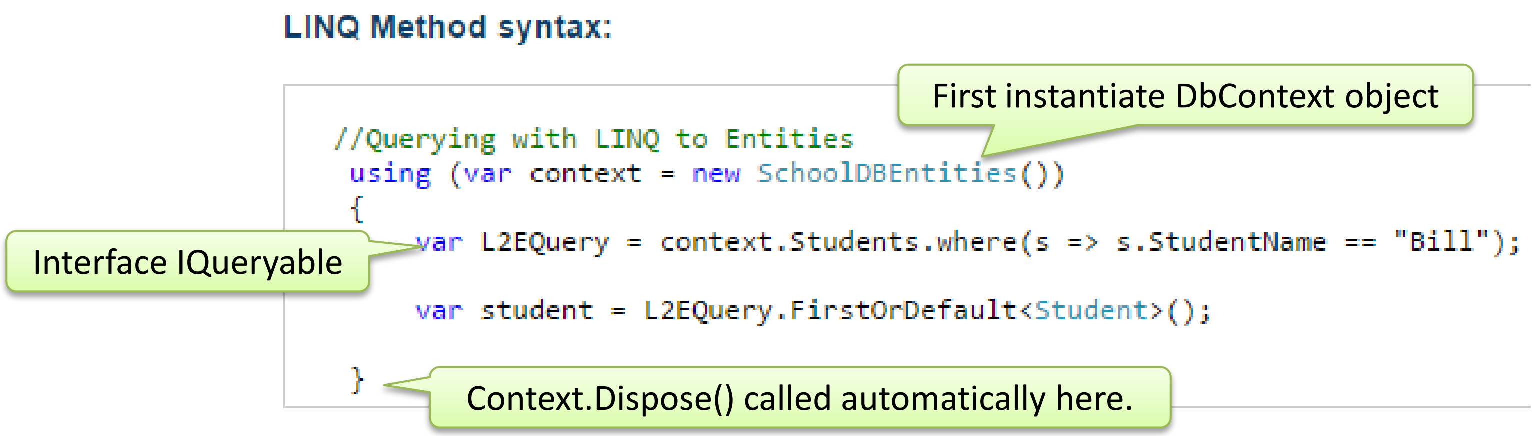
```
Student studentToDelete;
//1. Get student from DB
using (var ctx = new SchoolDBEntities())
{  studentToDelete = ctx.Students.Where(s => s.StudentName == "Student1")
                                 .FirstOrDefault<Student>();
}

//Create new context for disconnected scenario
using (var newContext = new SchoolDBEntities())
{
    newContext.Entry(studentToDelete).State = System.Data.Entity.EntityState.Deleted;
    newContext.SaveChanges();
}
```

Object state Deleted forces DbContext to issue DELETE request.

EF: Querying

- EF supports three types of queries:
  - LINQ to Entities
    - Language Integrated Query Language
    - Method syntax , query syntax
  - Entity SQL
    - Processed by EF Object Services
    - Raw SQL, ObjectContext, ObjectQuery
  - Native SQL
    - Similar to Entity SQL
    - query executed directly on a EF Entity

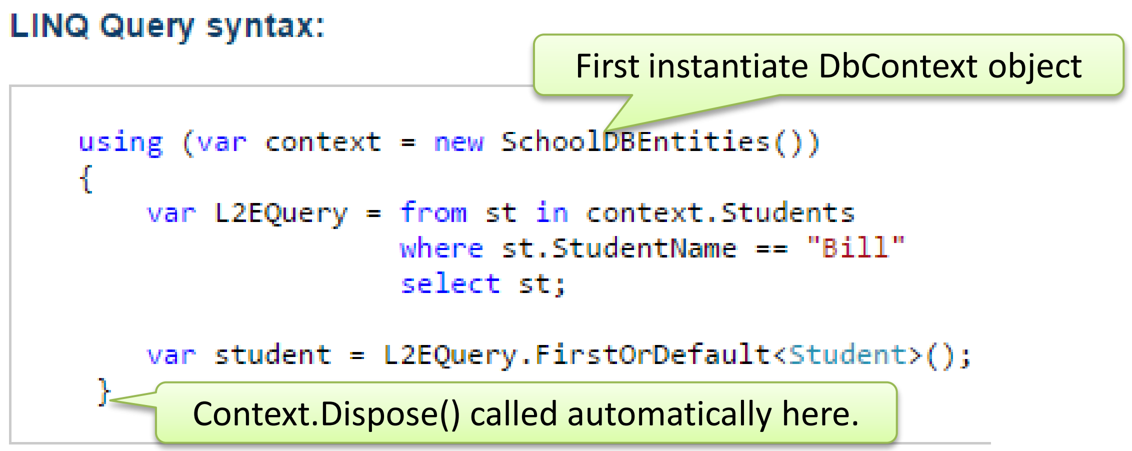
LINQ to Entities



LINQ Method syntax:

```
//Querying with LINQ to Entities
 using (var context = new SchoolDBEntities())
 {
     var L2EQuery = context.Students.where(s => s.StudentName == "Bill");

     var student = L2EQuery.FirstOrDefault<Student>();

 }
```

First instantiate DbContext object
Interface IQueryable
Context.Dispose() called automatically here.



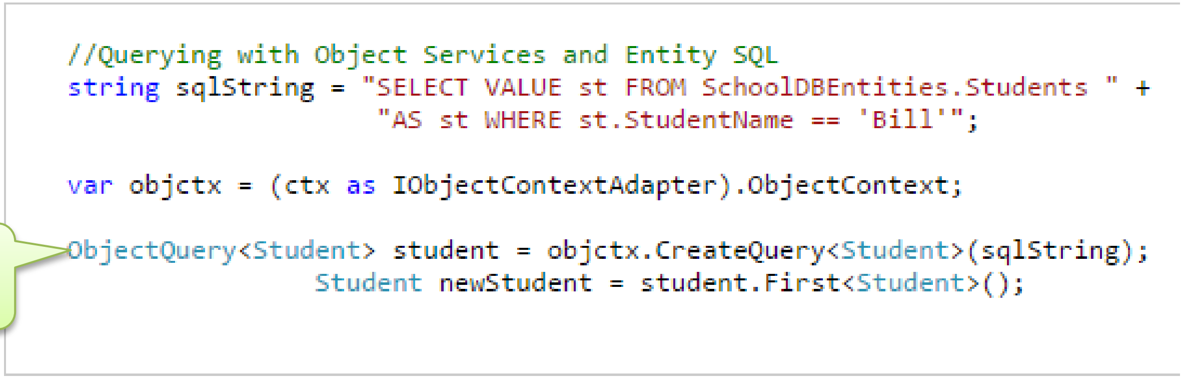
LINQ Query syntax:

```
using (var context = new SchoolDBEntities())
{
    var L2EQuery = from st in context.Students
                   where st.StudentName == "Bill"
                   select st;

    var student = L2EQuery.FirstOrDefault<Student>();
 }
```

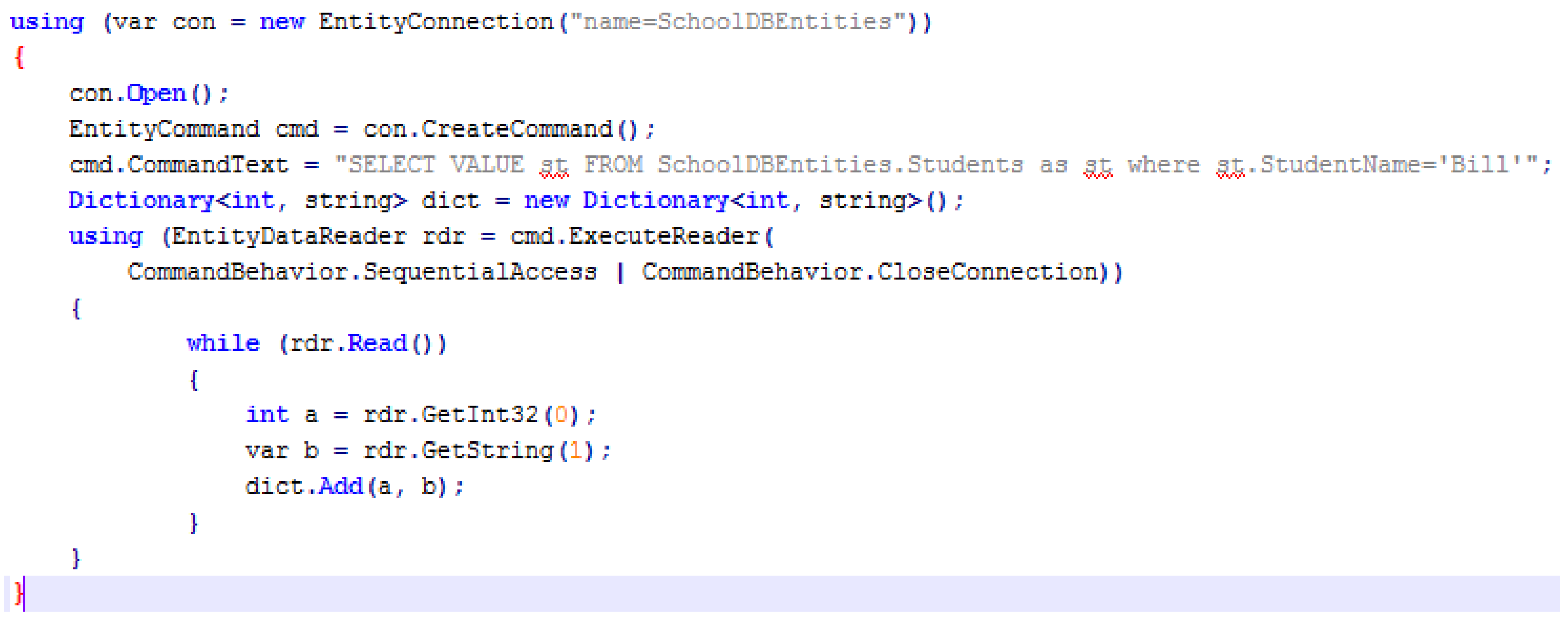
First instantiate DbContext object
Context.Dispose() called automatically here.

Entity SQL

ObjectQuery instead of IQueryable!



```
//Querying with Object Services and Entity SQL
string sqlString = "SELECT VALUE st FROM SchoolDBEntities.Students " +
                    "AS st WHERE st.StudentName == 'Bill'";

var objctx = (ctx as IObjectContextAdapter).ObjectContext;

ObjectQuery<Student> student = objctx.CreateQuery<Student>(sqlString);
                Student newStudent = student.First<Student>();
```



```
using (var con = new EntityConnection("name=SchoolDBEntities"))
{
    con.Open();
    EntityCommand cmd = con.CreateCommand();
    cmd.CommandText = "SELECT VALUE st FROM SchoolDBEntities.Students as st where st.StudentName='Bill'";
    Dictionary<int, string> dict = new Dictionary<int, string>();
    using (EntityDataReader rdr = cmd.ExecuteReader(
        CommandBehavior.SequentialAccess | CommandBehavior.CloseConnection))
    {
            while (rdr.Read())
            {
                int a = rdr.GetInt32(0);
                var b = rdr.GetString(1);
                dict.Add(a, b);
            }
    }
}
```

Native/Raw SQL

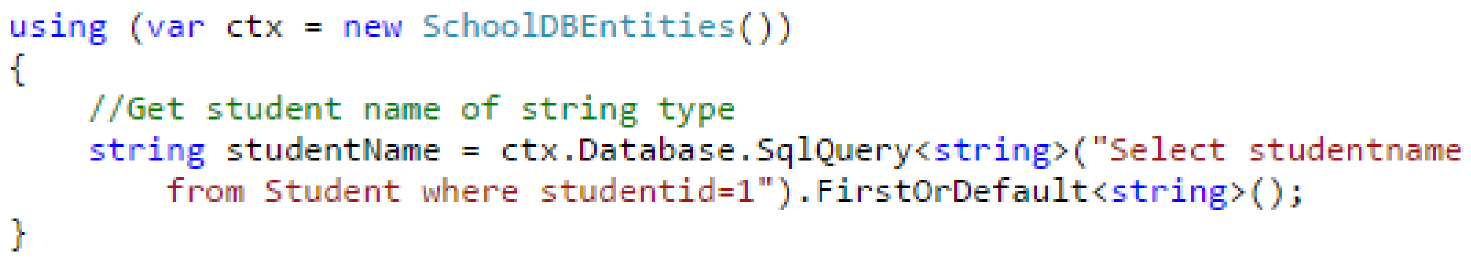
- SQL query for entity types which returns particular types of entities
- SQL query for non-entity types which returns a primitive data type
- Raw SQL commands to the database

Raw SQL – Entity types

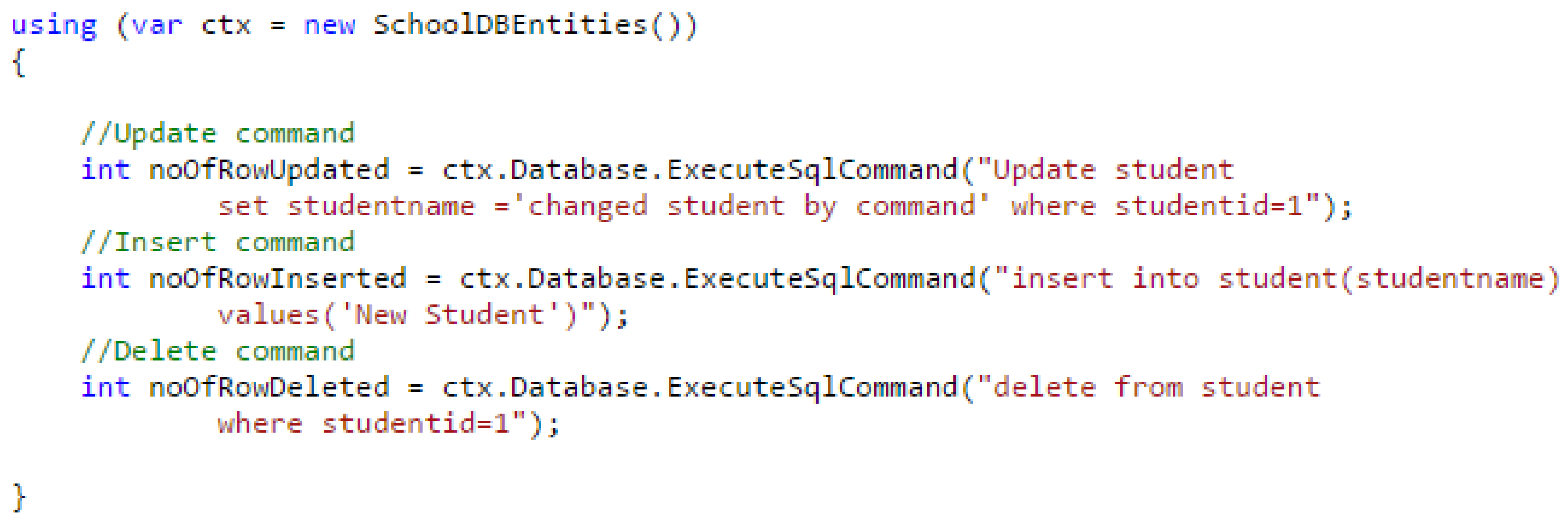
```
using (var ctx = new SchoolDBEntities())
{
    var studentList = ctx.Students.SqlQuery("Select * from Student").ToList<Student>();

}


using (var ctx = new  SchoolDBEntities())
{
    var studentName = ctx.Students.SqlQuery("Select studentid, studentname
        from Student where studentname='New Student1'").ToList();

}


using (var ctx = new SchoolDBEntities())
{
    //this will throw an exception
    var studentName = ctx.Students.SqlQuery("Select studentid as id, studentname as name
            from Student where studentname='New Student1'").ToList();
}
```

Should match DbSet property name. Otherwise, Exception!

Raw SQL: commands and non-entity types



```
using (var ctx = new SchoolDBEntities())
{
    //Get student name of string type
    string studentName = ctx.Database.SqlQuery<string>("Select studentname
        from Student where studentid=1").FirstOrDefault<string>();
}
```



```
using (var ctx = new SchoolDBEntities())
{

    //Update command
    int noOfRowUpdated = ctx.Database.ExecuteSqlCommand("Update student
            set studentname ='changed student by command' where studentid=1");
    //Insert command
    int noOfRowInserted = ctx.Database.ExecuteSqlCommand("insert into student(studentname)
            values('New Student')");
    //Delete command
    int noOfRowDeleted = ctx.Database.ExecuteSqlCommand("delete from student
            where studentid=1");

}
```

Projection L2E Queries

- First/FirstOrDefault
- Single/SingleOrDefault
- ToList
- GroupBy
- OrderBy
- Annonymous class result
- Nested queries

First / FirstOrDefault

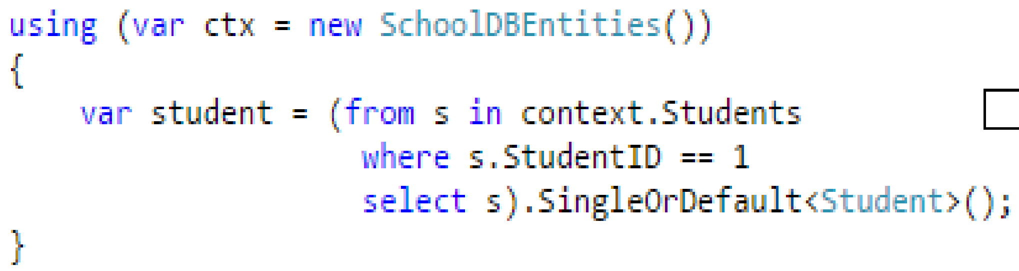
- Returns first entity from the result
- When the result does not contain any object:
  - First throws exception
  - FirstOrDefault returns null

```
using (var ctx = new SchoolDBEntities())
{
    var student = (from s in ctx.Students
                where s.StudentName == "Student1"
                select s).FirstOrDefault<Student>();
 }
```

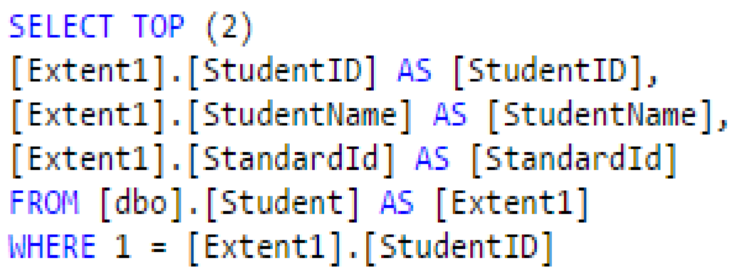
```
SELECT TOP (1)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
WHERE 'Student1' = [Extent1].[StudentName]
```

Single/SingleOrDefault

- Use it when a query is expected to return only one object.
- Exception thrown if there is more than one!
- If no object => Single throws exception!
- If no object => SingleOrDefault returns null.



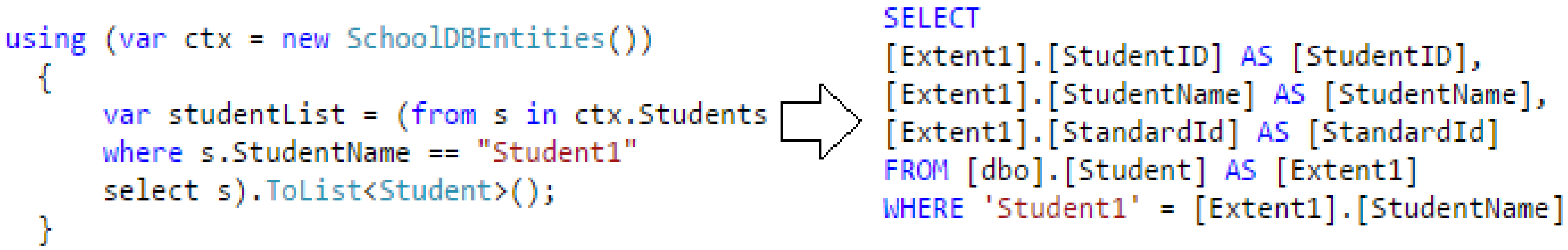
```
using (var ctx = new SchoolDBEntities())
{
    var student = (from s in context.Students
                   where s.StudentID == 1
                   select s).SingleOrDefault<Student>();
}
```



```
SELECT TOP (2)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
WHERE 1 = [Extent1].[StudentID]
```

ToList

- Retrieve all objects that satisfy certain condition.



```
using (var ctx = new SchoolDBEntities())
  {
      var studentList = (from s in ctx.Students
      where s.StudentName == "Student1"
      select s).ToList<Student>();
  }
```

```
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]
WHERE 'Student1' = [Extent1].[StudentName]
```

OrderBy

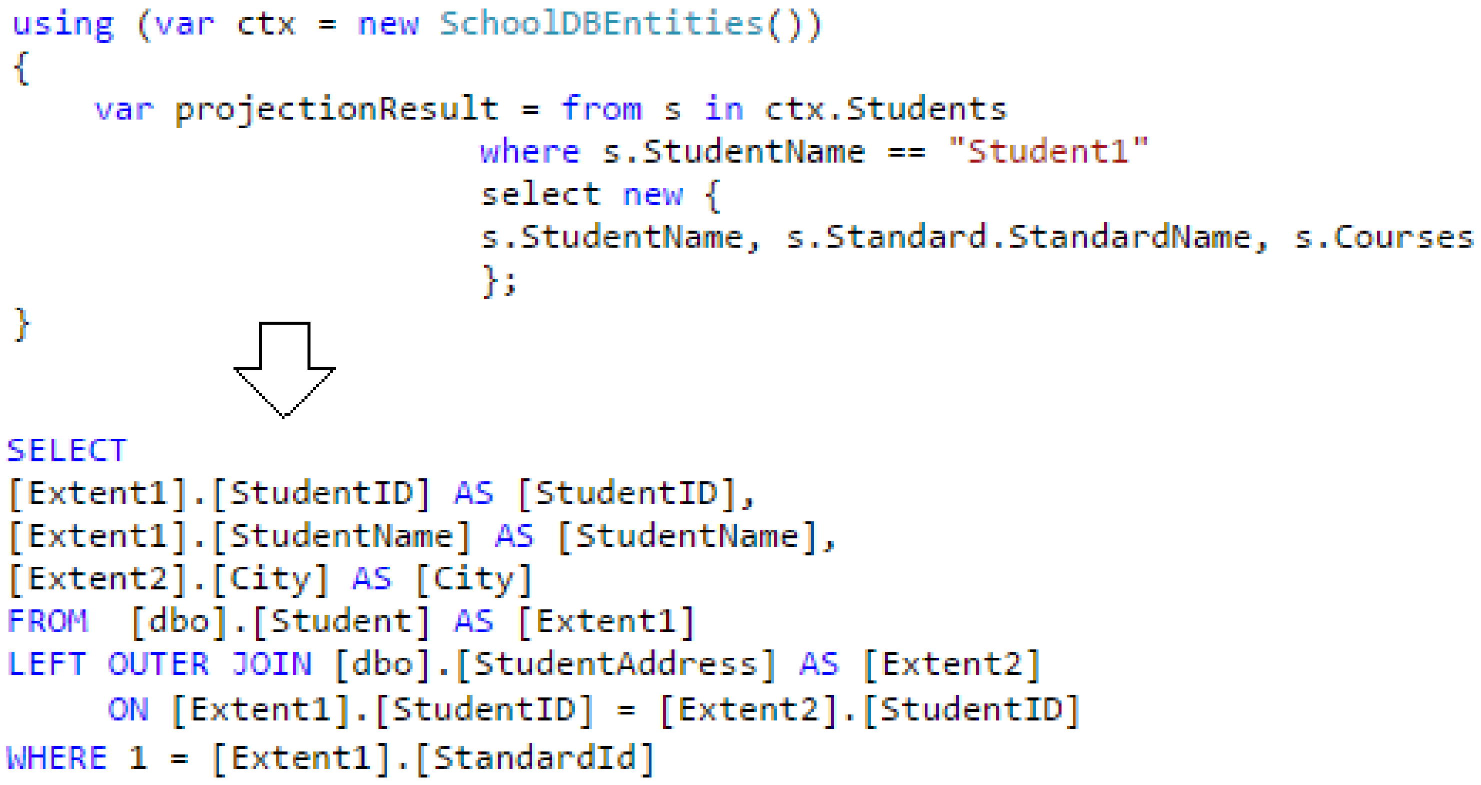
- Sort objects in result by certain criteria

```
using (var ctx = new SchoolDBEntities())
{
    var student1 = from s in ctx.Students
                   orderby s.StudentName ascending
                   select s;
}
```

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

Anonnymous class result

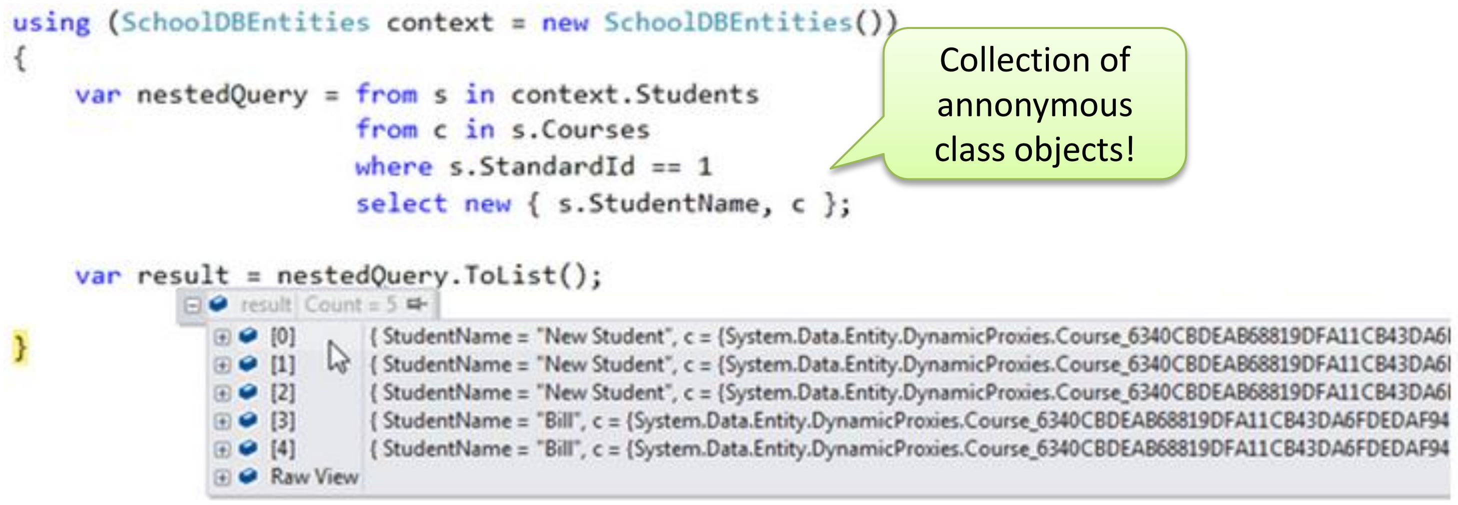
- Return combined result without creating a special entity class



```
using (var ctx = new SchoolDBEntities())
{
    var projectionResult = from s in ctx.Students
                           where s.StudentName == "Student1"
                           select new {
                           s.StudentName, s.Standard.StandardName, s.Courses
                           };
}
```

```
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent2].[City] AS [City]
FROM  [dbo].[Student] AS [Extent1]
LEFT OUTER JOIN [dbo].[StudentAddress] AS [Extent2]
     ON [Extent1].[StudentID] = [Extent2].[StudentID]
WHERE 1 = [Extent1].[StandardId]
```

Nested classes



```
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();
}
```

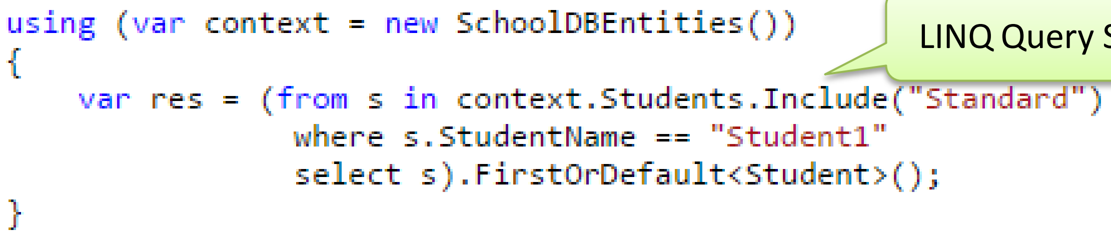
```
result  Count = 5
  [0]   { StudentName = "New Student", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6
  [1]   { StudentName = "New Student", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6
  [2]   { StudentName = "New Student", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6
  [3]   { StudentName = "Bill", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94
  [4]   { StudentName = "Bill", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94
  Raw View
```

Collection of annonymous class objects!

```
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]
```

Eager loading

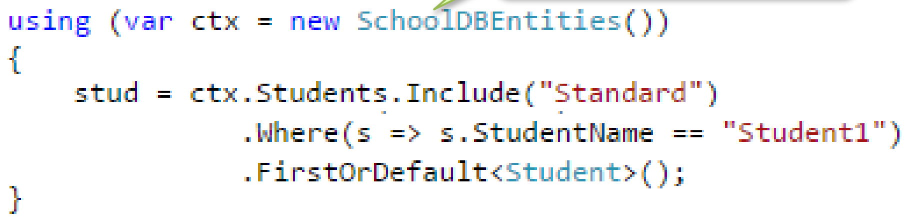
- Query returns requested objects and their related/linked (over assoc. end) objects, too.
- Achieved using Include method of IQueryable



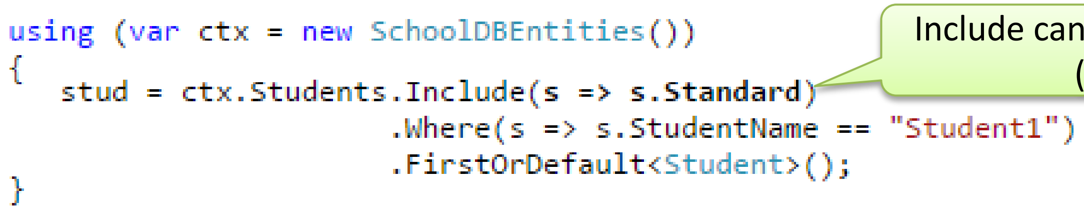
```
using (var context = new SchoolDBEntities())
{
    var res = (from s in context.Students.Include("Standard")
               where s.StudentName == "Student1"
               select s).FirstOrDefault<Student>();
}
```

LINQ Query Syntax

LINQ Method Syntax



```
using (var ctx = new SchoolDBEntities())
{
    stud = ctx.Students.Include("Standard")
                .Where(s => s.StudentName == "Student1")
                .FirstOrDefault<Student>();
}
```

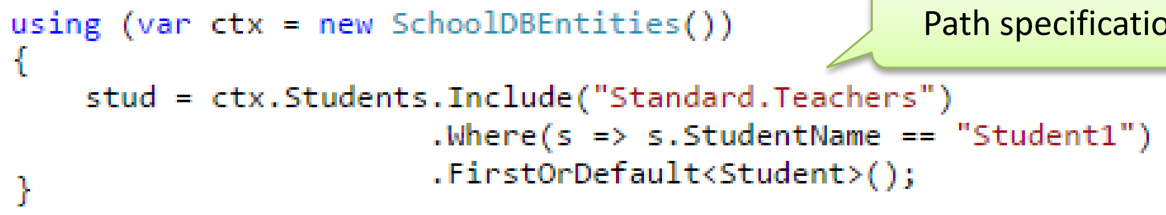


```
using (var ctx = new SchoolDBEntities())
{
   stud = ctx.Students.Include(s => s.Standard)
                   .Where(s => s.StudentName == "Student1")
                   .FirstOrDefault<Student>();
}
```

Include can accept a lambda expression (instead of string)!

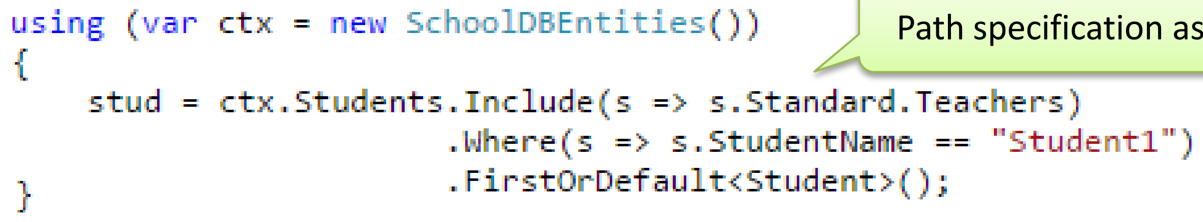
Eager loading – multiple levels

- Load object across several link hops
- Load a student with name “Student 1” and its related standard and teachers
  (three level of objects)



```
using (var ctx = new SchoolDBEntities())
{
    stud = ctx.Students.Include("Standard.Teachers")
                        .Where(s => s.StudentName == "Student1")
                        .FirstOrDefault<Student>();
}
```

Path specification as string.



```
using (var ctx = new SchoolDBEntities())
{
    stud = ctx.Students.Include(s => s.Standard.Teachers)
                        .Where(s => s.StudentName == "Student1")
                        .FirstOrDefault<Student>();
}
```

Path specification as lambda.

Lazy loading

- Delayed loading of related objects/data

```
using (var ctx = new SchoolDBEntities())
{
    //Loading students only
    IList<Student> studList = ctx.Students.ToList<Student>();

    Student std = studList[0];

   //Loads Student address for particular Student only (seperate SQL query)
   StudentAddress add = std.StudentAddress;
}
```

Lazy loading occurs at this line.
Property StudentAddress must be defined as public virtual!

```
public partial class SchoolDBEntities : DbContext
{   public SchoolDBEntities(): base("name=SchoolDBEntities")
    {  this.Configuration.LazyLoadingEnabled = false;   }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {  throw new UnintentionalCodeFirstException();   }
}
```

Disables lazy loading.

Lazy loading - rules

- *context.Configuration.ProxyCreationEnabled* should be true.
- *context.Configuration.LazyLoadingEnabled* should be true.
- Navigation property should be defined as public, virtual.
  - Context will **NOT** do lazy loading if the property is not define as virtual.

Explicit loading – Load()

```
using (var context = new SchoolDBEntities())
{
    //Disable Lazy loading
    context.Configuration.LazyLoadingEnabled = false;

    var student = (from s in context.Students
                        where s.StudentName == "Bill"
                        select s).FirstOrDefault<Student>();

    context.Entry(student).Reference(s => s.Courses).Load();
}
```

Method Reference used for loading a single object.

```
using (var context = new SchoolDBEntities())
{
    context.Configuration.LazyLoadingEnabled = false;

    var student = (from s in context.Students
                        where s.StudentName == "Bill"
                        select s).FirstOrDefault<Student>();

    context.Entry(student).Collection(s => s.Courses).Load();
}
```

Method Collection used for loading a collection of linked objects.

References

- The official EF portal: https://msdn.microsoft.com/en-us/data/ef.aspx
- EF tutorial: http://www.entityframeworktutorial.net
- Entity Framework 6 Recipes, paperback http://www.amazon.com/Entity-Framework-Recipes-Brian-Driscoll/dp/1430257881