

SERIES VỀ ENTITY FRAMEWORK

Phần 1 – Giới thiệu

I. Giới thiệu về Entity Framework

Bạn sẽ học được những khái niệm cơ bản của Entity Framework sau khi hoàn thành những bài hướng dẫn trong mục này. Chúng ta sẽ sử dụng Entity Framework 6.0 và Visual Studio 2012 cho tất cả các bài hướng dẫn.

Bảng sau liệt kê danh sách tất cả các phiên bản quan trọng của Entity Framework.

Phiên bản EF và những chức năng được giới thiệu

- **EF 3.5:** Hỗ trợ O/RM cơ bản với phương pháp tiếp cận Database First.
- **EF 4.0:** Hỗ trợ POCO, Lazy loading, cải thiện khả năng kiểm tra, tùy biến mã và phương pháp tiếp cận Model First.
- **EF 4.1:** Lần đầu xuất hiện trên NuGet, đơn giản hóa DbContext API quaObjectContext, phương pháp tiếp cận Code First. Ra mắt gói EF 4.1.1 để vá các lỗi của phiên bản 4.1.
- **EF 4.3:** Chức năng Code First Migrations cho phép một CSDL được tạo bởi Code First để gia tăng thay đổi như sự tiến hóa mô hình Code First của bạn. Ra mắt gói EF 4.3.1 để vá các lỗi của phiên bản EF 4.3.
- **EF 5.0:** Công bố EF là mã nguồn mở. Giới thiệu việc hỗ trợ Enum, các hàm table-valued, kiểu dữ liệu không gian, multiple-diagrams per model, màu sắc của các hình trên bề mặt thiết kế và nhập một nhóm của stored procedures, EF Power Tools và các cải thiện về hiệu năng khác.
- **EF 6.0 – Current release:** EF 6.0/6.1 là bản phát hành mới nhất của Entity Framework. Nó bao gồm nhiều chức năng mới liên quan tới Code First và thiết kế EF giống như truy vấn và lưu bất đồng bộ, kết nối Resiliency, giải pháp dependency, ...

Những từ viết tắt:

- O/RM (Object-relational mapping)
- POCO (Plain Old CLR Object)
- EF (Entity Framework)
- II. Entity Framework là gì?

Viết và quản lý mã ADO .Net cho việc truy cập dữ liệu là một công việc nhàm chán và tẻ nhạt. Microsoft đã cung cấp một O/RM framework gọi là “Entity Framework” để tự động hóa các hoạt động liên quan đến CSDL cho ứng dụng của bạn.

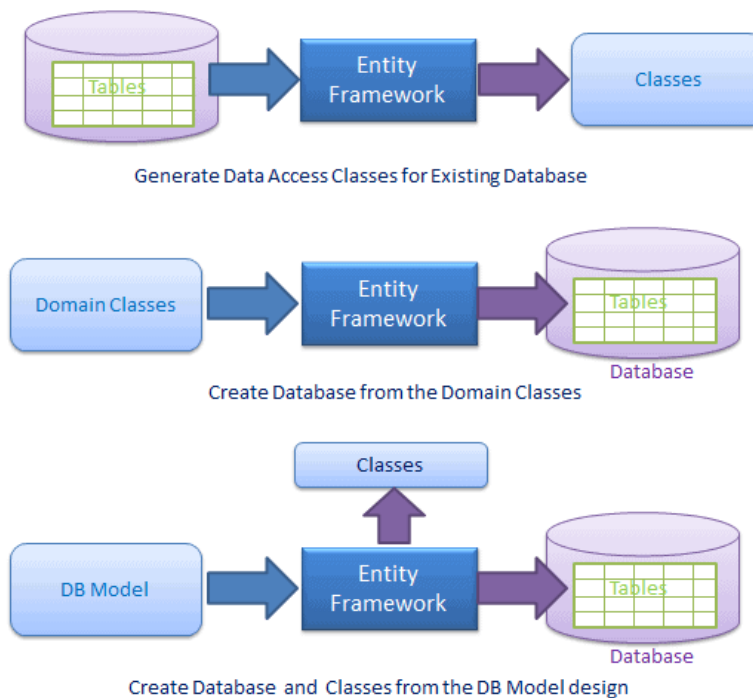
[https://msdn.microsoft.com/en-us/library/aa937723\(v=vs.113\).aspx](https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx)

Entity framework là một Object/Relational Mapping (O/RM) framework. Nó là một sự cải tiến tới ADO.NET và đưa cho các lập trình viên một cơ chế tự động cho việc truy xuất và sắp xếp dữ liệu trong CSDL.

Entity framework là rất hữu ích trong ba tình huống. Đầu tiên là nếu bạn đã có một CSDL hoặc muốn thiết kế CSDL trước khi làm các phần khác của ứng dụng. Thứ hai là bạn muốn tập trung vào các

domain class rồi mới tạo CSDL từ các domain class đó. Thứ ba là bạn muốn thiết kế schema của CSDL trên visual designer rồi mới tạo CSDL và các class.

Hình dưới đây minh họa các tình huống trên:



Như hình trên thì EF tạo các lớp truy cập dữ liệu cho CSDL hiện tại của bạn để bạn có thể sử dụng các lớp này để tương tác với CSDL thay vì trực tiếp bằng ADO.Net.

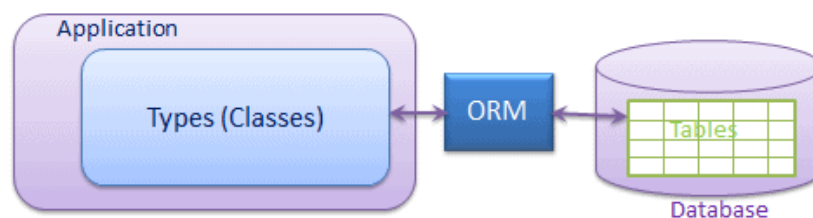
EF có thể cũng tạo CSDL từ các domain class của bạn vì thế bạn có thể tập trung vào thiết kế domain-driven.

EF cung cấp cho bạn một model designer mà ở đó bạn có thể thiết kế DB model của bạn rồi EF sẽ tạo CSDL và các lớp dựa trên DB model.

III. O/RM là gì?

ORM là một công cụ dành cho việc lưu trữ dữ liệu từ domain objects tới CSDL quan hệ như MS SQL theo một cách tự động hóa và không phải lập trình nhiều. ORM bao gồm ba phần chính: những đối tượng Domain class, những đối tượng CSDL quan hệ và thông tin Mapping về cách làm thế nào domain objects nối với những đối tượng CSDL quan hệ (tables, views & stored procedures). ORM cho phép chúng ta giữ thiết kế CSDL tách biệt với thiết kế domain class. Điều này giúp cho việc bảo trì và nâng cấp ứng dụng. Nó cũng tự động hóa các thao tác CRUD (Create, Read, Update & Delete) giúp cho các lập trình viên không cần phải viết thủ công.

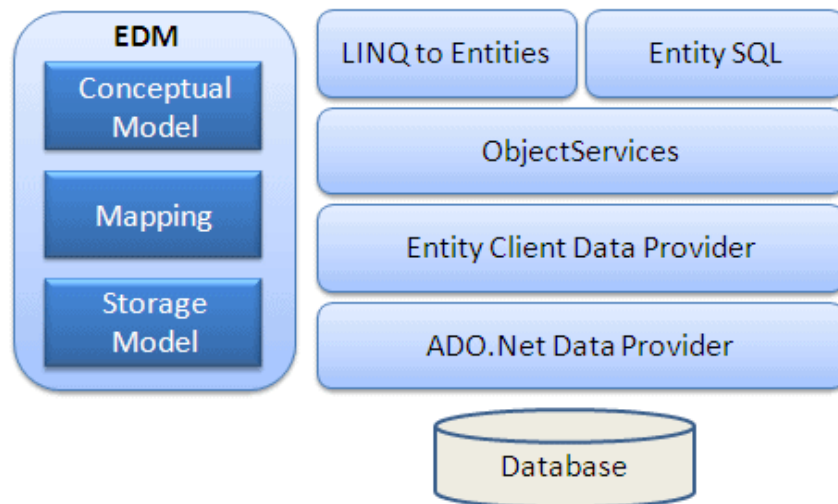
Một công cụ ORM điển hình khởi tạo các lớp để tương tác với CSDL như bên dưới:



Có nhiều ORM frameworks cho .Net trên thị trường như DataObjects.Net, NHibernate, OpenAccess, SubSonic,... Entity Framework là một ORM framework mã nguồn mở từ Microsoft.

IV. Cấu trúc của Entity Framework

Hình sau thể hiện cấu trúc tổng thể của Entity Framework. Cho phép chúng ta nhìn vào từng thành phần riêng biệt của cấu trúc:



EDM (Entity Data Model): EDM gồm ba phần chính – Conceptual model, Mapping và Storage model.

Conceptual Model: chứa các model class và những quan hệ của nó. Phần này sẽ độc lập với thiết kế bảng CSDL của bạn.

Storage Model: là database design model gồm các bảng, views, stored procedures, và những quan hệ của nó và các khóa.

Mapping: gồm có thông tin về cách làm thế nào Conceptual model nối với Storage model.

LINQ to Entities: là một ngôn ngữ truy vấn sử dụng để viết các truy vấn tới object model. Nó trả về các thực thể được định nghĩa trong Conceptual model. Bạn có thể sử dụng kỹ năng LINQ của bạn ở đây.

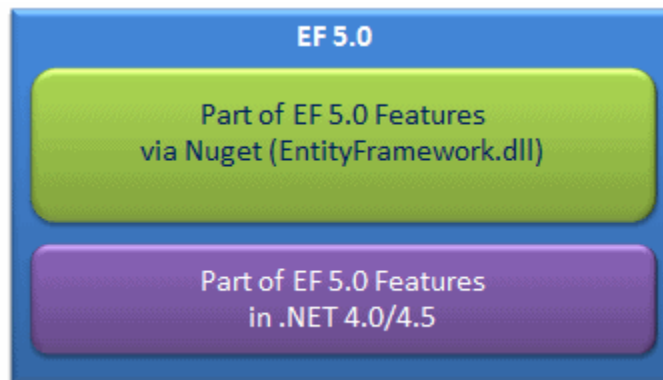
Entity SQL: là một ngôn ngữ truy vấn khác giống LINQ to Entities. Tuy nhiên nó có một chút khó khăn hơn L2E và các lập trình viên sẽ phải học nó riêng.

Object Service: là một điểm vào chính cho việc cho việc truy cập dữ liệu từ CSDL và trả về. Object service có trách nhiệm trong việc cụ thể hóa quá trình chuyển đổi dữ liệu trả về từ một entity client data provider (lớp tiếp theo) tới một entity object structure.

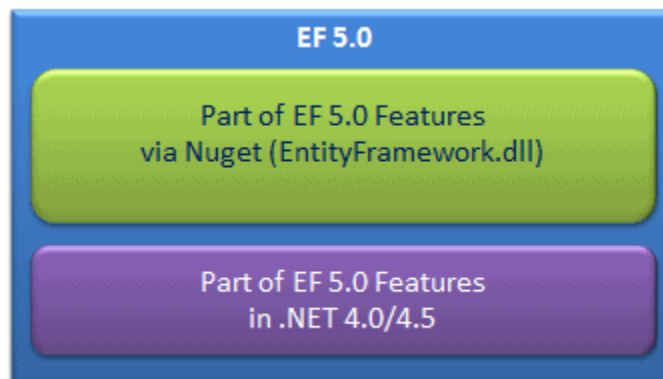
Entity Client Data Provider: Trách nhiệm chính của lớp này là chuyển đổi L2E hoặc những truy vấn Entity SQL vào một truy vấn SQL, nó được hiểu bởi CSDL cơ bản. Nó giao tiếp với ADO.Net data provider lần lượt gửi và nhận dữ liệu từ CSDL.

ADO.Net Data Provider: Lớp này giao tiếp với CSDL bằng việc sử dụng chuẩn ADO.Net.

Phần 2 – Cài đặt môi trường



Entity Framework 5.0 API được phân bố ở hai chỗ: trong gói NuGet và trong .NET framework. .NET framework 4.0/4.5 bao gồm EF core API trong khi EntityFramework.dll qua gói NuGet bao gồm những chức năng cụ thể EF 5.0



Điều này đã thay đổi với EF 6.0, nó bao gồm chỉ trong EntityFramework.dll và không phụ thuộc vào .NET framework.



Dành cho những bài hướng dẫn cơ bản chúng ta sẽ sử dụng EF 6.0 phiên bản mới nhất của Entity Framework.

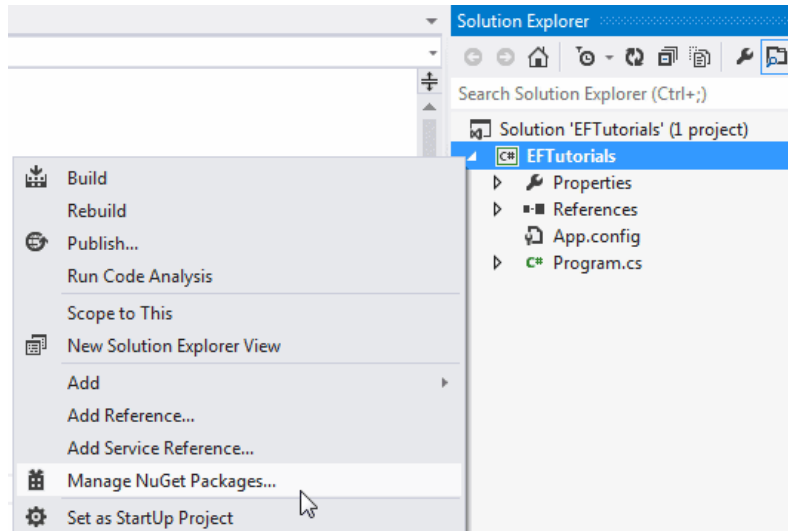
Cài đặt các công cụ sau để làm việc với Entity Framework:

- .NET Framework 4.5
- Visual Studio 2012
- MS SQL Server 2005/2008/2012 Express

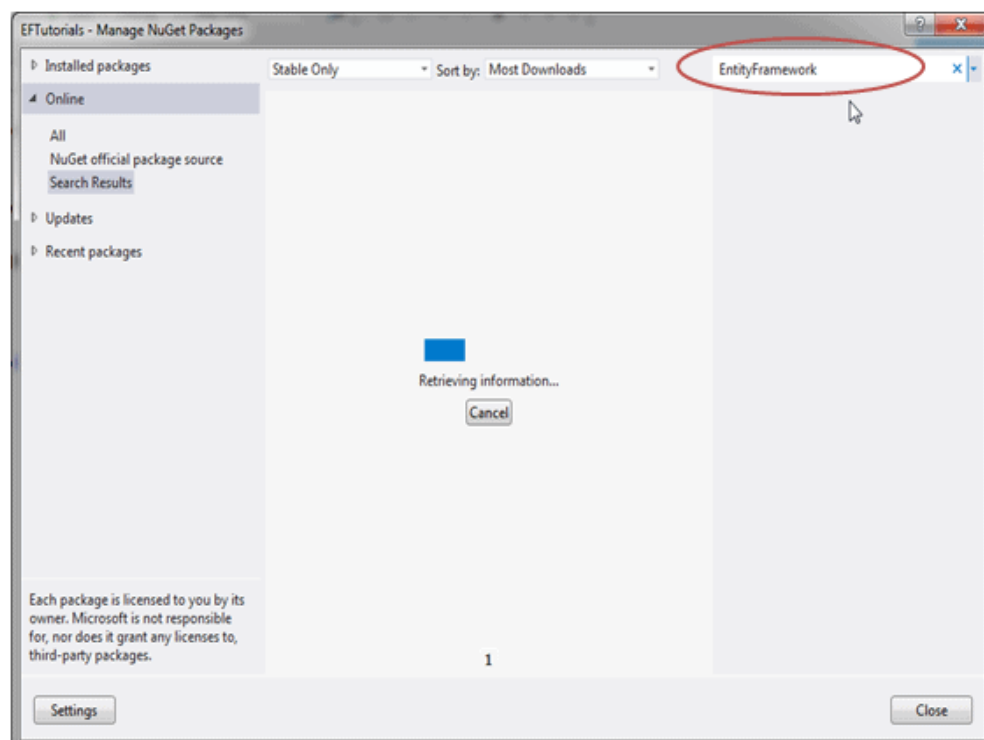
Cài đặt EF qua Nuget:

Bạn có thể cài đặt Entity Framework vào project của bạn qua NuGet. Chúng ta sẽ cài đặt EF (EntityFramework.dll) qua NuGet trong ứng dụng console trong VS 2012 ở đây. Bạn có thể cài đặt EF qua NuGet với cùng một cách đối với bất kỳ phiên bản nào của VS.

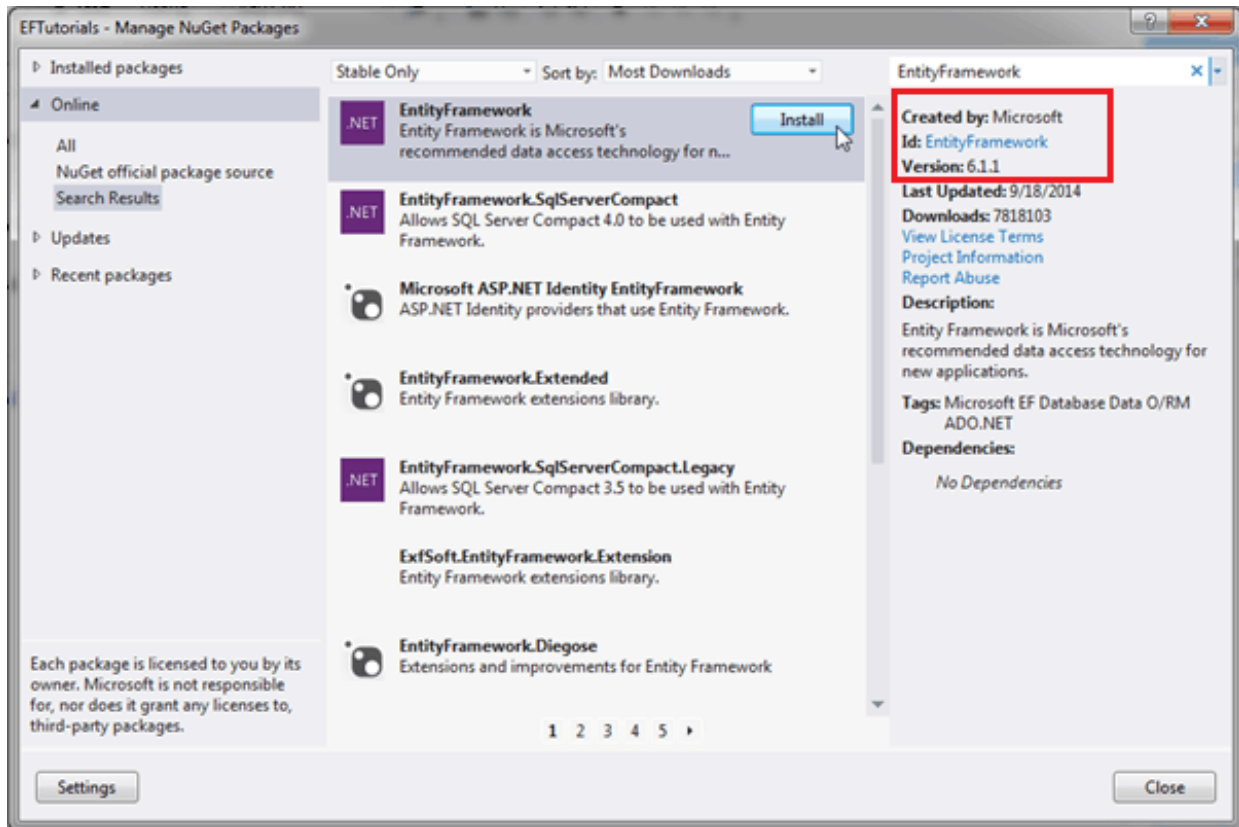
Click chuột phải vào project trong cửa sổ Solution Explorer và chọn Manage NuGet Packages.



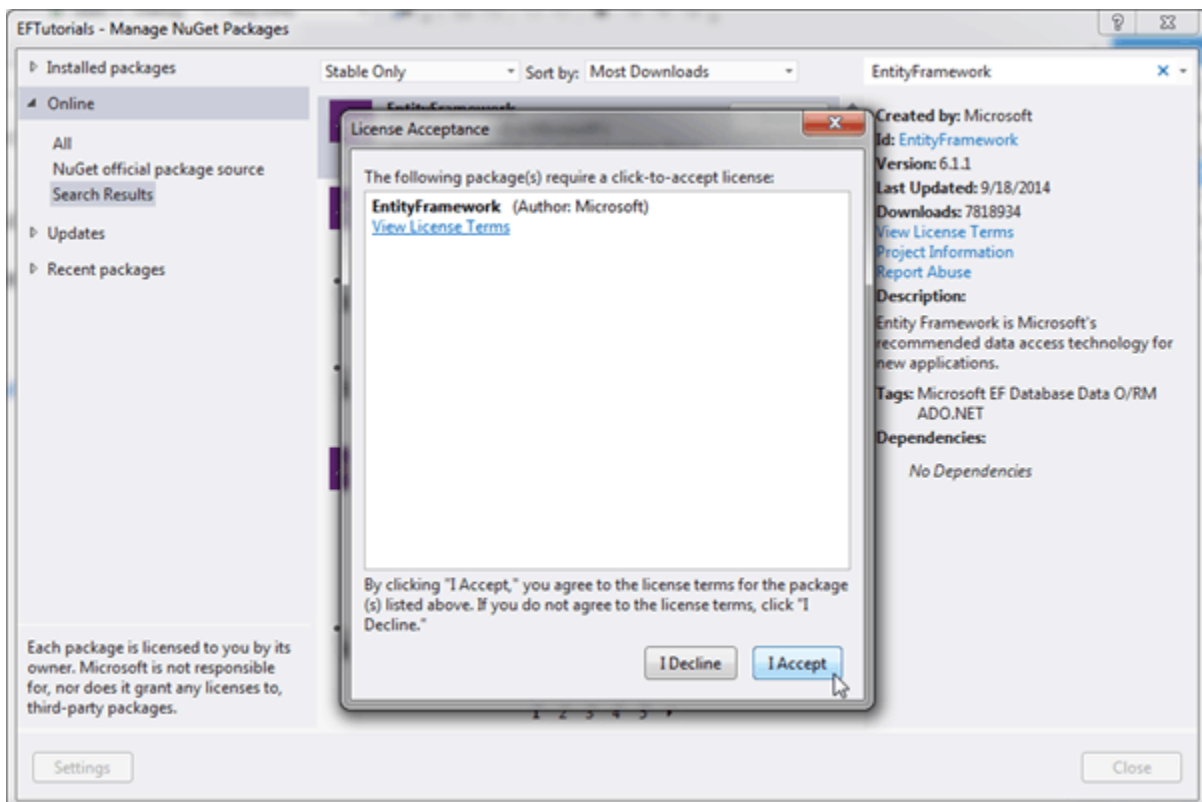
Sẽ mở ra hộp thoại Manage NuGet packages. Bây giờ chọn Online ở thanh bên trái và tìm kiếm EntityFramework như bên dưới



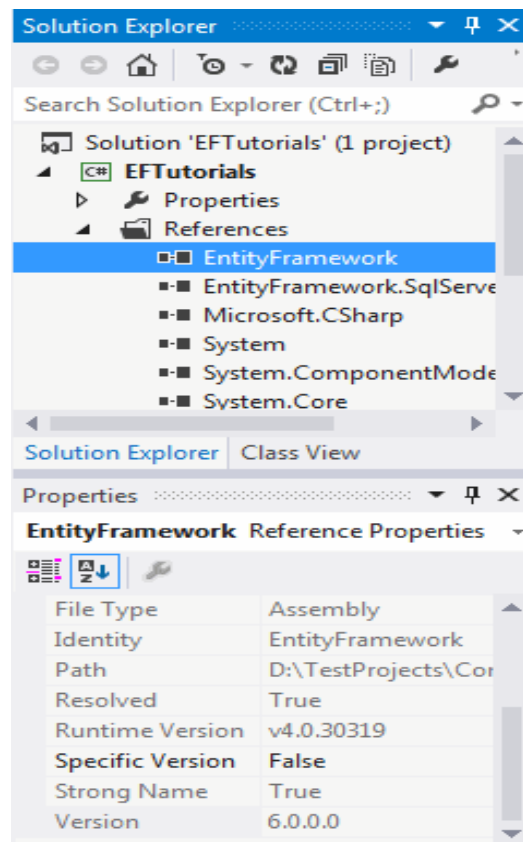
Sẽ tìm kiếm tất cả các gói liên quan tới Entity Framework. Chọn EntityFramework và bấm Install.



Bấm vào nút I Accept trong hộp thoại License Acceptance và sẽ bắt đầu cài đặt.



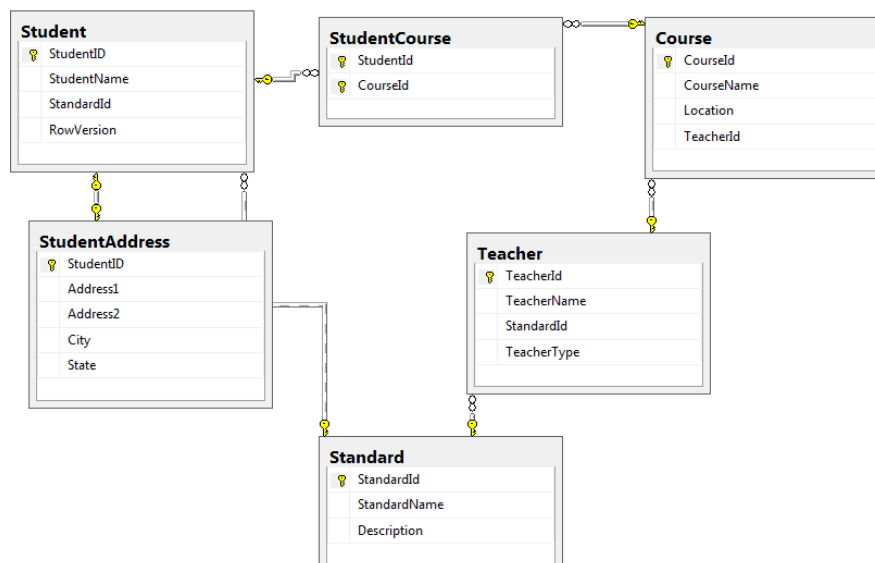
Sau khi cài đặt chắc chắn rằng phiên bản thích hợp của EntityFramework.dll là được đưa vào trong project.



Giờ bạn đã sẵn sàng để sử dụng Entity Framework trong project của bạn.

Cài đặt CSDL

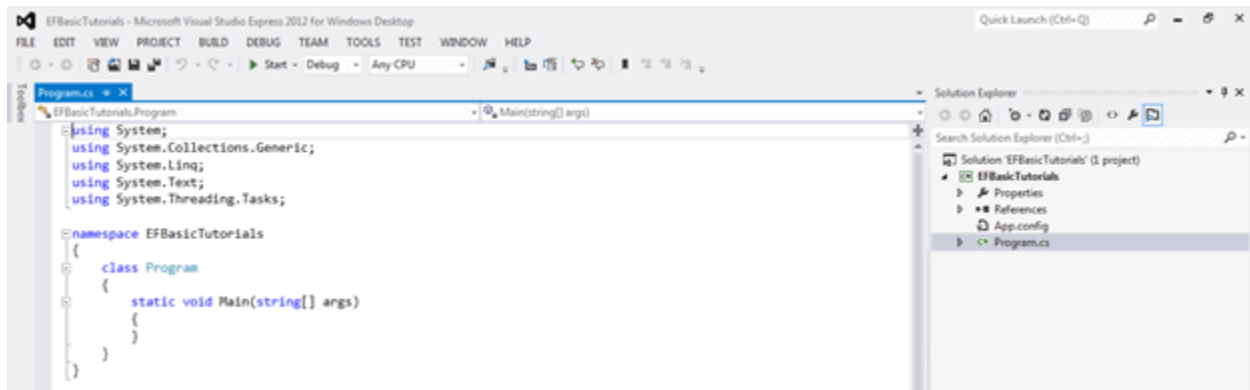
Bài hướng dẫn này sẽ sử dụng SchoolDB mẫu với các bảng khác nhau, stored procedures và views. CSDL SchoolDB được thiết kế như dưới đây:



Bạn có thể nhìn thấy trong sơ đồ ở trên CSDL SchoolDB mẫu bao gồm các bảng với các quan hệ sau cho mục đích demo.

- **One-to-One:** Student và StudentAddress có quan hệ one-to-one. Student có một hoặc không có StudentAddress.
- **One-to-Many:** Standard và Teacher có quan hệ one-to-many. nhiều Teachers có thể kết hợp với một Standard.
- **Many-to-Many:** Student và Course có quan hệ many-to-many sử dụng bảng StudentCourse và bảng StudentCourse bao gồm StudentId và CourseId. Do đó một sinh viên có thể tham gia nhiều khóa học và một khóa học cũng có thể có nhiều sinh viên.

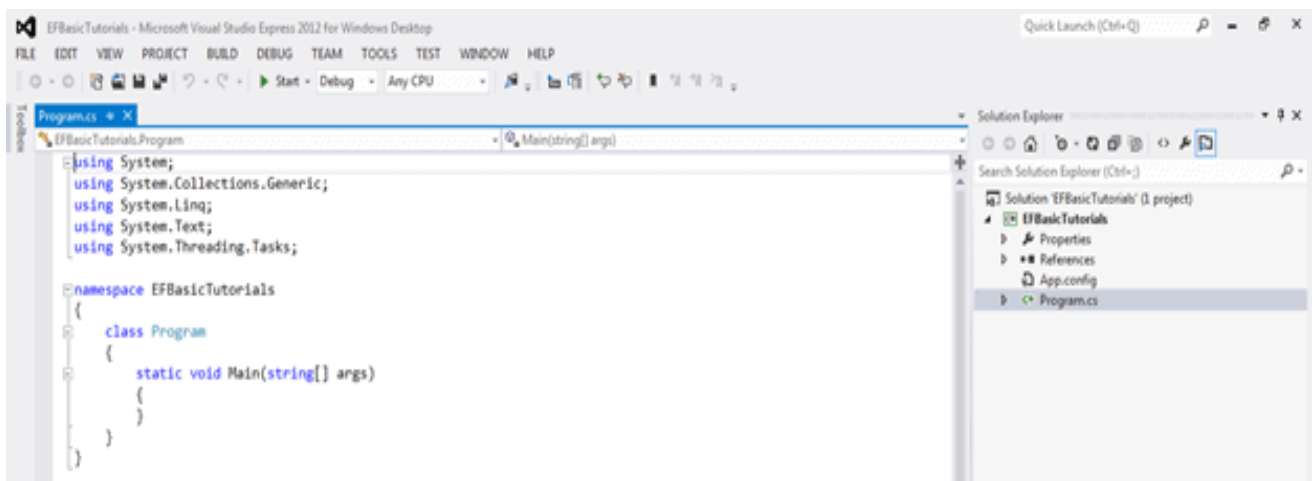
Phần 3 – Tạo Entity Data Model



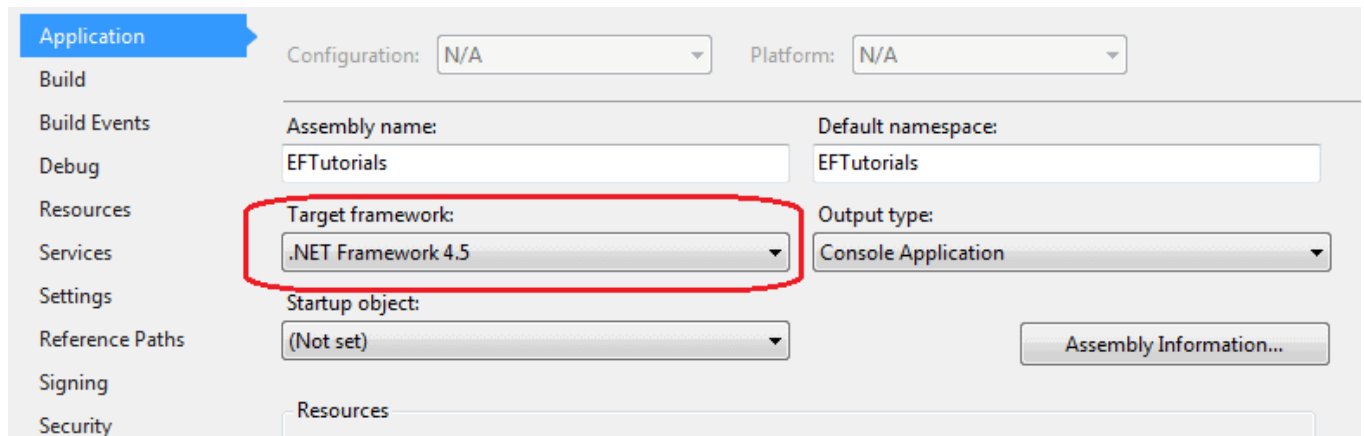
Chúng ta sẽ tạo một Entity Data Model (EDM) cho CSDL SchoolDB và hiểu những khối xây dựng cơ bản.

EDM là một mô hình mà ở đó mô tả các thực thể và những quan hệ giữa chúng. Hãy tạo một EDM đơn giản đầu tiên cho CSDL SchoolDB sử dụng Visual Studio 2012 và Entity Framework 6.

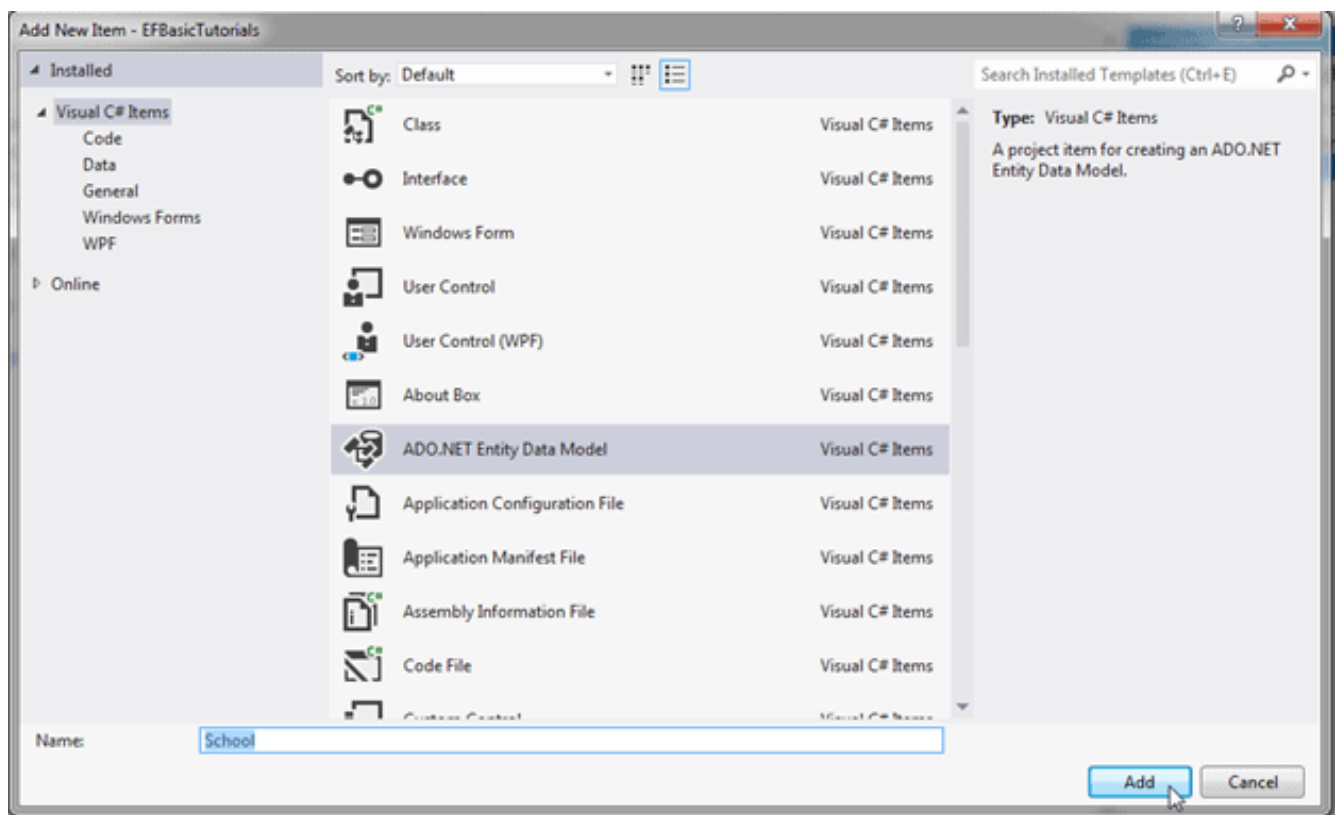
Mở Visual Studio 2012 và tạo một console project.



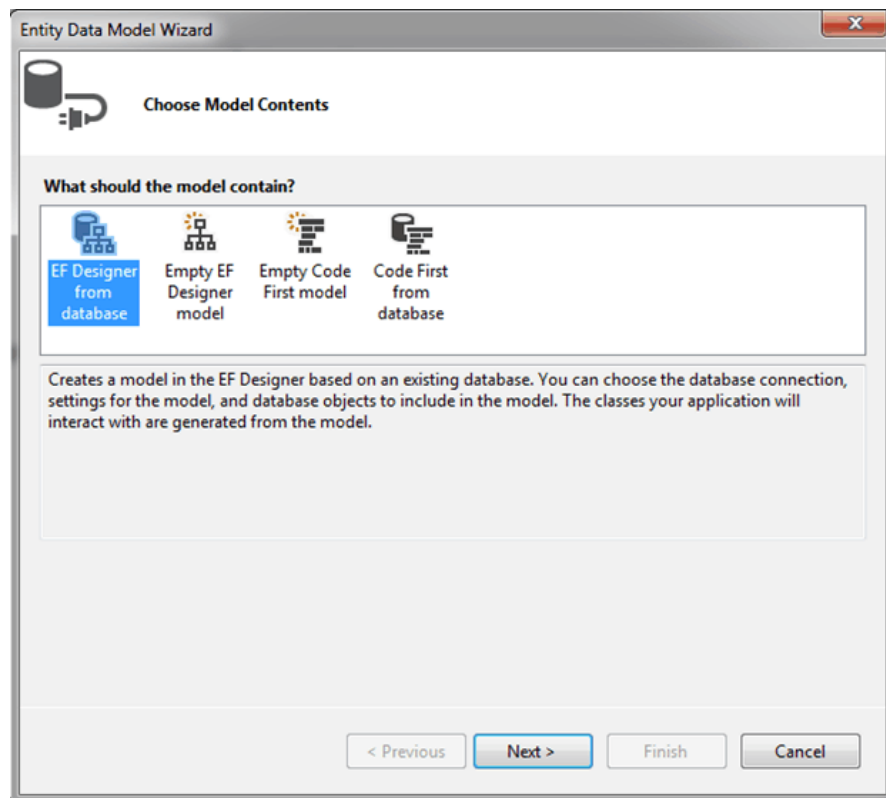
Tới PROJECT menu của visual studio -> {project name} properties – và chắc chắn rằng target framework của project là .NET Framework 4.5 như dưới đây:



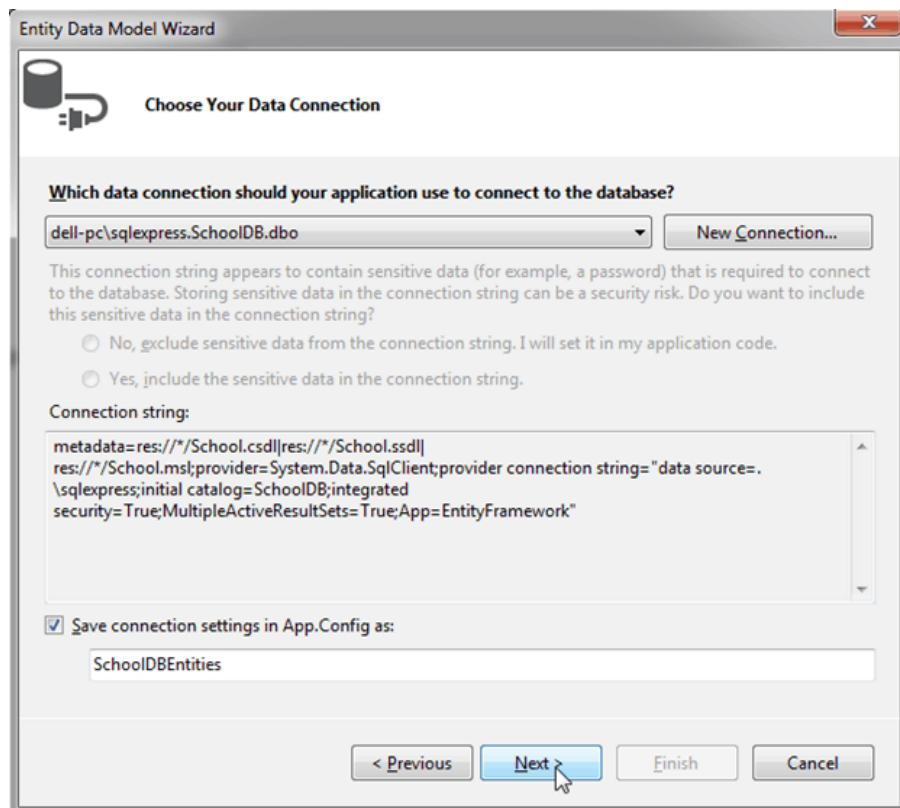
2. Giờ chúng ta thêm EDM bằng cách click chuột phải vào project trong solution explorer-> Add -> click New Item và chọn ADO.NET Entity Data Model từ hộp thoại, đặt tên là 'School' và click nút Add.



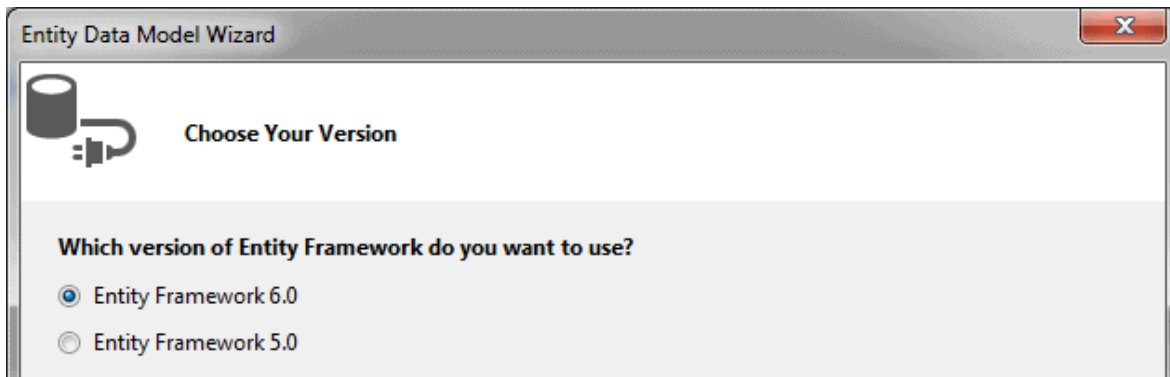
3. Entity Data Model Wizard trong VS2012 mở ra với 4 lựa chọn: EF Designer from database cho phương pháp tiếp cận Database First, Empty EF Designer model cho phương pháp tiếp cận Model First, Empty Code First model và Code First from database cho phương pháp tiếp cận Code First. Chúng ta sẽ tập trung vào phương pháp tiếp cận Database-First trong những bài hướng dẫn cơ bản này vì vậy chọn EF Designer from database và click Next.



4. Bạn có thể chọn từ DB Connections sẵn có của bạn hoặc tạo một kết nối mới bằng cách bấm nút 'New Connection'. Chúng ta sẽ sử dụng DB connection sẵn có tới CSDL SchoolDB. Thao tác này cũng sẽ thêm một connection string tới file app.config của bạn với hậu tố mặc định là tên CSDL. Bạn có thể đổi cái này nếu bạn muốn. Click 'Next' sau khi bạn cài đặt DB connection của bạn.

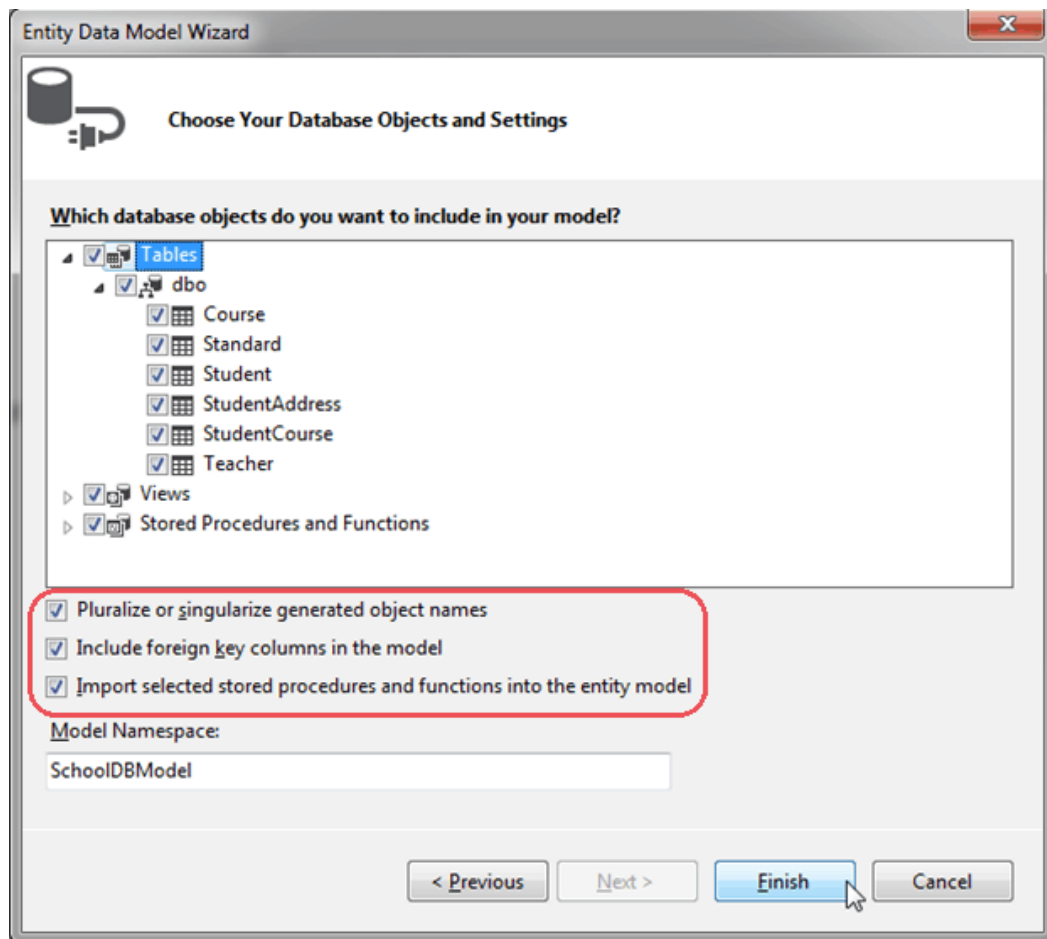


5. Trong bước này bạn cần chọn phiên bản của Entity Framework. Chúng ta sẽ sử dụng Entity Framework 6.0 trong các bài hướng dẫn cơ bản vì vậy chọn Entity Framework 6.0 và click Next.



Note: Nếu bạn đã cài đặt phiên bản mới nhất của Entity Framework sử dụng NuGet manager như trong bài Cài đặt môi trường rồi bước này của wizard sẽ không xuất hiện khi bạn đã cài đặt Entity Framework.

Bước này sẽ hiển thị tất cả Tables, Views và Stored Procedures (SP) trong CSDL. Lựa chọn Tables, Views and SPs bạn muốn và giữ các checkbox mặc định rồi click 'Finish'. Bạn có thể đổi Model Namespace nếu bạn muốn.



Note:

Pluralize or singularize generated object names checkbox chuyển tên một tập thực thể thành số ít, nếu tên bảng trong CSDL là số nhiều. VD: nếu SchoolDB có tên bảng Students rồi tập thực thể sẽ chuyển thành số ít Student. Tương tự quan hệ giữa những model sẽ là số nhiều nếu bảng có những quan hệ one-to-many hoặc many-to-many với những bảng khác. VD: bảng Student có quan hệ many-to-many với bảng Course vậy tập thực thể Student sẽ có tên thuộc tính số nhiều 'Courses' cho một tập hợp các khóa học.

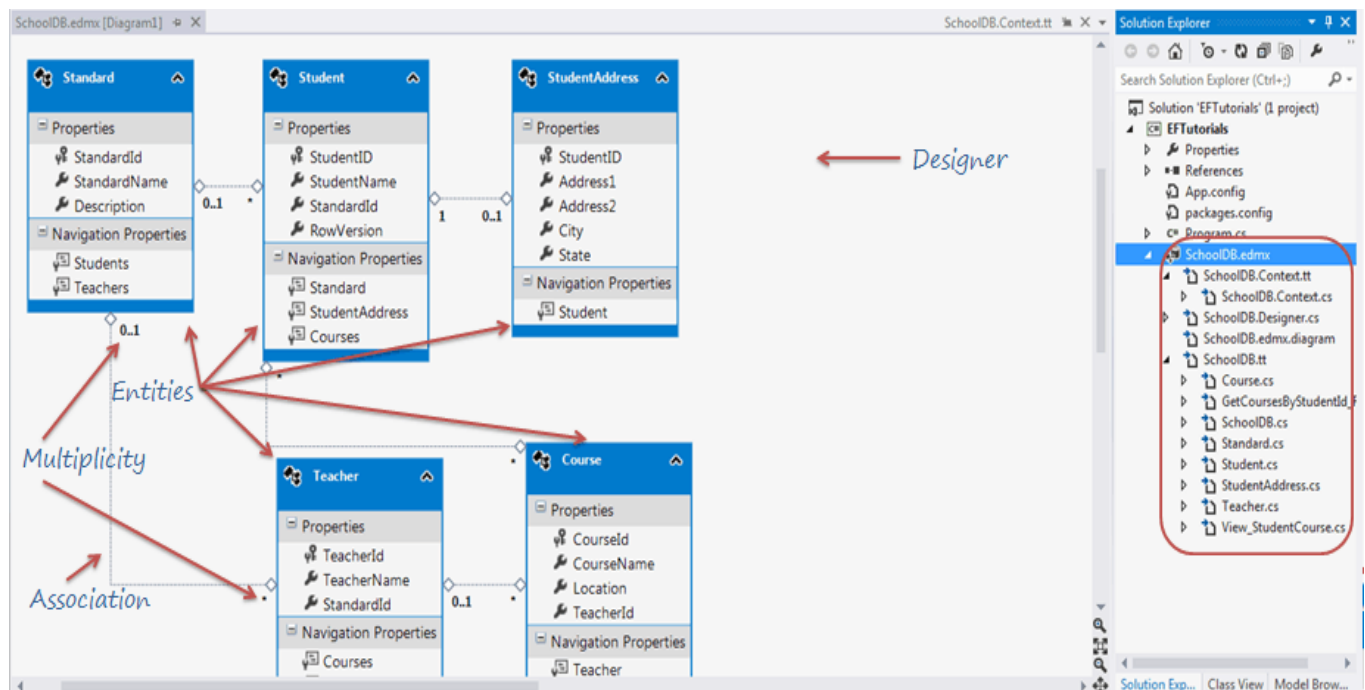
Checkbox thứ hai, **Include foreign key columns in the model**, gồm thuộc tính khóa ngoại rõ ràng để trình bày cho khóa ngoại. VD: bảng Student có quan hệ one-to-many với bảng Standard. Vậy mỗi sinh viên được kết hợp với chỉ một standard. Để trình bày điều này trong mô hình, tập thực thể Student gồm thuộc tính StandardId với thuộc tính điều hướng Standard. Nếu checkbox này là uncheck rồi nó sẽ chỉ có thuộc tính Standard, mà không có StandardId trong tập thực thể Student.

The third checkbox, **Import selected stored procedures and functions into entity model**, automatically creates Function Imports for the stored procedures and functions. You don't need to manually import this, as was necessary prior to Entity Framework 5.0.

Checkbox thứ ba, **Import selected stored procedures and functions into entity model**, tự động tạo Function Imports cho các stored procedures và functions. Bạn không cần import thủ công

Sau khi click 'Finish', một file School.edmx sẽ được thêm vào project của bạn.

Mở EDM designer bằng cách double click file School.edmx. File này sẽ hiện thị tất cả các thực thể của các bảng bạn đã chọn và những quan hệ giữa chúng như bên dưới:



EDM cũng thêm một chuỗi kết nối vào file config như dưới đây:

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false"/>
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>
  </startup>
  <entityFramework>
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
EntityFramework"/>
    <providers>
      <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer"/>
    </providers>
  </entityFramework>
  <connectionStrings>
    <add name="SchoolDBEntities"

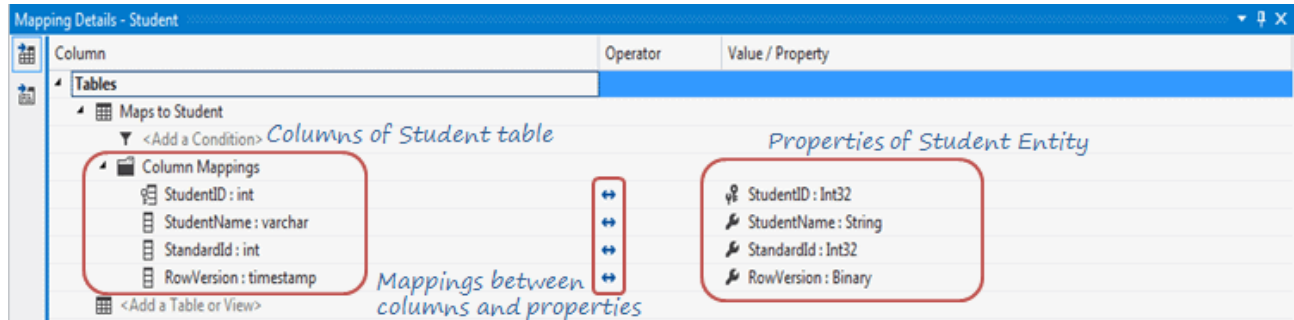
connectionString="metadata=res://*/SchoolDB.csdl|res://*/SchoolDB.ssdl|r
es://*/SchoolDB.msl;provider=System.Data.SqlClient;provider connection
string=&quot;data source=.\sqlexpress;initial
catalog=SchoolDB;integrated
security=True;multipleactiveresultsets=True;application
name=EntityFramework&quot;;" providerName="System.Data.EntityClient"/>
  </connectionStrings>
</configuration>
```

Theo cách này bạn có thể tạo một EDM đơn giản từ CSDL hiện tại của bạn.

Bây giờ chúng ta sẽ thử kiểm tra tất cả các khối xây dựng khi khởi tạo EDM (School.edmx) như hình minh họa bên trên.

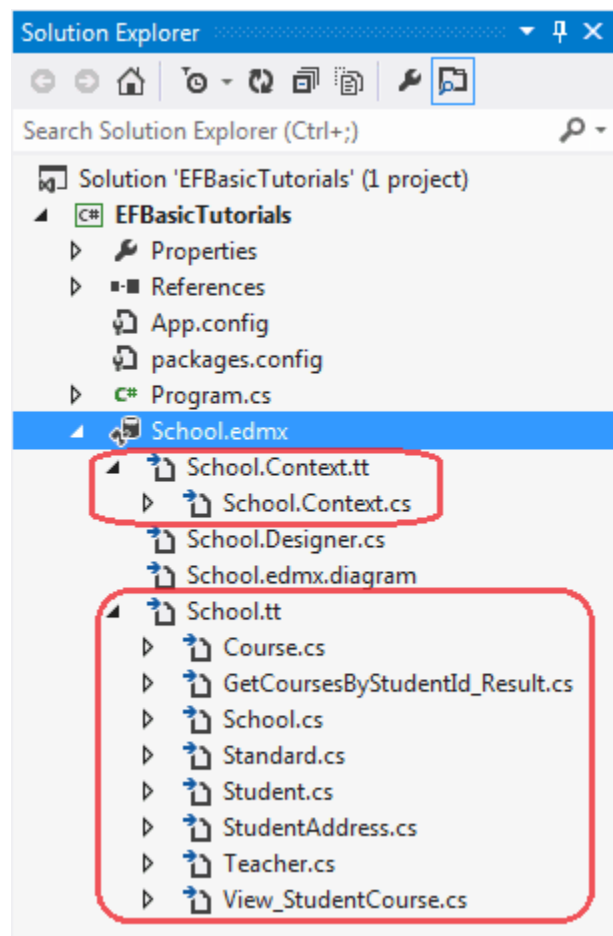
Entity-Table Mapping:

Mỗi thực thể trong EDM là nối với bảng của CSDL. Bạn có thể kiểm tra entity-table mapping bằng cách click vào bất kỳ thực thể nào trong EDM designer -> chọn Table Mapping. Cũng như nếu bạn thay đổi bất kỳ tên thuộc tính của thực thể từ designer rồi table mapping sẽ ánh xạ thay đổi đó tự động.

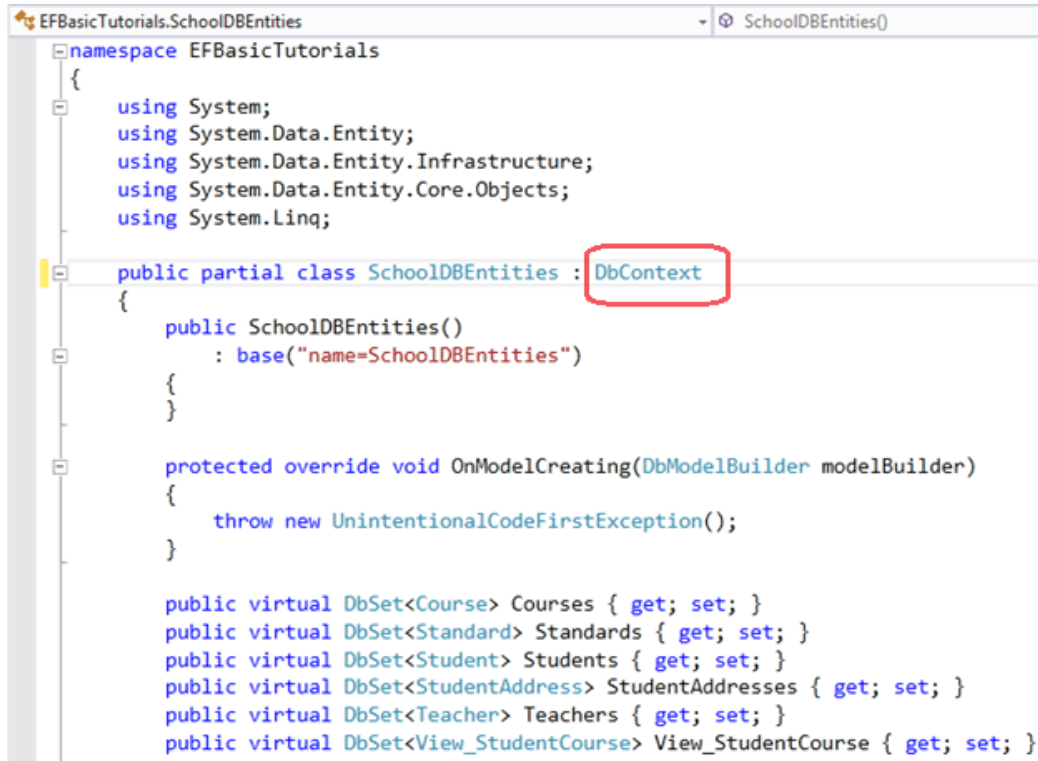


Context & Entity Classes:

Mỗi Entity Data Model khởi tạo một lớp context và lớp thực thể cho mỗi bảng CSDL trong EDM. Mở rộng School.edmx và bạn sẽ thấy hai file quan trọng {EDM Name}.Context.tt và {EDM Name}.tt:



School.Context.tt: File mẫu T4 này khởi tạo một lớp context bất cứ khi nào bạn thay đổi Entity Data Model (.edmx file). Bạn có thể thấy file context bằng cách mở rộng School.Context.tt. Lớp context lưu trữ trong file {EDM Name}.context.cs. Lớp context mặc định là {DB Name} + Entities. VD: tên lớp context cho SchoolDB là SchoolDBEntities rồi lớp context sẽ dẫn xuất từ lớp DbContext trong Entity Framework. (Trước EF 5.0 nó được dẫn xuất từObjectContext)



```
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; }
    }
}
```

School.tt: là một file T4 mẫu khởi tạo các lớp thực thể cho mỗi bảng của CSDL. Các lớp thực thể là những lớp POCO (Plain Old CLR Object). Đoạn mã sau trình bày cho thực thể Student

```
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; }

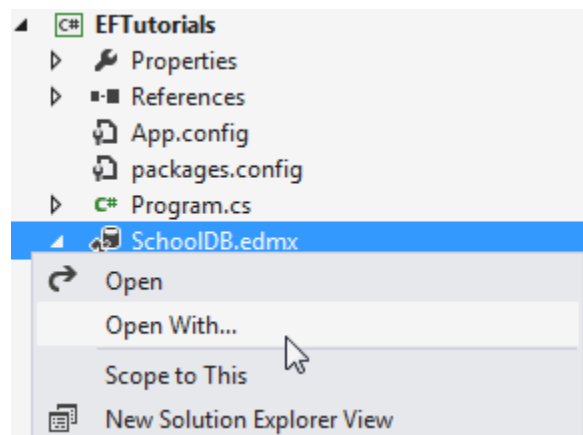
    public virtual Standard Standard { get; set; }
    public virtual StudentAddress StudentAddress { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}
```


EDM Designer: trình bày mô hình quan niệm của bạn. Nó bao gồm Entities, những liên kết và bội số giữa những thực thể. Ban đầu, nó sẽ trông giống chính xác cấu trúc bảng CSDL của bạn nhưng bạn có thể thêm, gộp hoặc xóa cột, và nó không yêu cầu bởi ứng dụng của bạn từ designer này. Bạn có thể thậm chí thêm một đối tượng mới trong mô hình này, nó có thể có những cột từ các bảng khác nhau từ menu context như hình bên trên. Nhớ rằng bất cứ sự thay đổi nào thực hiện ở đây đều nói với storage model. Vì vậy bạn phải cẩn thận trong khi thực hiện bất kỳ sự thay đổi nào trong designer.

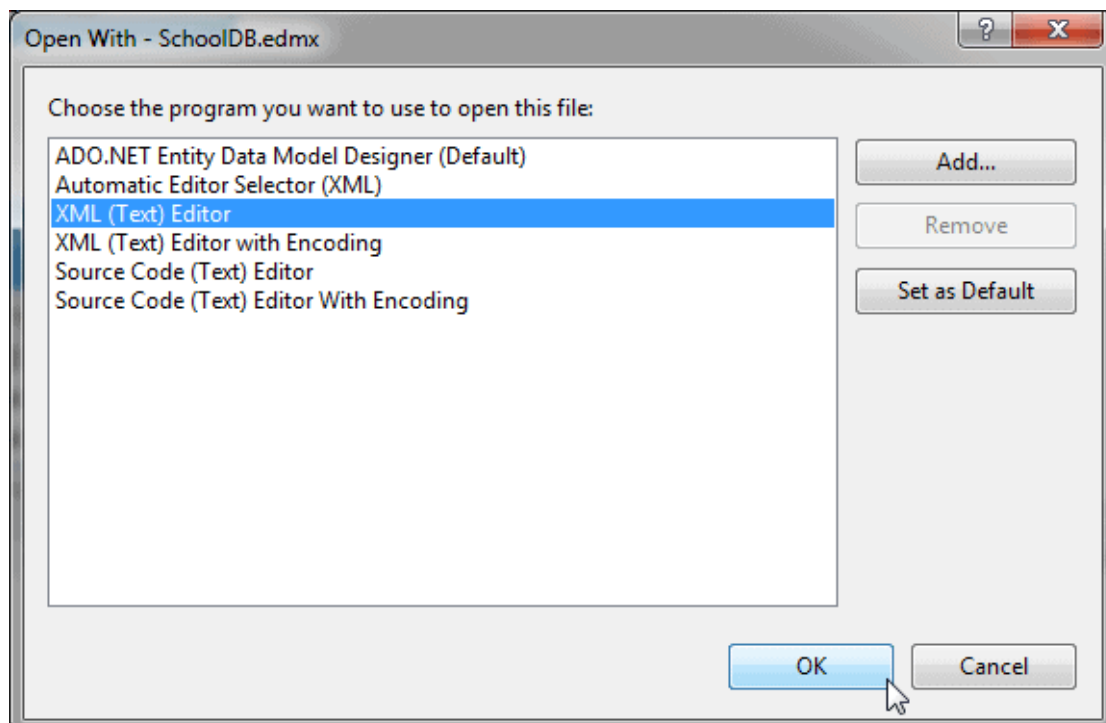
You can open this EDM designer in XML view where you can see all the three parts of the EDM – Conceptual schema (CSDL), Storage schema (SSDL) and mapping schema (MSL), together in XML view.

Bạn có thể mở EDM designer này trong XML và có thể nhìn thấy tất cả ba phần của EDM – Conceptual schema (CSDL), Storage schema (SSDL) và mapping schema (MSL), cùng trong XML view.

Click chuột phải trên School.edmx -> click ‘Open with..’, sẽ mở một cửa sổ.



Chọn ‘XML (text) Editor’ trong cửa sổ.

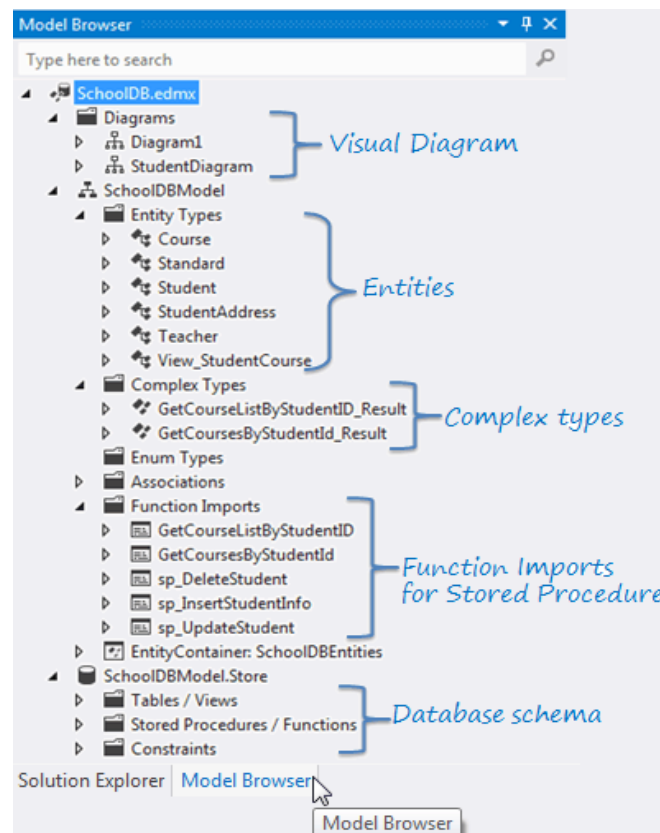


Visual Studio không thể hiển thị model trong Design view và trong XML format ở cùng thời điểm, vì vậy bạn sẽ nhìn thấy một tin nhắn hỏi nó có OK để đóng Design view của model. Click Yes. Và nó sẽ mở XML format view. Bạn có thể nhìn thấy XML view sau bằng cách mở tất cả tóm tắt như bên dưới:

```
School.edmx  X
<?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>...</edmx:Mappings>
  </edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx/Designer">...</Designer>
</edmx:Edmx>
```

Bạn có thể nhìn thấy nội dung SSDL (Store schema definition language), nội dung CSDL (Common Schema Definition Language) và nội dung C-S mapping ở đây. Nếu bạn mở rộng SSDL và CSDL, mỗi cái có vài nốt XML thông thường dưới mỗi nốt schema. Bạn không cần chỉnh sửa dữ liệu XML bởi vì dữ liệu này có thể thực hiện dễ dàng hơn trong Model Browser.

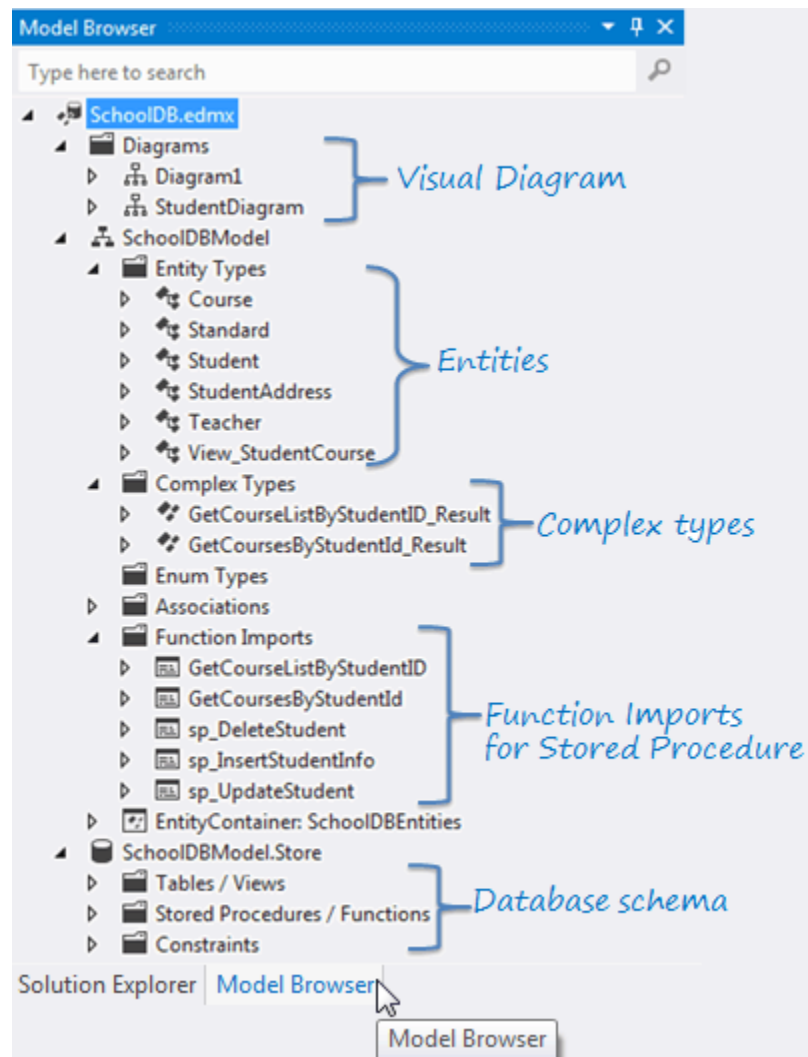
Phần 4 – Model Browser



Chúng ta đã tạo Entity Data Model đầu tiên cho CSDL School trong phần trước. Visual designer của EDM không hiển thị được tất cả các đối tượng nó tạo. Nó chỉ hiển thị những thực thể nối với những bảng và Views trong CSDL.

Model Browser đưa cho bạn tất cả thông tin về tất cả các đối tượng và functions EDM đã tạo. Mở Model Browser bằng cách click chuột phải vào EDM Designer và chọn Model Browser từ context menu.

Model browser chứa tất cả thông tin về EDM, conceptual model của nó, storage model và thông tin mapping như bên dưới:



Như bạn nhìn thấy trong hình minh họa trên, model browser chứa những đối tượng sau:

Diagrams: Model browser chứa visual diagrams của EDM. Chúng ta đã nhìn thấy một visual diagram mặc định được tạo bởi EDM. Bạn cũng có thể tạo nhiều diagrams cho một EDM, nếu ứng dụng của bạn có một số lượng lớn những thực thể.

Entity Types: liệt kê tất cả các kiểu lớp nối với các bảng của CSDL.

Complex Types: là những lớp được khởi tạo bởi EDM để chứa kết quả của stored procedures, table-valued function,... Những kiểu phức tạp là những lớp được tùy biến cho những mục đích khác nhau.

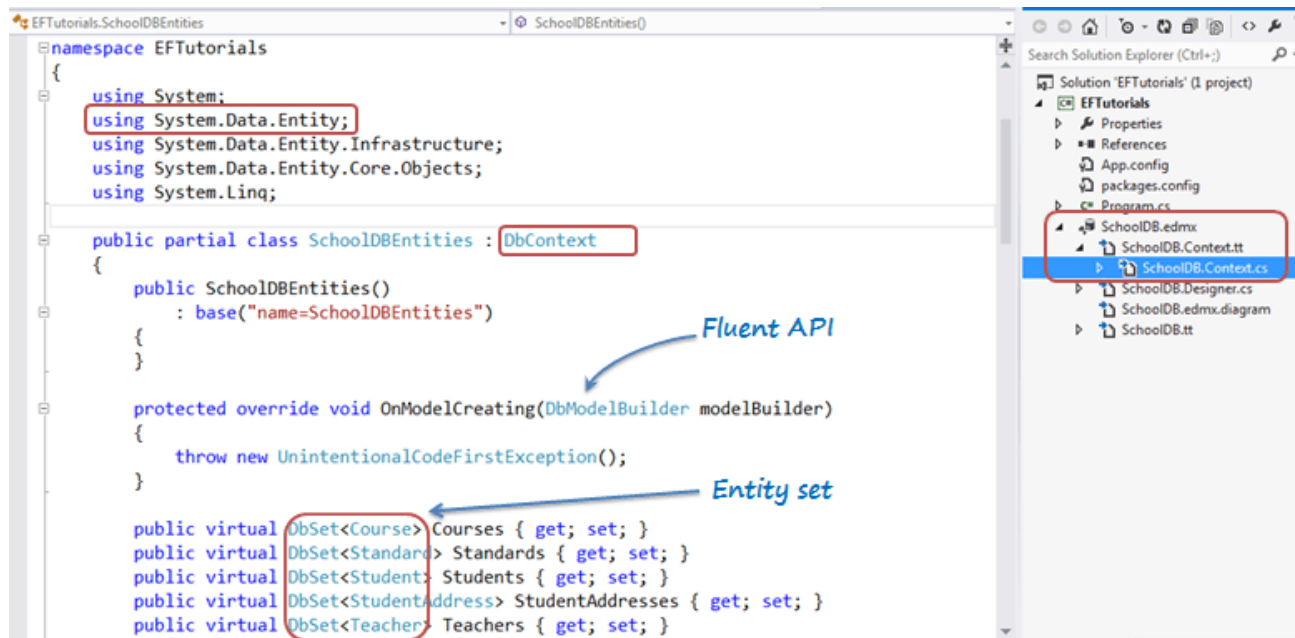
Enum Types: liệt kê tất cả thực thể được sử dụng như Enum trong Entity Framework.

Associations: liệt kê tất cả quan hệ khóa ngoại giữa những kiểu thực thể.

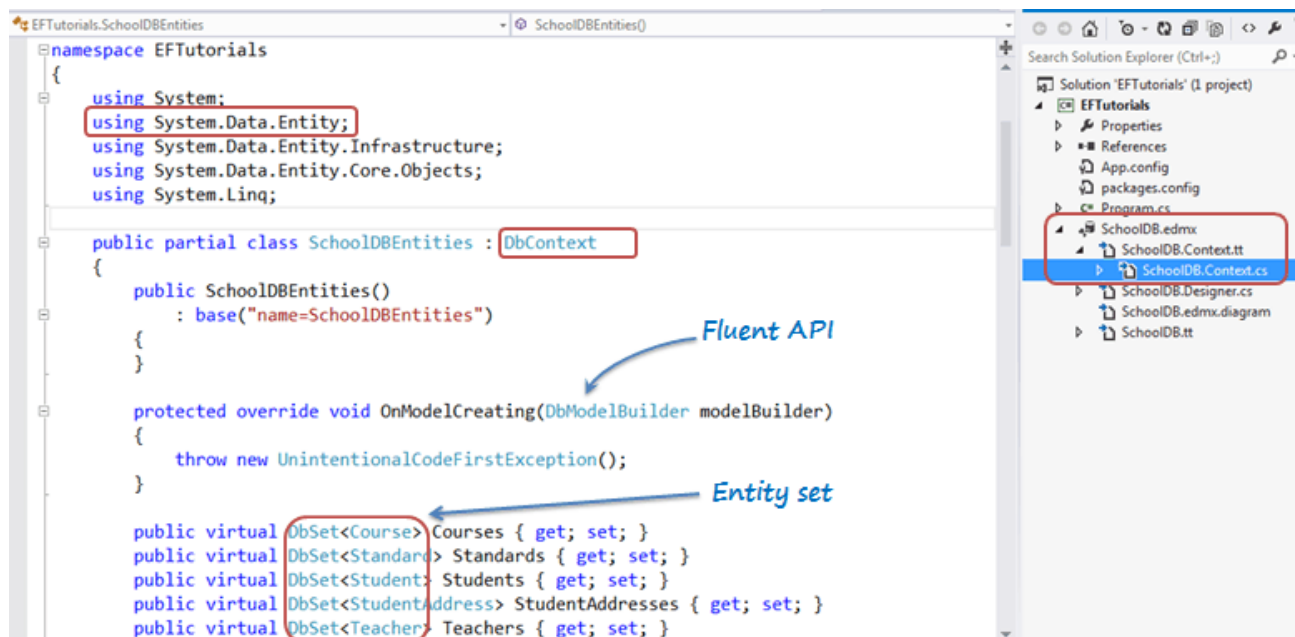
Function Imports: liệt kê tất cả các hàm sẽ nối với stored procedures, table-valued functions,... Stored procedures và table-valued functions sẽ được sử dụng như là hàm chứ không phải một thực thể trong EF.

.Store: Store represents database schema (SSDL).

Phần 5 – DbContext

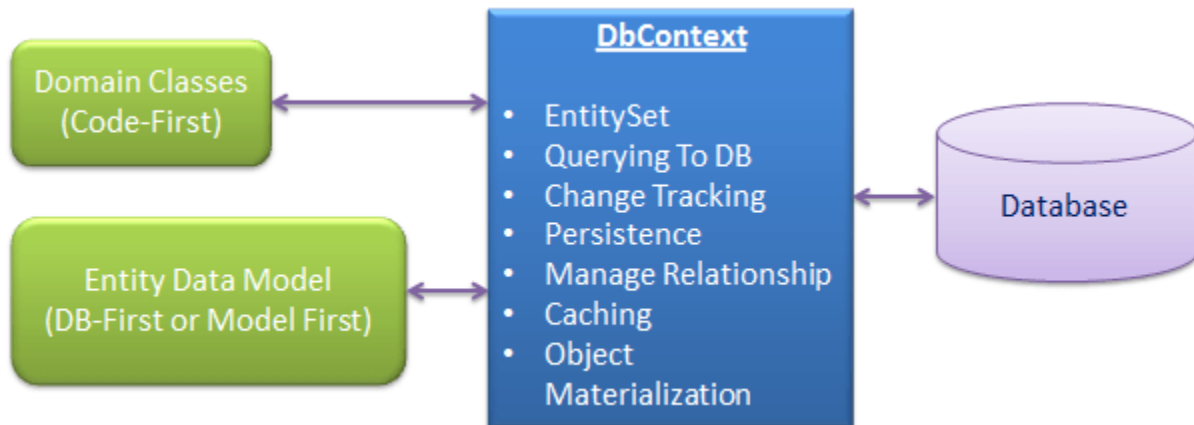


Như bạn đã nhìn thấy trong phần 3 Tạo Entity Data Model, EDM khởi tạo lớp `SchoolDBEntities` được dẫn xuất từ lớp `System.Data.Entity.DbContext` như bên dưới. Lớp dẫn xuất từ `DbContext` được gọi là lớp context trong Entity Framework.



Trước EntityFramework 4.1, EDM thường khởi tạo lớp context được dẫn xuất từ lớpObjectContext. Nó có một chút rắc rối khi làm việc vớiObjectContext. DbContext khái niệm tương tựObjectContext. Nó là một trình bao bọc quanhObjectContext và hữu ích trong tất cả mô hình phát triển: Code First, Model First và Database First.

DbContext là một phần quan trọng của Entity Framework. Nó là một cầu nối giữa lớp domain hoặc thực thể và CSDL của bạn.



DbContext là lớp chính chịu trách nhiệm cho việc tương tác với dữ liệu như là đối tượng. DbContext chịu trách nhiệm cho các hoạt động sau:

EntitySet: DbContext chứa tập thực thể (`DbSet<TEntity>`) cho tất cả thực thể nối với những bảng của CSDL.

Querying: DbContext chuyển đổi những truy vấn LINQ-to-Entities thành truy vấn SQL và gửi nó tới CSDL.

Change Tracking: Nó giữ việc theo dõi những thay đổi xảy ra trong những thực thể sau khi nó đã truy vấn từ CSDL.

Persisting Data: Nó cũng thực hiện các thao tác Insert, Update và Delete tới CSDL dựa trên những gì mà thực thể thể hiện.

Caching: DbContext mặc định thực hiện caching mức đầu tiên. Nó lưu những thực thể đã được nhận suốt vòng đời của một lớp context.

Manage Relationship: DbContext cũng quản lý những quan hệ sử dụng CSDL, MSL và SSDL trong phương pháp tiếp cận DB-First hoặc Model-First hoặc sử dụng fluent API trong phương pháp tiếp cận Code-First.

Object Materialization: DbContext chuyển đổi bảng dữ liệu thô vào những đối tượng thực thể.

Sau đây là một ví dụ của lớp `SchoolDBEntities` class (lớp class dẫn xuất DbContext) khởi tạo với EDM cho CSDL SchoolDB trong những bài hướng dẫn trước.

```

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; }
        public virtual DbSet<View_StudentCourse> View_StudentCourse {
get; set; }

```

```

        public virtual ObjectResult<GetCoursesByStudentId_Result>
GetCoursesByStudentId(Nullable<int> studentId)
        {
            var studentIdParameter = studentId.HasValue ?
                new ObjectParameter("StudentId", studentId) :
                new ObjectParameter("StudentId", typeof(int));

            return
                ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<GetCoursesBy
StudentId_Result>("GetCoursesByStudentId", studentIdParameter);
        }

        public virtual int sp_DeleteStudent(Nullable<int> studentId)
        {
            var studentIdParameter = studentId.HasValue ?
                new ObjectParameter("StudentId", studentId) :
                new ObjectParameter("StudentId", typeof(int));

            return
                ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp_DeleteSt
udent", studentIdParameter);
        }

        public virtual ObjectResult<Nullable<decimal>>
sp_InsertStudentInfo(Nullable<int> standardId, string studentName)
        {
            var standardIdParameter = standardId.HasValue ?
                new ObjectParameter("StandardId", standardId) :
                new ("StandardId", typeof(int));

```



```

        var studentNameParameter = studentName != null ?
            new ObjectParameter("StudentName", studentName) :
            new ObjectParameter("StudentName", typeof(string));

        return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<Nullable<decimal>>("sp_InsertStudentInfo", standardIdParameter,
        studentNameParameter);
    }

    public virtual int sp_UpdateStudent(Nullable<int> studentId,
    Nullable<int> standardId, string studentName)
    {
        var studentIdParameter = studentId.HasValue ?
            new ObjectParameter("StudentId", studentId) :
            new ObjectParameter("StudentId", typeof(int));

        var standardIdParameter = standardId.HasValue ?
            new ObjectParameter("StandardId", standardId) :
            new ObjectParameter("StandardId", typeof(int));

        var studentNameParameter = studentName != null ?
            new ObjectParameter("StudentName", studentName) :
            new ObjectParameter("StudentName", typeof(string));

        return
        ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("sp_UpdateStu
        dent", studentIdParameter, standardIdParameter, studentNameParameter);
    }
}

```

As you can see in the above example, context class (SchoolDBEntities) includes entity set of type DbSet<TEntity> for all the entities. Learn more about DbSet class [here](#). It also includes functions for the stored procedures and views included in EDM.

Như bạn có thể thấy trong ví dụ trên, lớp context (SchoolDBEntities) gồm tập thực thể của kiểu DbSet<TEntity> cho tất cả những thực thể. Nó cũng gồm những hàm cho stored procedures and views included trong EDM.

Lớp Context ghi đè phương thức OnModelCreating. Tham số DbModelBuilder được gọi là Fluent API, nó có thể sử dụng để cấu hình những thực thể trong phương pháp tiếp cận Code-First.

Cài đặt DbContext:

Bạn có thể sử dụng DbContext bằng cách cài đặt lớp context và sử dụng cho thao tác CRUD như bên dưới.

```
using (var ctx = new SchoolDBEntities()) { //Can perform CRUD operation using ctx here.. }
```

Getting ObjectContext từ DbContext:

DbContext API là dễ sử dụng hơn ObjectContext API cho tất cả các tác vụ thông thường. Tuy nhiên bạn có thể lấy tham chiếu của ObjectContext từ DbContext để sử dụng những chức năng của ObjectContext. Điều này có thể thực hiện bằng cách sử dụng IObjectContextAdapter như bên dưới:


```
using (var ctx = new SchoolDBEntities()) { var objectContext = (ctx as System.Data.Entity.Infrastructure.IObjectContextAdapter).ObjectContext; //use objectContext here.. }
```

EDM cũng khởi tạo các lớp thực thể. Học về những kiểu khác nhau của thực thể trong phần tiếp theo.

Phần 6 – Entity #1

```
using (SchoolDBEntities context = new SchoolDBEntities())  
{  
    Student studentEntity = context.Students.FirstOrDefault<Student>();  
    studentEntity (System.Data.Entity.DynamicProxies.Student_8C9C606A94AFB4EB28256E8214C30B620217205B13ED3916324B4A  
}
```

I. Những kiểu thực thể trong Entity Framework:

Chúng ta tạo EDM từ CSDL hiện tại trong phần trước. Như bạn đã được học trong phần trước rằng EDM chứa những thực thể cho mỗi bảng trong CSDL. Có hai kiểu thực thể trong Entity Framework 5.0/6.0: POCO entity và dynamic proxy entity

POCO Entity (Plain Old CLR Object):

Lớp POCO là lớp không phụ thuộc vào bất kỳ lớp framework cụ thể nào. Nó giống như bất kỳ lớp .Net thông thường khác đó là tại sao nó được gọi là “Plain Old CLR Objects”.

Những thực thể POCO này (cũng được biết đến như những đối tượng persistence-ignorant) hỗ trợ hầu hết cùng kiểu truy vấn, những hành vi insert, update, and delete như những kiểu thực thể được khởi tạo bởi Entity Data Model.

```
public class Student  
{  
    public Student()  
    {  
        this.Courses = new List<Course>();  
    }  
  
    public int StudentID { get; set; }  
    public string StudentName { get; set; }  
    public Nullable<int> StandardId { get; set; }  
  
    public Standard Standard { get; set; }  
    public StudentAddress StudentAddress { get; set; }  
    public IList<Course> Courses { get; set; }  
}
```

Dynamic Proxy (POCO Proxy):

Dynamic Proxy là một lớp runtime proxy của POCO entity. Nó giống như một lớp bọc phía ngoài của POCO entity. Những thực thể dynamic proxy cho phép lazy loading và automatic change tracking.

POCO entity nên đáp ứng những yêu cầu sau để trở thành một POCO proxy:

1. Một lớp POCO phải được khai báo với kiểu truy cập public.
2. Một lớp POCO không phải là sealed (NotInheritable in Visual Basic).
3. Một lớp POCO không phải là abstract (MustInherit in Visual Basic).
4. Mỗi thuộc tính điều hướng phải được khai báo như public, virtual.
5. Mỗi thuộc tính tập hợp phải là ICollection<T>.
6. ProxyCreationEnabled option không thể false (mặc định là true) trong lớp context.

Thực thể Student POCO sau đáp ứng tất cả những yêu cầu trên để trở thành thực thể dynamic proxy ở thời điểm khởi chạy.

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

Note: Mặc định dynamic proxy là được kích hoạt cho mỗi thực thể. Tuy nhiên bạn có thể vô hiệu hóa dynamic proxy bằng cách cài đặt ProxyCreationEnabled option thành false trong lớp context.

```
context.Configuration.ProxyCreationEnabled = false;
```

Mặc định EDM khởi tạo những thực thể POCO đáp ứng được những yêu cầu trên cho một dynamic proxy

Ở thời điểm khởi chạy, kiểu của Student sẽ là System.Data.Entity.DynamicProxies.Student như bên dưới:

```
using (SchoolDBEntities context = new SchoolDBEntities())
{
    Student studentEntity = context.Students.FirstOrDefault<Student>();
    studentEntity (System.Data.Entity.DynamicProxies.Student_8C9C606A94AFB4EB28256E8214C30B620217205B13ED3916324B4A)
}
```

Lấy kiểu thực thể từ một dynamic proxy:

Bạn có thể sử dụng `ObjectContext.GetObjectType()` để tìm kiểu thực thể của dynamic proxy như bên dưới:

```
using (SchoolDBEntities context = new SchoolDBEntities())
{
    Student studentEntity = context.Students.FirstOrDefault<Student>();
    var entityType = ObjectContext.GetObjectType(studentEntity.GetType());
}
entityType {Name = "Student" FullName = "EFBasicTutorials.Student"}
```

Thực thể có hai kiểu thuộc tính, Scalar và Navigation.

Scalar properties:

Thuộc tính Scalar là những thuộc tính mà những giá trị thực thể được chứa đựng trong thực thể. VD: Thực thể Student có những thuộc tính scalar giống như `StudentId` và `StudentName`. Những thuộc tính này tương đương với những cột trong bảng Student.

Navigation properties:

Thuộc tính Navigation là những con trỏ tới những thuộc tính liên quan khác. Student có thuộc tính Standard như là thuộc tính navigation và sẽ kích hoạt ứng dụng để điều hướng từ một Student tới thực thể Standard liên quan.

II. Vòng đời của thực thể:

Trước khi chúng ta là việc với CRUD operation (Create, Read, Update, Delete), nó là quan trọng để hiểu vòng đời của thực thể và EntityFramework quản lý nó như thế nào.

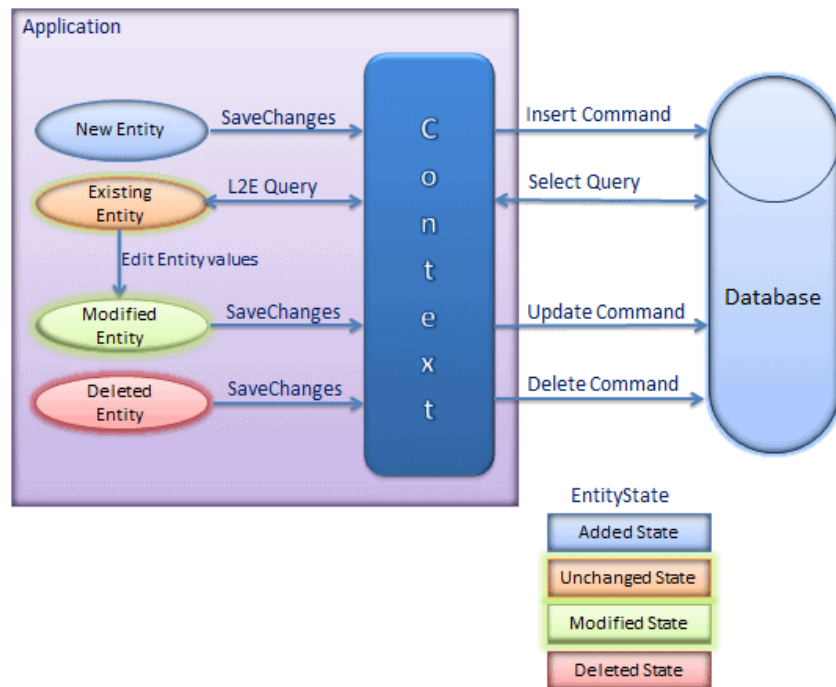
Suốt vòng đời của một thực thể, mỗi thực thể có một trạng thái thực thể dựa trên hoạt động được thực hiện trên nó qua context (`DbContext`). Trạng thái thực thể là một enum của kiểu `System.Data.Entity.EntityState` bao gồm các giá trị sau:

1. Added
2. Deleted
3. Modified
4. Unchanged
5. Detached

Context không chỉ giữ những tham chiếu tới tất cả các đối tượng nhận được từ CSDL mà còn giữ những trạng thái thực thể và duy trì những thay đổi tới những thuộc tính của thực thể. Chức năng này được biết như là Change Tracking.

Thay đổi trong trạng thái thực thể từ trạng thái Unchanged thành Modified chỉ là trạng thái được điều khiển tự động bởi context. Tất cả những thay đổi khác phải được thực hiện rõ ràng sử dụng những phương thức thích hợp của `DbContext` và `DbSet`.

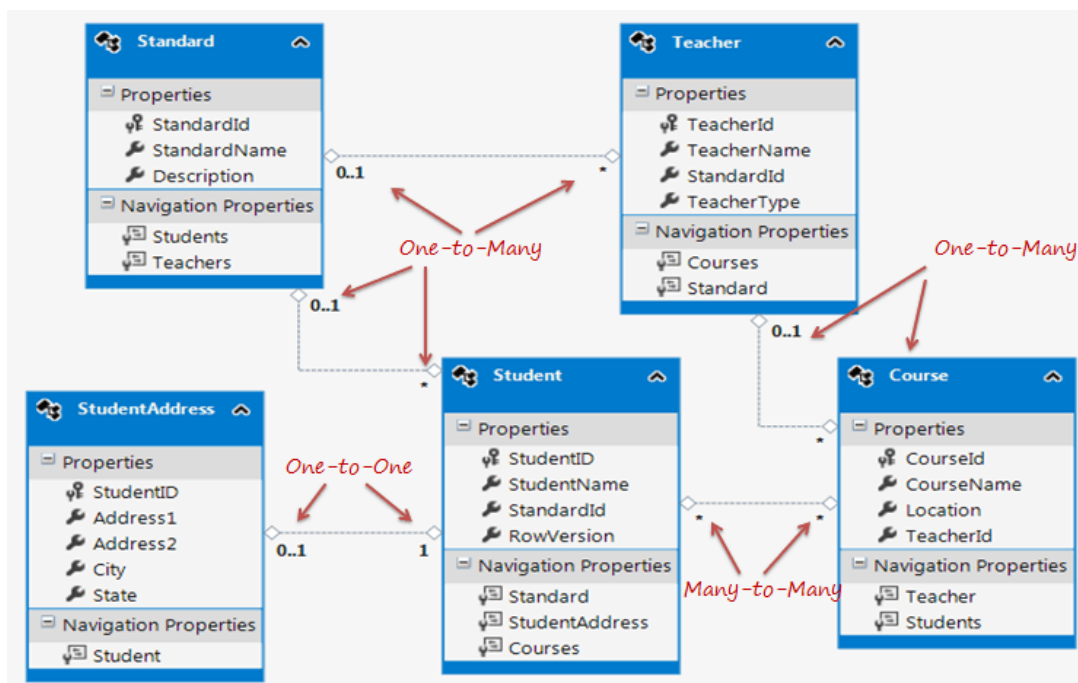
Hình sau minh họa hoạt động thực hiện thế nào trên những thay đổi trạng thái của thực thể lần lượt ảnh hưởng đến hoạt động của CSDL.



Như bạn nhìn thấy ở hình minh họa trên, thực thể mới trong context có trạng thái thực thể Added. Do đó context sẽ thực thi lệnh insert trên CSDL. Cùng một cách khi bạn nhận được một thực thể tồn tại sử dụng truy vấn L2E, nó sẽ có trạng thái thực thể Unchanged, điều này là bởi vì bạn đã vừa nhận được một thực thể và chưa thực hiện bất kỳ hành động trên nó. Khi bạn điều chỉnh giá trị của thực thể tồn tại, nó thay đổi trạng thái thành Modified và sẽ lần lượt thực thi lệnh update trên SaveChanges. Xóa thực thể từ context sẽ có trạng thái Deleted và sẽ lần lượt thực thi lệnh delete trên CSDL.

Vậy theo cách này, những hoạt động thực hiện trên những trạng thái thay đổi của thực thể. Context xây dựng và thực thi lệnh trên CSDL dựa vào trạng thái của một thực thể.

Phần 7 – Entity #2

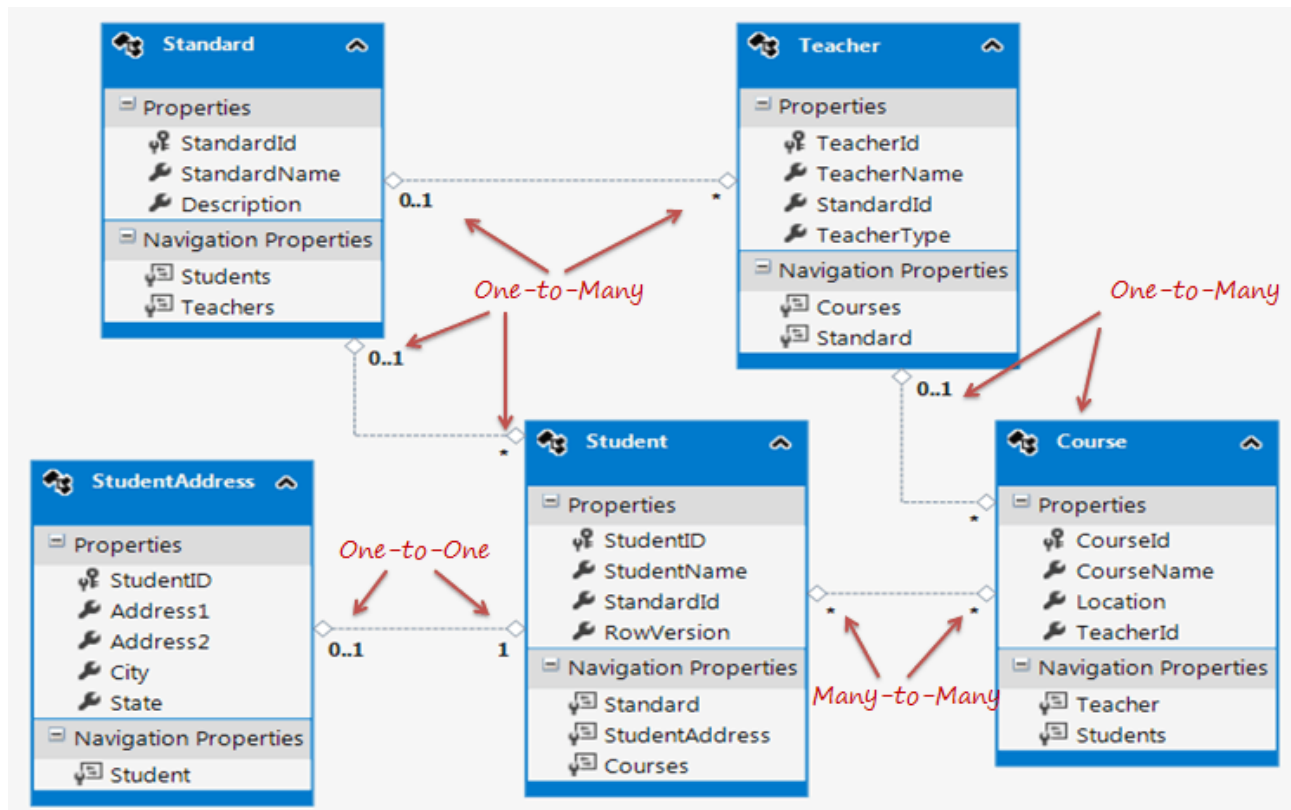


III. Những quan hệ của thực thể

Phần này bạn sẽ học được Entity Framework quản lý quan hệ giữa những thực thể như thế nào.

Entity Framework hỗ trợ 3 kiểu quan hệ giống như CSDL: One to One, One to Many, và Many to Many.

Chúng ta đã tạo một Entity Data Model cho CSDL SchoolDB trong phần Tạo Entity Data Model. Hình minh họa sau thể hiện visual designer cho EDM đó với tất cả các thực thể và quan hệ giữa chúng.



Hãy nhìn xem Entity Framework đã quản lý mỗi quan hệ như thế nào.

Quan hệ One-to-One:

Như bạn nhìn thấy trong hình minh họa ở trên, Student và StudentAddress có quan hệ One-to-One (không hoặc một). Một sinh viên có thể có chỉ một hoặc không có địa chỉ nào. Entity Framework thêm thuộc tính điều hướng Student vào thực thể StudentAddress và thuộc tính điều hướng StudentAddress vào thực thể Student. Cũng như thực thể StudentAddress có thuộc tính StudentId như là PrimaryKey điều đó làm nó thành một quan hệ One-to-One.

Đoạn mã ngắn sau thể hiện những lớp thực thể Student và StudentAddress.

```

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; }
}

```

Như bạn nhìn thấy trong đoạn mã trên, lớp thực thể Student có thuộc tính điều hướng StudentAddress và StudentAddress có thuộc tính điều hướng Student với thuộc tính khóa ngoại StudentId. Cách này EF kiểm soát quan hệ one-to-one giữa những thực thể.

Quan hệ One-to-Many:

Thực thể Standard và Teacher có một quan hệ One-to-Many đánh dấu bằng số bội mà ở đó 1 là dành cho One và * là cho many. Điều này có nghĩa rằng Standard có thể có nhiều Teachers trong khi Teacher có thể kết hợp với chỉ một Standard.

Để thể hiện điều này, thực thể Standard có tập hợp thuộc tính điều hướng Teachers (vui lòng chú ý nó là số nhiều) chỉ ra rằng một Standard có thể có một tập hợp Teachers (many Teachers). Và thực thể Teacher có chỉ một thuộc tính điều hướng Standard (không phải một tập hợp) chỉ ra rằng Teacher là kết hợp với chỉ một Standard. Nó cũng có khóa ngoại StandardId (StandardId là một PK trong thực thể Standard). Điều này biến nó thành quan hệ One-to-Many.

Đoạn mã bên dưới thể hiện lớp thực thể Standard và Teacher được tạo bởi EDM.

```
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; }
}
```

Như bạn nhìn thấy trong đoạn mã trên, lớp thực thể Standard có thuộc tính Teachers kiểu ICollection để nó có thể chứa nhiều đối tượng Teacher. (Nó khởi tạo thuộc tính Teachers với HashSet<Teacher> trong hàm tạo để bạn có thể thêm đối tượng Teacher vào tập hợp mà không phải lo lắng về việc khởi tạo nó.)

Lớp thực thể Teacher cũng có thuộc tính Standard với StandardId cho thuộc tính khóa ngoại. Entity Framework có thuộc tính khóa ngoại này bởi vì chúng ta đã check Include foreign key columns in the model checkbox trong EDM wizard khi tạo EDM trong phần Tạo Entity Data Model.

Quan hệ Many-to-Many:

Student và Course có quan hệ Many-to-Many đánh dấu bằng số bội *. Nó có nghĩa là một Student có thể đăng ký nhiều Courses và một Course cũng có thể dạy cho nhiều Students.

Thiết kế CSDL gồm bảng nối StudentCourse và nó có khóa chính của cả hai bảng (Student và Course). Entity Framework thể hiện quan hệ many-to-many không phải bằng cách tạo một tập thực thể cho bảng nối mà thay vì đó nó quản lý qua mapping.

Như bạn nhìn thấy ở hình trên, thực thể Student có thuộc tính Courses và thực thể Course có thuộc Students để thể hiện cho quan hệ many-to-many giữa chúng.

Đoạn mã sau trình bày lớp thực thể Student và Course

```
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 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; }
}
```

Note: Entity Framework hỗ trợ quan hệ many-to-many chỉ khi bảng nối (StudentCourse trong trường hợp này) không có bất kỳ cột nào khác ngoài khóa chính của hai bảng. Nếu bảng nối chứa những cột bổ sung như là DateCreated rồi EDM cũng tạo thực thể cho bảng trung gian và bạn sẽ phải quản lý thao tác CRUD cho những thực thể many-to-many bằng tay.

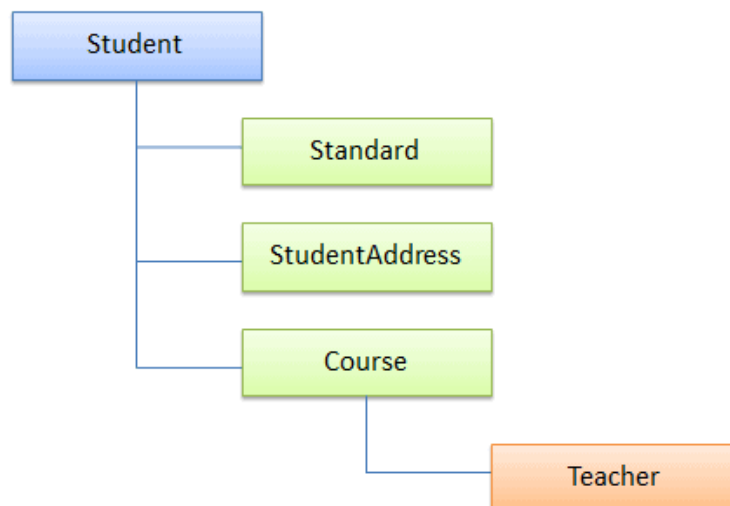
Mở EDM trong XML view. Bạn có thể nhìn thấy SSDL có tập thực thể StudentCourse nhưng CSDL thì không có tập thực thể này. Thay vì đó nó đang nối vào thuộc tính điều hướng của thực thể Student và Course. Trong MSL (C-S Mapping), nó có kết nối giữa Student và Course đưa vào bảng StudentCourse trong mục <AssociationSetMapping/>

```
SchoolDB.edmx
<?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="StudentAddress">...</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>
```

Như vậy quan hệ Many-to-Many đang được quản lý bởi C-S mapping trong EDM. Vậy thì khi bạn thêm một Student vào một Course hoặc một Course vào một thực thể Student và lưu nó, nó sẽ thêm khóa chính của sinh viên và khóa học mới được thêm vào bảng StudentCourse. Vậy mapping này không chỉ kích hoạt một liên kết thích hợp trực tiếp giữa hai thực thể mà còn quản lý truy vấn, inserts, và updates qua kết nối này.

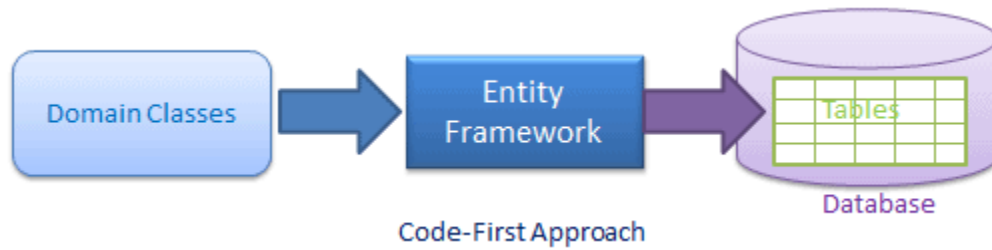
Entity Graph:

Khi một thực thể có một quan hệ với những thực thể khác thì hệ phân cấp đầy đủ các đối tượng được gọi là 'entity graph'. Ví dụ sau là một Student entity graph gồm phân cấp của thực thể Student với những thực thể Standard, StudentAddress và Course.



Student Entity Graph

Phần 8 – Các phương pháp phát triển với Entity Framework

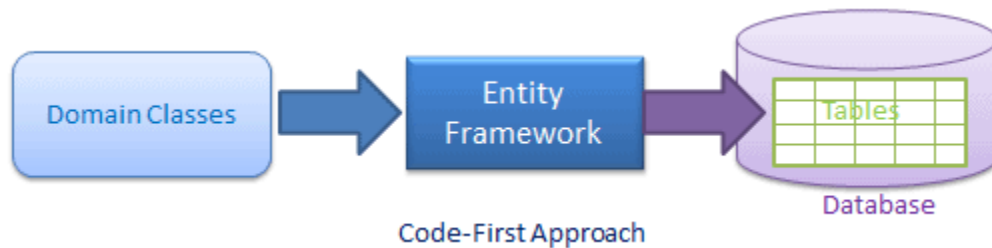


Entity Framework hỗ trợ 3 phương pháp tiếp cận phát triển khác nhau để sử dụng Entity Framework trong ứng dụng của bạn.

I. Code First:

Với phương pháp tiếp cận Code First bạn hoàn toàn tránh làm việc với visual model designer (EDMX). Bạn sẽ viết các lớp POCO của bạn trước sau đó tạo CSDL từ những lớp POCO này.

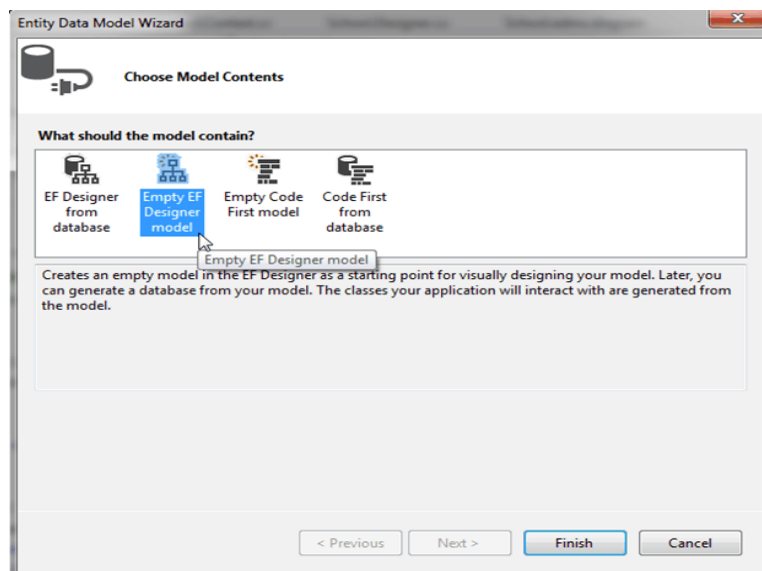
Các nhà phát triển theo con đường của nguyên lý Domain-Driven Design (DDD), thích bắt đầu bằng cách viết mã các lớp domain và sau đó khởi tạo CSDL yêu cầu để tiếp tục dữ liệu của họ.



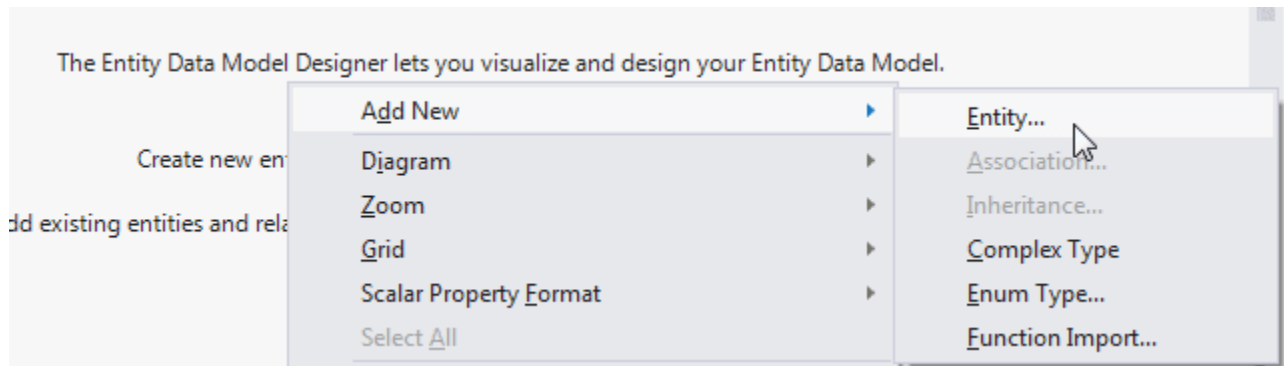
II. Model First:

Với phương pháp tiếp cận Model First, bạn tạo các Entities, quan hệ và những phân cấp kế thừa trực tiếp trên mặt thiết kế của EDMX và sau đó khởi tạo CSDL từ model của bạn.

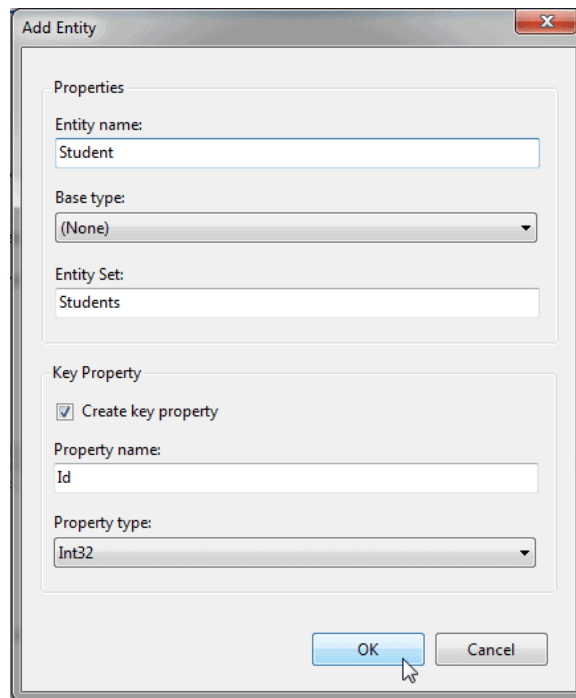
Như vậy với phương pháp tiếp cận Model First thêm mới ADO.NET Entity Data Model và chọn Empty EF Designer model trong Entity Data Model Wizard.



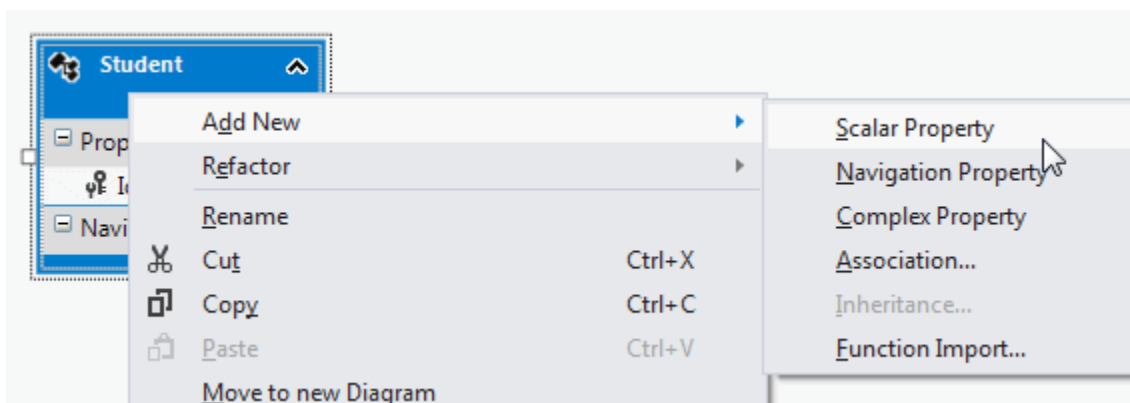
Bạn có thể tạo một thực thể, liên kết và kế thừa trên một vùng thiết kế trống bằng cách click chuột phải vào mặt thiết kế -> Add New -> Entity.



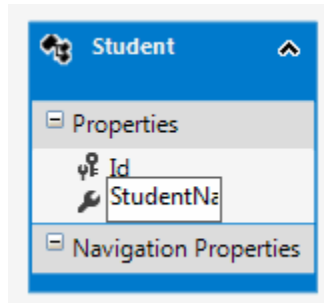
Ở hộp thoại Add Entity nhập tên của thực thể. Bạn cũng có thể đổi những cài đặt mặc định cho Entity Set and Key Property. Chúng ta sẽ giữ những cài đặt mặc định và click OK để khởi tạo một thực thể.



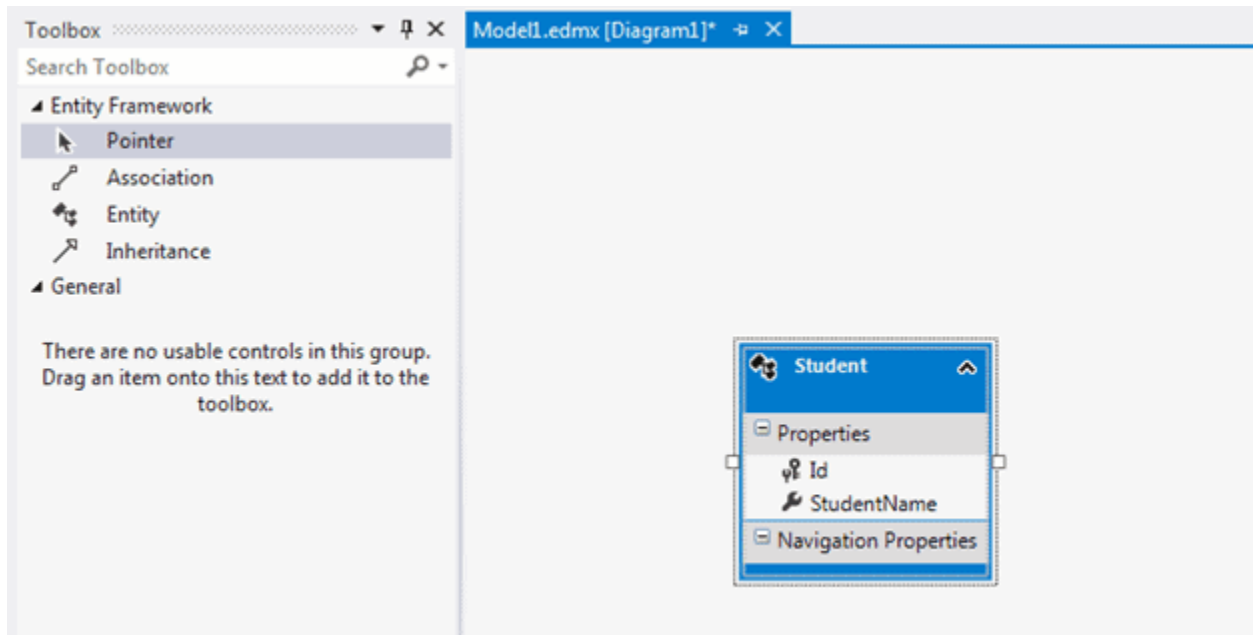
Bạn cũng có thể thêm những thuộc tính trong khi khởi tạo thực thể bằng cách click chuột phải trên thực thể -> Add New -> Scalar property.



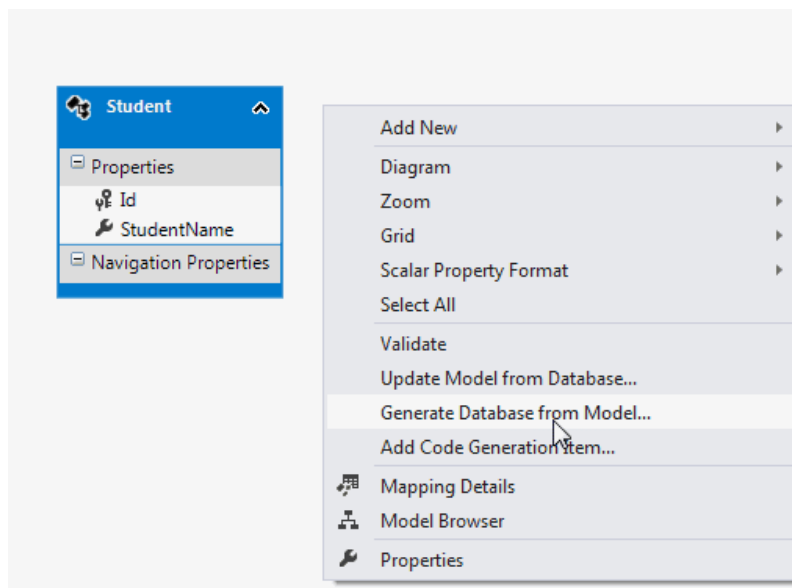
Nhập tên của thuộc tính scalar như bên dưới.



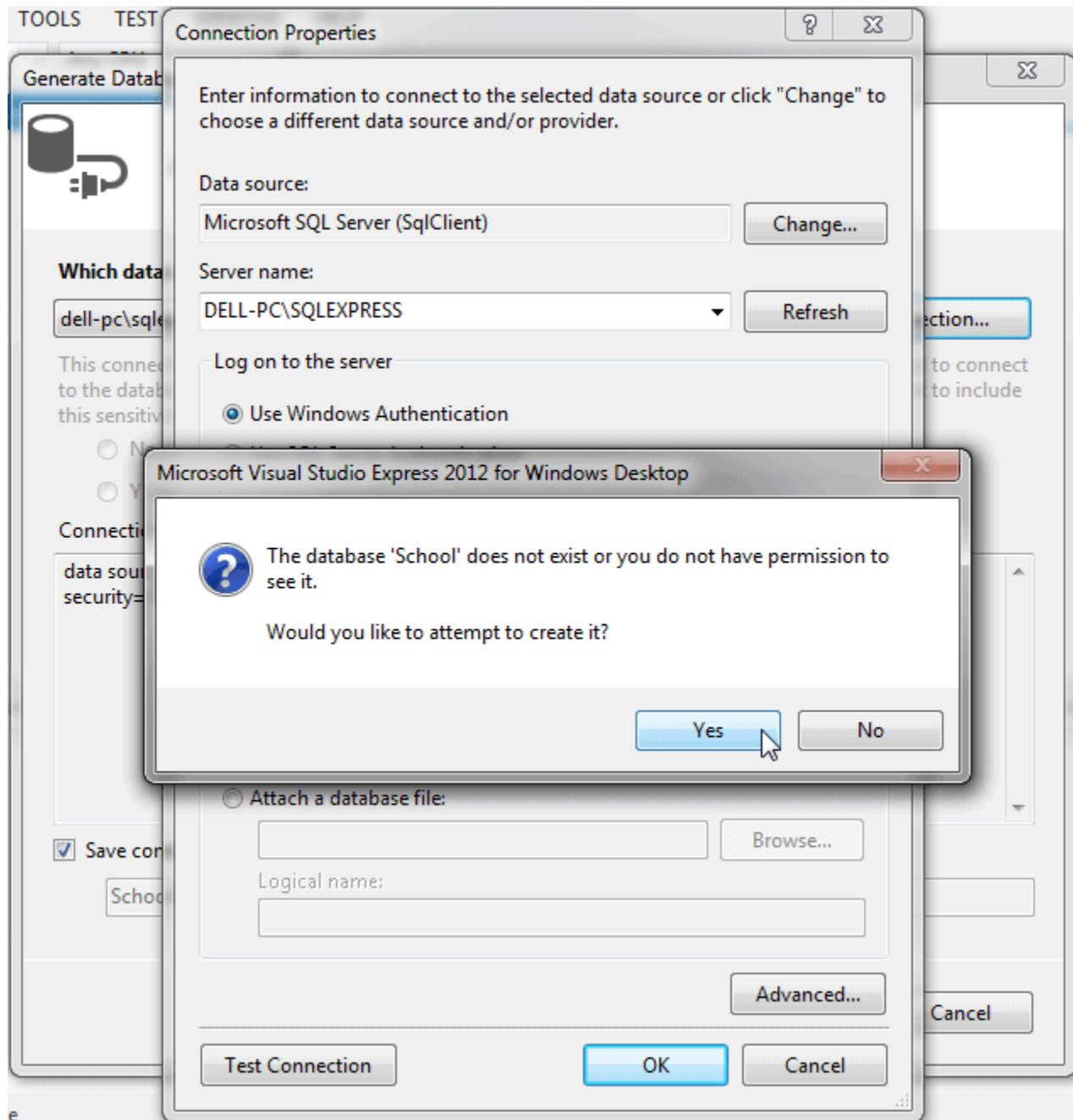
Với cách này bạn có thể thêm những thực thể và liên kết khác nhau sử dụng toolbox.



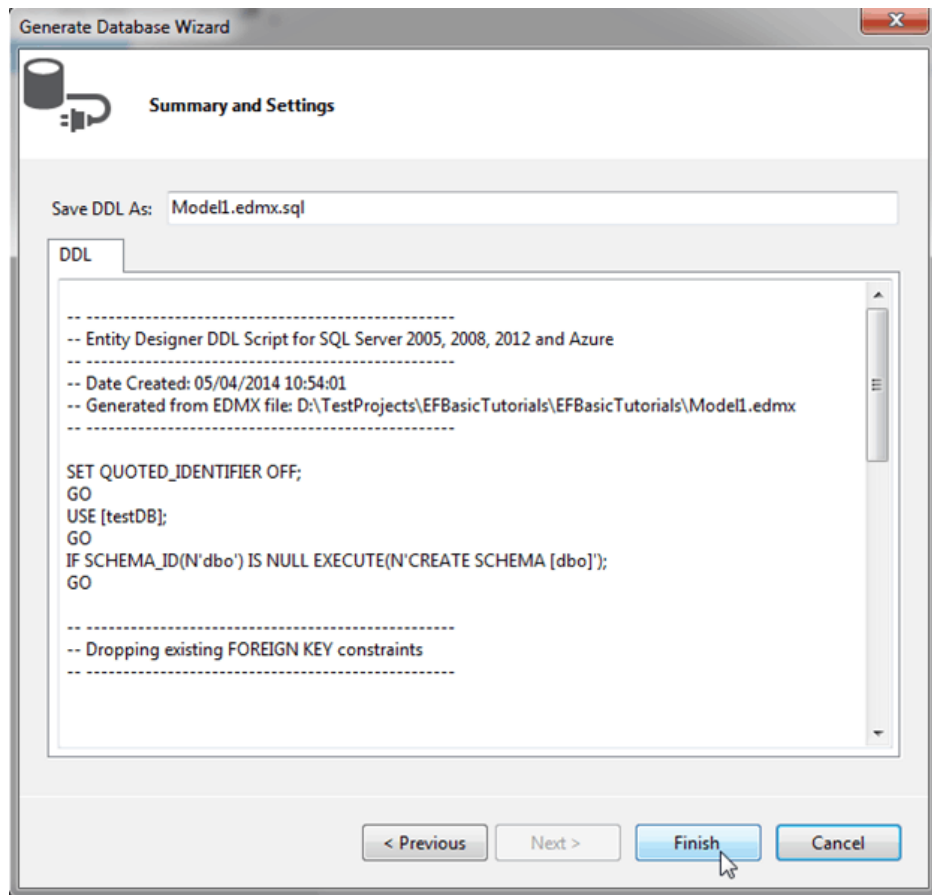
Sau khi tạo những thực thể, liên kết và thừa kế cần thiết ở mặt thiết kế của một model trống, bạn có thể sử dụng 'Generate database from model' trên menu ngữ cảnh của vùng thiết kế để khởi tạo DDL script.



Thao tác này sẽ mở Generate Database Wizard. Bạn có thể lựa chọn CSDL sẵn có hoặc tại một kết nối mới bằng cách click trên New Connection.. lựa chọn database server và nhập tên của CSDL để tạo rồi click OK. Nó sẽ hỏi bạn để xác nhận việc tạo một CSDL mới. Click Yes để tạo CSDL.

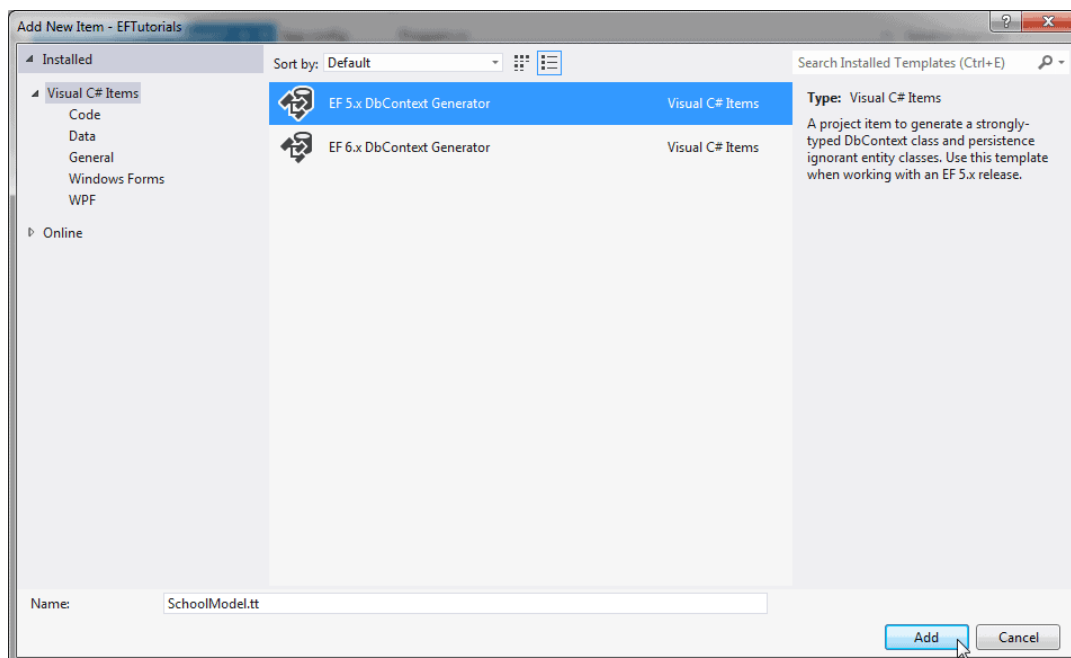


Click Next để khởi tạo DDL cho DB model như dưới đây.

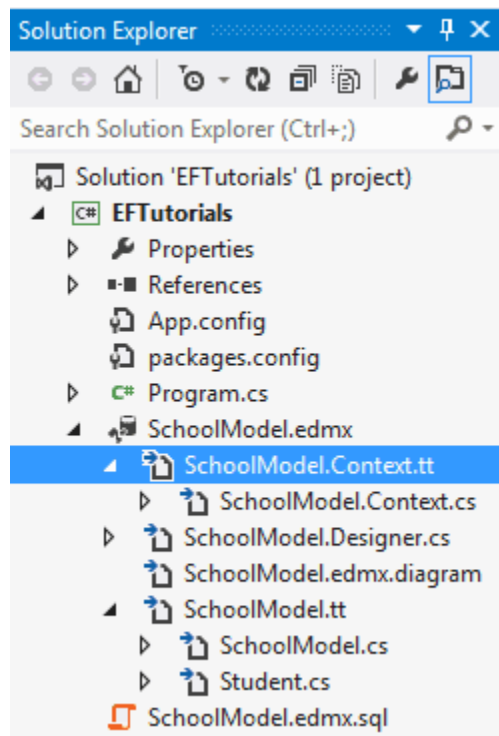


Nó sẽ thêm file ModelName.edmx.sql vào project của bạn. Bạn có thể thực thi DDL scripts trong Visual Studio bằng cách mở file .sql -> right click -> Execute.

Giờ bạn có thể khởi tạo lớp context và những thực thể bằng cách click chuột phải trên vùng thiết kế và chọn Add Code Generation Item..



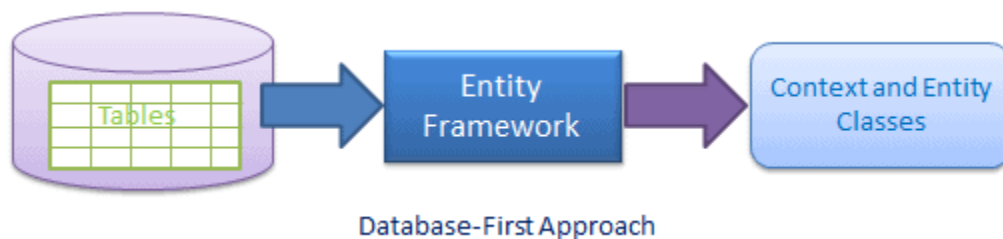
Nó sẽ thêm (ModelName).Context.tt và (ModelName).tt với lớp context và những lớp thực thể như bên dưới.



Như vậy theo cách này bạn có thể thiết kế DB model của bạn và sau đó khởi tạo CSDL và các lớp dựa trên model. Phương pháp này gọi là phương pháp tiếp cận Model-First/

III. Database First:

Chúng ta đã thấy phương pháp tiếp cận này trong phần Tạo Entity Data Model mà ở đó chúng ta tạo EDM, context và các lớp thực thể từ một CSDL sẵn có. Như vậy khi bạn khởi tạo EDMX từ một CSDL sẵn có thì nó là một phương pháp tiếp cận Database First.



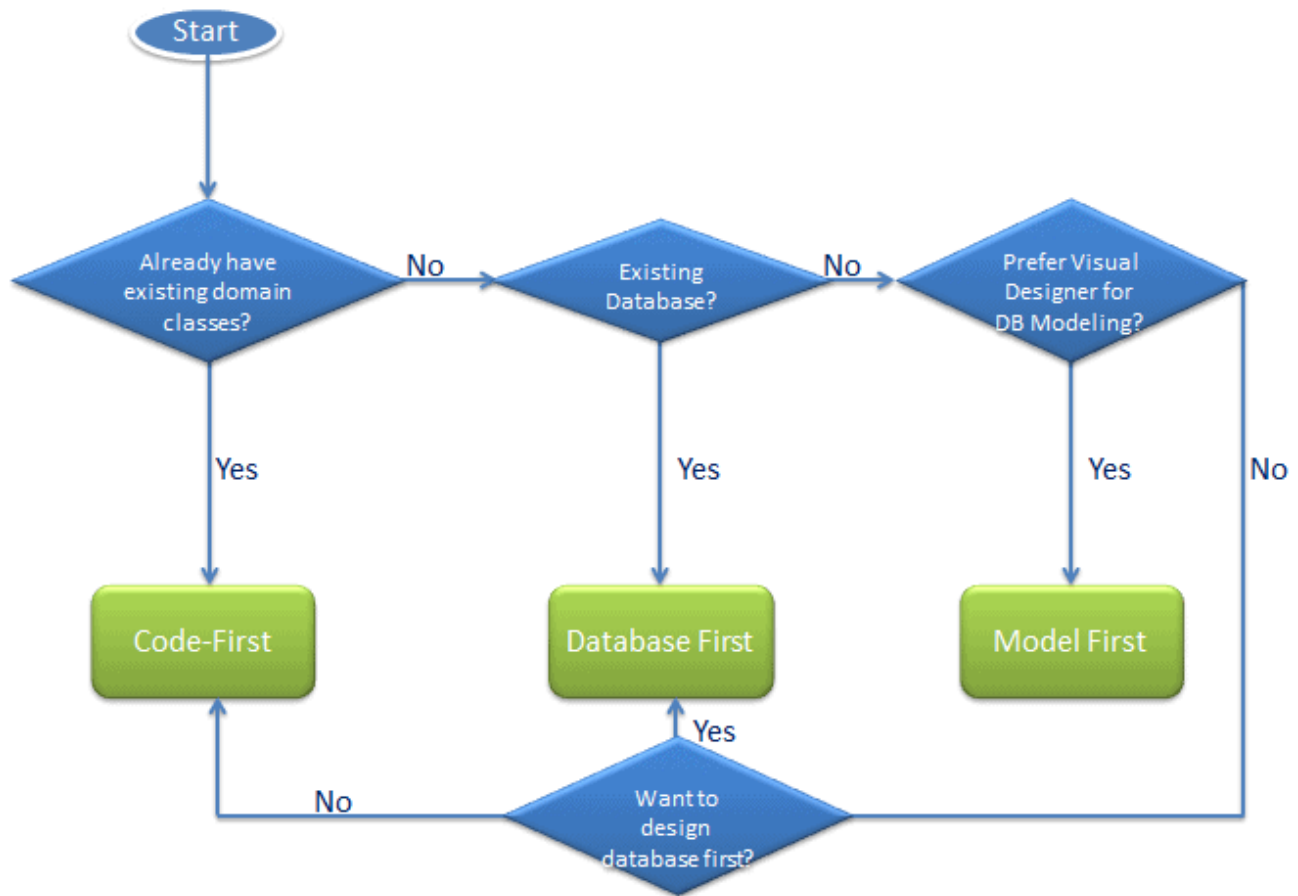
Entity Data Model có thể cập nhật bất cứ khi nào CSDL schema thay đổi. Phương pháp tiếp cận cũng hỗ trợ stored procedure, view, ...

Những bài hướng dẫn cơ bản này sẽ sử dụng phương pháp tiếp cận Database-First.

Chúng ta sẽ học cách làm thế nào để chọn phương pháp phát triển tiếp cận EF phù hợp trong phần tới.

IV. Chọn phương pháp tiếp cận với Entity Framework:

Chúng ta đã nhìn thấy những phương pháp tiếp cận ở những mục trên. Vậy giờ bạn phải quyết định sử dụng phương pháp tiếp cận nào cho ứng dụng của bạn. Hình sau minh họa cho cây quyết định



Như hình minh họa trên nếu bạn đã có sẵn một ứng dụng với những lớp domain thì bạn có thể sử dụng phương pháp tiếp cận Code-First bởi vì bạn có thể tạo một CSDL từ những lớp sẵn có trong phương pháp tiếp cận này. Nếu bạn đã có sẵn một CSDL thì bạn có thể tạo một EDM từ một CSDL có sẵn bằng phương pháp tiếp cận Database-First. Nếu bạn không có sẵn một CSDL hoặc những lớp domain thì bạn có thể chọn thiết kế DB model của bạn trên visual designer rồi tiếp tục bằng phương pháp tiếp cận Model-First.

Phần 9 – Truy vấn với EDM

```
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_6340CBDEAB68819DFA11CB43DA6I
        [1] { StudentName = "New Student", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6I
        [2] { StudentName = "New Student", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6I
        [3] { StudentName = "Bill", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94
        [4] { StudentName = "Bill", c = {System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94
    }
    Raw View
}
```

Chúng ta đã tạo EDM, DbContext và những lớp thực thể trong những phần trước. Bạn sẽ học những kiểu truy vấn khác nhau mà Entity Framework, nó lần lượt chuyển đổi thành truy vấn SQL cho CSDL bên dưới.

Entity Framework hỗ trợ 3 kiểu truy vấn: LINQ to Entities, Entity SQL và Native SQL.

I. LINQ to Entities:

Language-Integrated Query (LINQ) là một ngôn ngữ truy vấn mạnh mẽ được giới thiệu trong VS 2008. Bạn có thể sử dụng LINQ trong C# hoặc Visual Basic để truy vấn những dữ liệu nguồn khác nhau. LINQ-to-Entities hoạt động trên những thực thể Entity Framework để truy cập dữ liệu từ CSDL bên dưới. Bạn có thể sử dụng cú pháp LINQ method hoặc cú pháp truy vấn khi truy vấn với EDM.

Cú pháp LINQ Method:

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

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

Cú pháp LINQ Query:

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

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

Đầu tiên bạn phải tạo một đối tượng của lớp context đó là SchoolDBEntities. Bạn nên khởi tạo nó ở using() để một khi nó bên ngoài phạm vi thì nó sẽ tự động gọi phương thức Dispose() của DbContext. Cả hai cú pháp trên, context đều trả về IQueryable.

Những truy vấn Linq-to-Entities Projection:

Projection là một quá trình lựa chọn dữ liệu trong một dạng khác hơn là một dạng thực thể cụ thể đang truy vấn. Có nhiều cách của projection. Chúng ta giờ sẽ nhìn thấy vài kiểu của projection:

First/FirstOrDefault:

Nếu bạn muốn lấy một đối tượng sinh viên đơn khi có nhiều sinh viên, tên của đối tượng đó là “Student1” trong CSDL thì bạn sử dụng First hoặc FirstOrDefault như bên dưới:

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

Câu truy vấn trên sẽ thành truy vấn CSDL như sau:

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

Khác nhau giữa First và FirstOrDefault là First() sẽ ném ra một exception nếu không có dữ liệu kết quả cho tiêu thức đã cung cấp trong khi FirstOrDefault() trả về giá trị mặc định (null) nếu không có dữ liệu kết quả.

Single/SingleOrDefault:

Bạn cũng có thể sử dụng Single hoặc SingleOrDefault để lấy một đối tượng sinh viên đơn như thể hiện bên dưới:

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

Câu truy vấn trên sẽ thực thi truy vấn CSDL như sau:

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

Single hoặc SingleOrDefault sẽ ném ra một exception nếu kết quả trả về có nhiều hơn một phần tử. Sử dụng Single hoặc SingleOrDefault nếu bạn chắc chắn rằng kết quả trả về sẽ có chỉ một phần tử. Nếu kết quả có nhiều phần tử thì sẽ có vài vấn đề.

ToList:

Nếu bạn muốn liệt kê tất cả những sinh viên với tên 'Student1' (nếu có nhiều sinh viên trùng tên) thì sử dụng ToList():

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

Câu truy vấn trên sẽ thành truy vấn CSDL như sau:

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

GroupBy:

Nếu bạn muốn nhóm những sinh viên bằng StandardId thì sử dụng groupby:

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

Câu truy vấn trên sẽ thực thi truy vấn CSDL như sau:

```
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:

Nếu bạn muốn lấy danh sách của những sinh viên được sắp xếp theo StudentName thì sử dụng OrderBy:

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

Câu truy vấn trên sẽ thực thi truy vấn CSDL như sau:

```
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 Class result:

Nếu bạn muốn lấy chỉ StudentName, StandardName và danh sách của những khóa học trong một đối tượng đơn thì viết projection sau:

```
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
                           };
}
```

Câu truy vấn trên sẽ thực thi truy vấn CSDL như sau:

```
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]
go
```

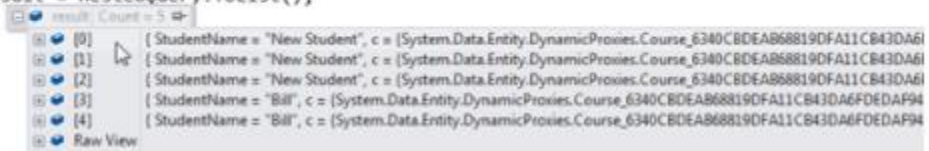
ProjectionResult trong truy vấn trên sẽ là kiểu nặc danh bởi vì không có lớp hoặc thực thể nào có những thuộc tính đó. Vì vậy trình biên dịch sẽ đánh dấu nó như là nặc danh.

Những truy vấn nested:

Bạn cũng có thể thực thi những truy vấn nested LINQ to entity như sau:

```
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_6340CBDEAB68819DFA11CB43DA6I) |
| [1] | New Student | (System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6I) |
| [2] | New Student | (System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6I) |
| [3] | Bill | (System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94) |
| [4] | Bill | (System.Data.Entity.DynamicProxies.Course_6340CBDEAB68819DFA11CB43DA6FDEDAF94) |

Truy vấn nested thể hiện ở trên sẽ kết quả trong một danh sách nặc danh với một đối tượng StudentName và Course.


```

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

```

Theo cách này bạn có thể làm một projection của kết quả. Theo cách bạn sẽ giống dữ liệu.

II. Entity SQL:

Entity SQL là cách khác để tạo một truy vấn. Nó được xử lý bởi Object Services của Entity Framework Object Services trực tiếp. Nó trả về ObjectQuery thay vì IQueryable.

Bạn cầnObjectContext để tạo một truy vấn sử dụng Entity SQL.

Đoạn mã sau thể hiện cùng một kết quả truy vấn như truy vấn L2E trên.

```

//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 IOObjectContextAdapter).ObjectContext;

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

```

Bạn cũng có thể sử dụng EntityConnection và EntityCommand để thực thi Entity SQL như thể hiện bên dưới:

```

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);
        }
    }
}

```

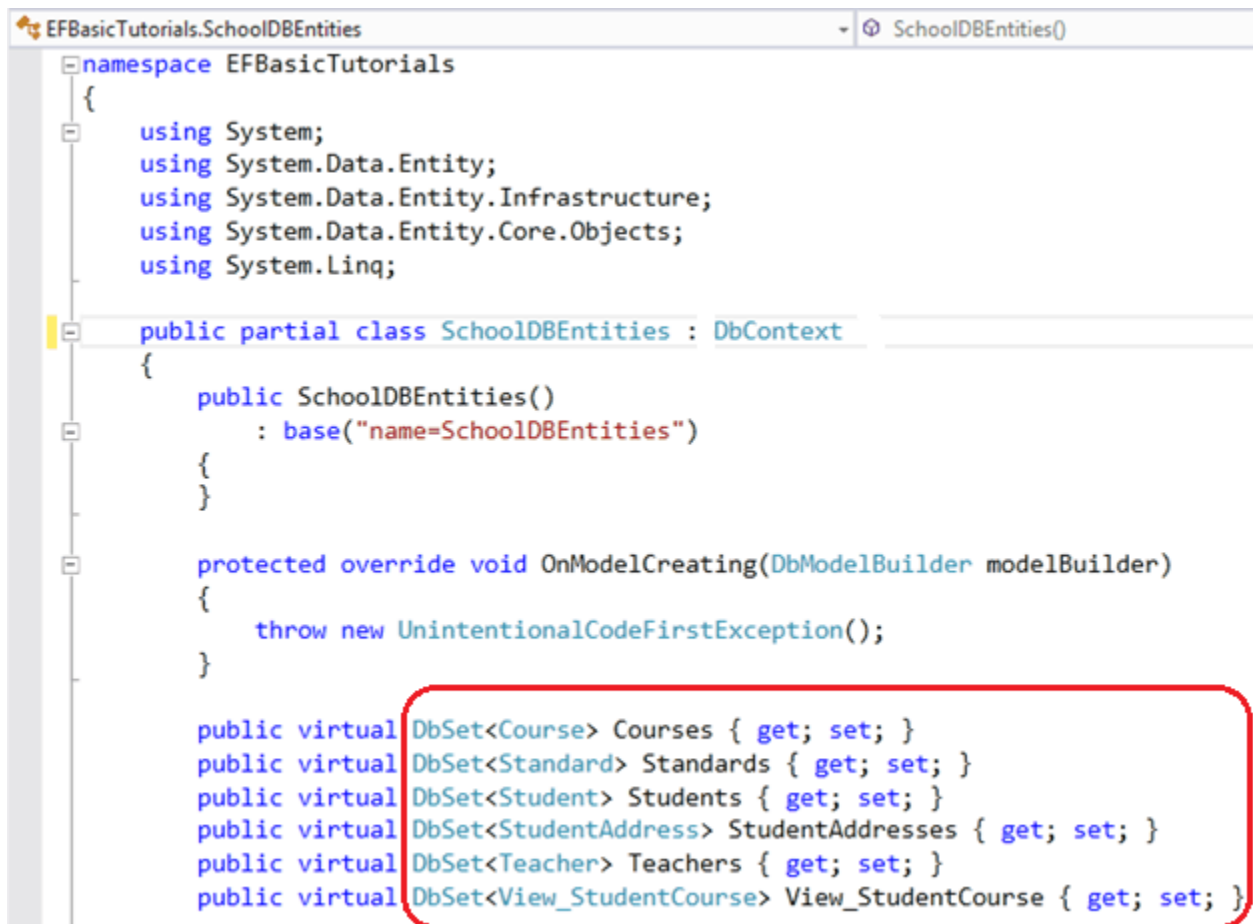
EntityDataReader không trả về ObjectQuery. Thay vì đó nó trả về dữ liệu theo dòng và cột.

Native SQL:

Bạn có thể thực thi những truy vấn native SQL cho một CSDL quan hệ như sau:

```
using (var ctx = new SchoolDBEntities())
{
    var studentName = ctx.Students.SqlQuery("Select studentid,
studentname, standardId from Student where
studentname='Bill'").FirstOrDefault<Student>();
}
```

Phần 10 – Lớp DbSet



Lớp DbSet thể hiện một tập thực thể được sử dụng để thao tác create, read, update, and delete. Một phiên bản chung của DbSet (DbSet<TEntity>) có thể sử dụng khi kiểu của thực thể không được biết ở thời điểm xây dựng.

Bạn có thể lấy tham chiếu của DbSet bằng cách sử dụng DbContext. VD: dbcontext.Students, dbcontext.Standards, hoặc bất kỳ tập thực thể khác. Lớp DbContext gồm DbSet như bên dưới:

```

EFBasicTutorials.SchoolDBEntities
SchoolDBEntities()

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; }
    }
}

```

Vài phương thức quan trọng của lớp DbSet được trình bày trong bảng sau:

| | | |
|-------------------------------------|-------------------|---|
| Add | Added entity type | <p>Thêm thực thể đã cho vào context với trạng thái Added. Khi những thay đổi được lưu lại, những thực thể ở trạng thái Added được insert vào CSDL. Sau khi những thay đổi được lưu, trạng thái đối tượng đổi thành Unchanged.</p> <p>VD: dbcontext.Students.Add(studentEntity)</p> |
| AsNoTracking<Entity>DBQuery<Entity> | | <p>Trả về một truy vấn mới mà ở đó các thực thể sẽ không được cached trong DbContext (Thừa kế từ DBQuery)</p> <p>Những thực thể trả về như AsNoTracking sẽ không được theo dõi bởi DbContext. Điều này sẽ làm tăng hiệu năng đáng kể cho những thực thể read only.</p> <p>VD: var studentList = dbcontext.Students.AsNoTracking<Student>().ToList<Student>();</p> |

| | | |
|----------------|--------------------------------------|--|
| Attach(Entity) | Entity which was passed as parameter | <p>Đính kèm thực thể đã cho vào context ở trạng thái Unchanged.</p> <p>VD: <code>dbContext.Students.Attach(studentEntity);</code></p> |
| Create | Entity | <p>Tạo một thể hiện mới của một thực thể cho kiểu của tập này. Thể hiện này không thêm hoặc đính kèm tới tập này được. Thể hiện trả về sẽ là một proxy nếu context bên dưới được cấu hình để tạo những proxy và kiểu thực thể đáp ứng được những yêu cầu tạo một proxy.</p> <p>Example: <code>var newStudentEntity = dbContext.Students.Create();</code></p> |
| Find(int) | Entity type | <p>Sử dụng giá trị khóa chính để thử tìm một thực thể được theo dõi bởi context. Nếu thực thể không ở trong context thì một truy vấn sẽ được thực thi và đánh giá theo dữ liệu trong data source và null được trả về nếu thực thể không được tìm thấy trong context hoặc trong data source. Lưu ý rằng Find cũng trả về những thực thể đã được thêm vào context nhưng chưa được lưu vào CSDL.</p> <p>VD: <code>Student studEntity = dbContext.Students.Find(1);</code></p> |
| Include | DBQuery | <p>Returns the included non generic LINQ to Entities query against a DbContext. (Inherited from DbQuery)</p> <p>Example: <code>var studentList = dbContext.Students.Include("StudentAddress").ToList<Student>();</code> <code>var studentList = dbContext.Students.Include(s => s.StudentAddress).ToList<Student>();</code></p> |
| Remove | Removed entity | <p>Đánh dấu thực thể chỉ ra như là Deleted. Khi những thay đổi được lưu, thực thể sẽ bị xóa từ CSDL. Thực thể phải tồn tại trong context ở một số trạng thái khác trước khi phương thức này được gọi.</p> <p>VD: <code>dbContext.Students.Remove(studentEntity);</code></p> |

SqlQuery

DBSqlQuery

Tạo ra một truy vấn raw SQL sẽ trả về những thực thể trong tập này. Mặc định những thực thể trả về được kiểm tra bởi context, điều này có thể thay đổi bằng cách gọi AsNoTracking trên DbSqlQuery<TEntity> trả về từ phương thức này.

VD : var studentEntity =
dbcontext.Students.SqlQuery("select * from student where
studentid = 1").FirstOrDefault<Student>();

Phần 11 – Lớp DBEntityEntry

DBEntityEntry là một lớp quan trọng, nó là hữu ích trong việc nhận những thông tin khác nhau về một thực thể. Bạn có thể nhận một thể hiện của DBEntityEntry của một thực thể riêng biệt bằng cách sử dụng phương thức Entry của DbContext.

VD: DBEntityEntry studentEntry = dbcontext.Entry(StudentEntity);

DBEntityEntry cho phép bạn truy cập trạng thái thực thể, giá trị hiện thời và nguyên bản của tất cả thuộc tính của thực thể đưa ra. Đoạn mã sau trình bày làm thế nào để nhận thông tin quan trọng của thực thể riêng biệt.

```
using (var dbCtx = new SchoolDBEntities())
{
    //get student whose StudentId is 1
    var student = dbCtx.Students.Find(1);

    //edit student name
    student.StudentName = "Edited name";

    //get DbEntityEntry object for student entity object
    var entry = dbCtx.Entry(student);

    //get entity information e.g. full name
    Console.WriteLine("Entity Name: {0}",
        entry.Entity.GetType().FullName);

    //get current EntityState
    Console.WriteLine("Entity State: {0}", entry.State );
```

```
Console.WriteLine("*****Property Values*****");

foreach (var propertyName in entry.CurrentValues.PropertyNames )
{
    Console.WriteLine("Property Name: {0}", propertyName);

    //get original value
    var orgVal = entry.OriginalValues[propertyName];
    Console.WriteLine("    Original Value: {0}", orgVal);

    //get current values
    var curVal = entry.CurrentValues[propertyName];
    Console.WriteLine("    Current Value: {0}", curVal);
}
}
```

Output:

Entity Name: Student

Entity State: Modified

*****Property Values*****

Property Name: StudentID

Original Value: 1

Current Value: 1

Property Name: StudentName

Original Value: First Student Name

Current Value: Edited name

Property Name: StandardId

Original Value:

Current Value:

DbEntityEntry cho phép bạn cài Added, Modified hoặc Deleted EntityState tới một thực thể như bên dưới:

```
context.Entry(student).State = System.Data.Entity.EntityState.Modified;
```

Lớp DbEntityEntry có những phương thức quan trọng sau:

| Tên phương thức | Kiểu trả về | Mô tả |
|-----------------|------------------------|--|
| Collection | DbCollectionEntry | <p>Nhận một đối tượng thể hiện tập hợp thuộc tính điều hướng từ thực thể này tới tập hợp của những thực thể liên quan.</p> <p>VD:</p> <pre>var studentDbEntityEntry = dbContext.Entry(studentEntity); var collectionProperty = studentDbEntityEntry.Collection<course>(s => s.Courses);</pre> |
| ComplexProperty | DbComplexPropertyEntry | <p>Nhận một đối tượng thể hiện thuộc tính phức tạp của một thực thể.</p> <p>VD:</p> <pre>var studentDbEntityEntry = dbContext.Entry(studentEntity); var complexProperty = studentDbEntityEntry.ComplexProperty(stud.StudentStandard);</pre> |

| | | |
|-------------------|------------------|---|
| | | <p>Những truy vấn CSDL cho việc sao chép dữ liệu của những thực thể được theo dõi cũng như chúng hiện đang tồn tại trong CSDL. Thay đổi những giá trị trong mảng trả về sẽ không cập nhật những giá trị trong CSDL. Nếu thực thể là không được tìm thấy trong CSDL thì nó sẽ trả về null.</p> |
| GetDatabaseValues | DBPropertyValues | <p>VD:</p> <pre>var studentDBEntityEntry = dbContext.Entry(studentEntity); var dbPropValues = studentDBEntityEntry.GetDatabaseValues();</pre> |
| Property | DBPropertyEntry | <p>Nhận một đối tượng thể hiện một thuộc tính scalar hoặc phức tạp của thực thể này.</p> <p>VD:</p> <pre>var studentDBEntityEntry = dbContext.Entry(studentEntity); string propertyName = studentDBEntityEntry.Property("StudentName").Name;</pre> |
| Reference | DBReferenceEntry | <p>Nhận một đối tượng thể hiện thuộc tính điều hướng tham chiếu từ thực thể này tới thực thể khác.</p> <p>VD:</p> <pre>var studentDBEntityEntry = dbContext.Entry(studentEntity); var referenceProperty = studentDBEntityEntry.Reference(s => s.Standard);</pre> |
| Reload | void | <p>Tải lại thực thể từ CSDL ghi đè bất kỳ giá trị thuộc tính nào với những giá trị từ CSDL. Thực thể này sẽ ở trong trạng thái Unchanged sau khi gọi phương thức này.</p> <p>VD:</p> <pre>var studentDBEntityEntry = dbContext.Entry(studentEntity); studentDBEntityEntry.Reload();</pre> |

Phần 12 – Change Tracking trong Entity Framework

Ở phần này bạn sẽ học được làm thế nào Entity Framework theo dõi những thay đổi trên thực thể suốt vòng đời của nó.

Entity Framework hỗ trợ tự động theo dõi thay đổi của những thực thể đã tải suốt vòng đời của context. Lớp `DbChangeTracker` đưa cho bạn tất cả thông tin về những thực thể hiện tại đang được theo dõi bởi context.

Vui lòng lưu ý rằng mỗi thực thể phải có thuộc tính EntityKey (khóa chính) để được theo dõi bởi context. Entity Framework sẽ không thêm bất kỳ thực thể nào trong conceptual model mà không có một thuộc tính EntityKey.

Đoạn mã sau trình bày làm thế nào lớp context theo dõi những thực thể và sự thay đổi xảy ra trong nó:

```
static void Main(string[] args)
{
    using (var ctx = new SchoolDBEntities())
    {

        Console.WriteLine("Find Student");
        var std1 = ctx.Students.Find(1);

        Console.WriteLine("Context tracking changes of {0} entity.",
ctx.ChangeTracker.Entries().Count());

        DisplayTrackedEntities(ctx.ChangeTracker);

        Console.WriteLine("Find Standard");

        var standard = ctx.Standards.Find(1);

        Console.WriteLine("Context tracking changes of {0} entities.",
ctx.ChangeTracker.Entries().Count());
        Console.WriteLine("");
        Console.WriteLine("Editing Standard");

        standard.StandardName = "Edited name";
        DisplayTrackedEntities(ctx.ChangeTracker);

    }
}
```

```
        Teacher tchr = new Teacher() { TeacherName = "new teacher" };
        Console.WriteLine("Adding New Teacher");

        ctx.Teachers.Add(tchr);
        Console.WriteLine("");
        Console.WriteLine("Context tracking changes of {0} entities.",
ctx.ChangeTracker.Entries().Count());
        DisplayTrackedEntities(ctx.ChangeTracker);

        Console.WriteLine("Remove Student");
        Console.WriteLine("");

        ctx.Students.Remove(std1);
        DisplayTrackedEntities(ctx.ChangeTracker);
    }
}
```

```

private static void DisplayTrackedEntities(DbChangeTracker
changeTracker)
{
    Console.WriteLine("");

    var entries = changeTracker.Entries();
    foreach (var entry in entries)
    {
        Console.WriteLine("Entity Name: {0}",
entry.Entity.GetType().FullName);
        Console.WriteLine("Status: {0}", entry.State);
    }
    Console.WriteLine("");
    Console.WriteLine("-----");
}

```

Output:

Find Student

Context tracking changes of 1 entity.

Entity Name: EFTutorials.Student

Status: Unchanged

Find Standard

Context tracking changes of 2 entities.

Editing Standard

Entity Name: EFTutorials.Standard

Status: Modified

Entity Name: EFTutorials.Student

Status: Unchanged

Adding New Teacher

Context tracking changes of 3 entities.

Entity Name: EFTutorials.Teacher

Status: Added

Entity Name: EFTutorials.Standard

Status: Modified

Entity Name: EFTutorials.Student

Status: Unchanged

Remove Student

Entity Name: EFTutorials.Teacher

Status: Added

Entity Name: EFTutorials.Standard

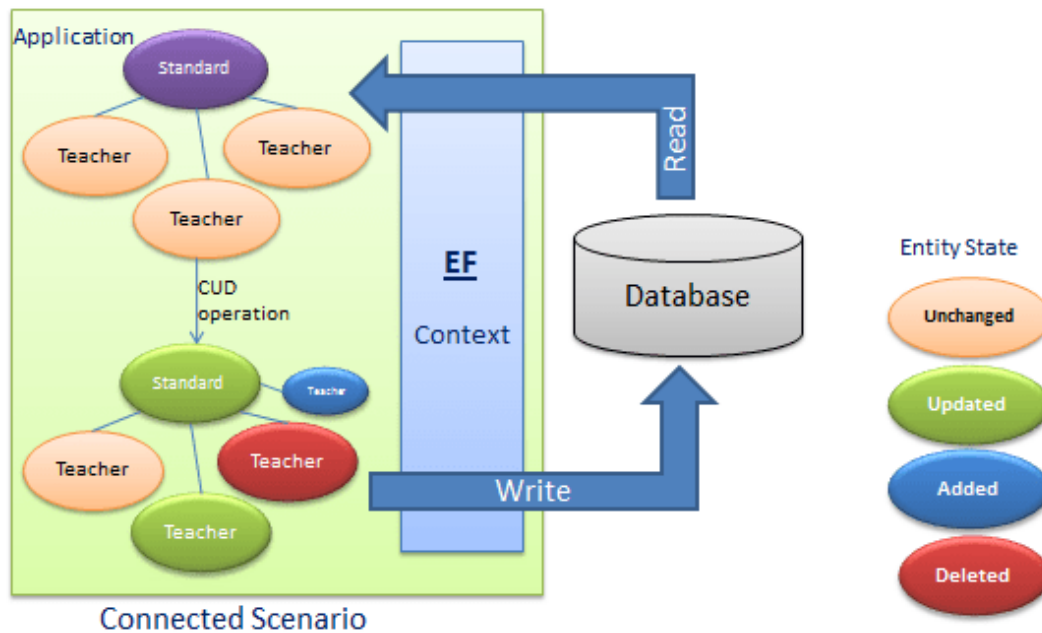
Status: Modified

Entity Name: EFTutorials.Student

Status: Deleted

Như bạn có thể nhìn thấy ở ví dụ trên và output, context giữ việc theo dõi những thực thể bất cứ khi nào chúng nhận, thêm, điều chỉnh hoặc xóa thực thể. Vui lòng lưu ý rằng context tồn tại suốt quá trình hoạt động trên những thực thể. Context sẽ không giữ theo dõi nếu bạn thực hiện bất kỳ thao tác nào trên thực thể bên ngoài phạm vi của nó.

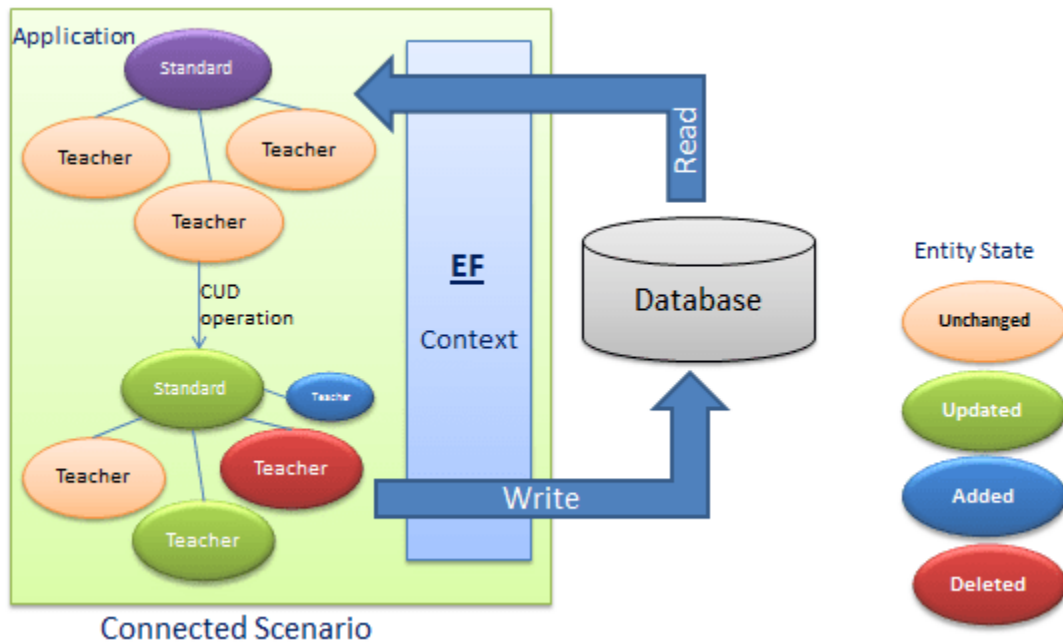
Phần 13 – Persistence trong Entity Framework



Có hai kiểu kịch bản khi làm bền vững một thực thể sử dụng EntityFramework: connected và disconnected scenarios

I. Connected Scenario

Là khi một thực thể được nhận từ CSDL và tính bền vững được sử dụng trong cùng context. Đối tượng Context không bị hủy bỏ giữa thực thể truy tìm và sự bền vững của những thực thể.



Thao tác CRUD trong Connected Scenario:

Là một tác vụ khá dễ bởi vì context theo dõi tự động những thay đổi đã xảy ra trong thực thể suốt vòng đời của nó, mặc định `AutoDetectChangesEnabled` đã cung cấp là `true`.

Chắc chắn rằng bạn đã tạo một Entity Data Model như hướng dẫn trong phần Tạo Entity Data Model cho CSDL mẫu SchoolDB.

Ví dụ sau chỉ cho bạn làm thế nào bạn có thể add, update và delete một thực thể trong kịch bản connected (trong phạm vi của context), nó sẽ lần lượt thực thi lệnh insert, update và delete trên CSDL. Context sẽ tự động phát hiện những thay đổi và cập nhật trạng thái của một thực thể.

```
using (var context = new SchoolDBEntities())
{
    var studentList = context.Students.ToList<Student>();

    //Perform create operation
    context.Students.Add(new Student() { StudentName = "New Student" });

    //Perform Update operation
    Student studentToUpdate = studentList.Where(s => s.StudentName ==
"student1").FirstOrDefault<Student>();
    studentToUpdate.StudentName = "Edited student1";

    //Perform delete operation
    context.Students.Remove(studentList.ElementAt<Student>(0));

    //Execute Inser, Update & Delete queries in the database
    context.SaveChanges();
}
```

Note: Nếu `context.Configuration.AutoDetectChangesEnabled = false` thì context không thể phát hiện những thay đổi đã thực hiện trên những thực thể hiện tại vì vậy không thực thi truy vấn update. Bạn cần gọi `context.ChangeTracker.DetectChanges()` trước `SaveChanges()` để phát hiện những thực thể có chỉnh sửa và đánh dấu trạng thái của chúng như `Modified`

Context phát hiện những thực thể thêm và xóa khi thao tác được thực hiện chỉ trên `DbSet`. Nếu bạn thực hiện thêm và xóa thực thể trên một tập hợp hoặc danh sách riêng biệt thì nó sẽ không phát hiện được những thay đổi này.

Đoạn mã sau sẽ không thêm hoặc xóa sinh viên. Nó chỉ cập nhật thực thể sinh viên bởi vì chúng ta đang thêm và xóa những thực thể từ `List` mà không phải từ `DbSet`.

```
using (var context = new SchoolDBEntities())
{
    var studentList = context.Students.ToList<Student>();

    //Add student in list
    studentList.Add(new Student() { StudentName = "New Student" });

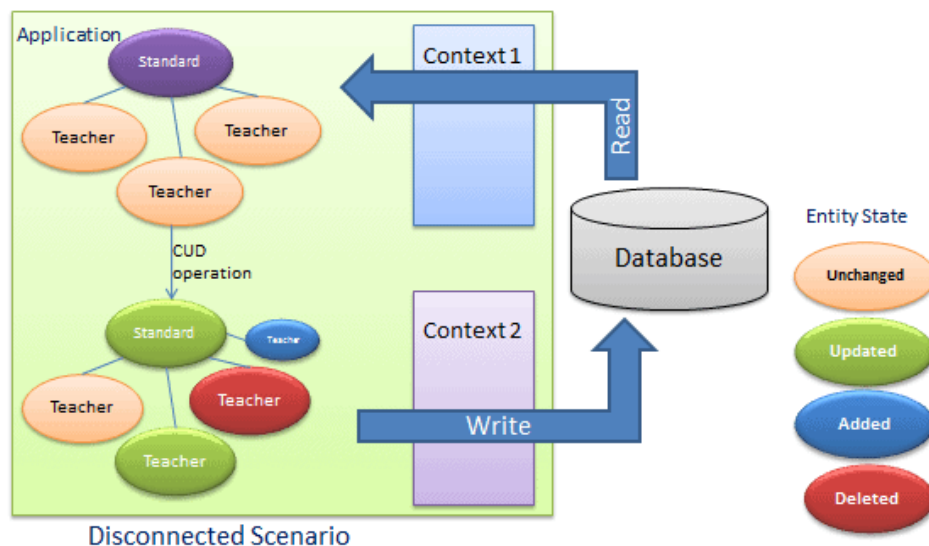
    //Perform update operation
    Student studentToUpdate = studentList.Where(s => s.StudentName ==
    "Student1").FirstOrDefault<Student>();
    studentToUpdate.StudentName = "Edited student1";

    //Delete student from list
    if (studentList.Count > 0)
        studentList.Remove(studentList.ElementAt<Student>(0));

    //SaveChanges will only do update operation not add and delete
    context.SaveChanges();
}
```

II. Disconnected Scenario

Là khi một thực thể được nhận từ CSDL và những thực thể đã thay đổi được submit sử dụng những đối tượng khác nhau trong context. Ví dụ sau minh họa kịch bản disconnected:

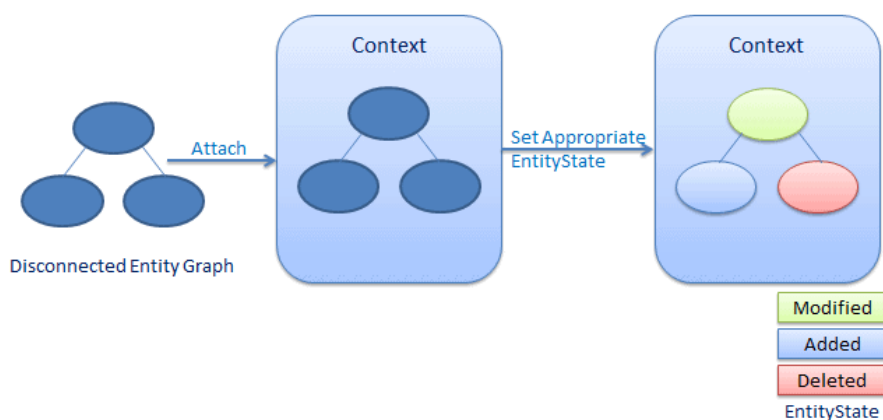


Với mỗi kịch bản trên, Context1 là sử dụng cho thao tác read và sau đó Context1 bị hủy bỏ. Một khi những thực thể thay đổi, ứng dụng submit những thực thể sử dụng Context2 – một đối tượng context khác.

Kịch bản disconnected là khá phức tạp bởi vì context mới không biết bất kỳ điều gì về thực thể đã thay đổi vì vậy bạn sẽ phải chỉ dẫn cho context những gì đã thay đổi trong thực thể. Trong hình minh họa bên dưới, ứng dụng nhận một entity graph sử dụng Context 1 và sau đó ứng dụng thực hiện vài thao tác CRUD (Create, Update, Delete) trên nó và cuối cùng nó lưu trữ entity graph sử dụng Context 2. Context 2 không biết thao tác gì đã thực hiện trên entity graph trong kịch bản này.

Bạn sẽ học được làm thế nào để thực hiện thao tác CRUD trong kịch bản disconnected trong phần tiếp theo.

Phần 14 – Disconnected Scenario #1

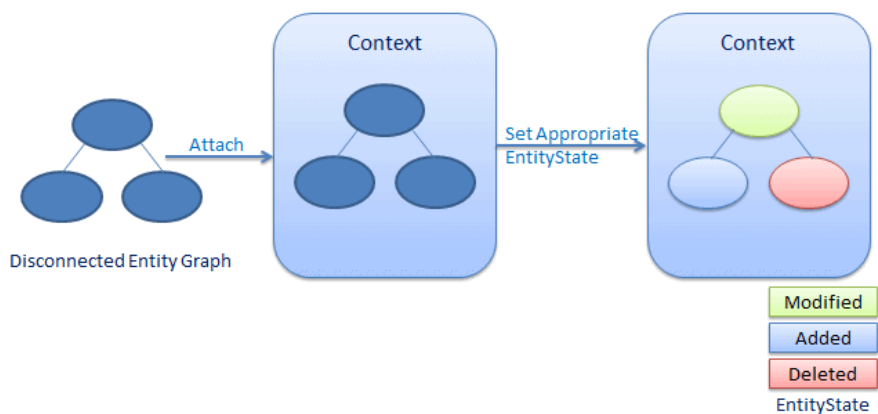


Disconnected Entities:

Phản trước chúng ta đã thấy làm thế nào để thực hiện thao tác CRUD trên disconnected entity graph, chúng ta hãy xem làm thế nào để liên kết disconnected entity graph với thể hiện mới của context.

Có hai thứ chúng ta cần làm khi lấy về một disconnected entity graph hoặc thậm chí một thực thể disconnected đơn. Đầu tiên chúng ta cần gắn những thực thể với context instance mới và làm cho context nhận biết về những thực thể này. Thứ hai cài đặt EntityStates thích hợp với những thực thể bằng tay bởi vì context instance mới không biết bất cứ điều gì về những thao tác đã thực hiện trên những thực thể disconnected vì vậy context mới không thể áp dụng EntityState thích hợp.

Hình sau minh họa cho tiến trình này.



Entity Framework API cung cấp những phương thức quan trọng để gắn những thực thể disconnected tới context mới và cài đặt EntityState tới tất cả những thực thể của một entity graph.

DbSet.Add():

Phương thức DbSet.Add() gắn toàn bộ entity graph tới context mới và tự động áp dụng Added entity state tới tất cả những thực thể.

Xem xét đoạn mã sau.

```
//disconnected entity graph
Student disconnectedStudent = new Student() { StudentName = "New
Student" };
disconnectedStudent.StudentAddress = new StudentAddress() { Address1
= "Address", City = "City1" };

using (var ctx = new SchoolDBEntities())
{
    //add disconnected Student entity graph to new context instance
    - ctx
    ctx.Students.Add(disconnectedStudent);

    // get DbEntityEntry instance to check the EntityState of
    specified entity
    var studentEntry = ctx.Entry(disconnectedStudent);
    var addressEntry =
    ctx.Entry(disconnectedStudent.StudentAddress);

    Console.WriteLine("Student EntityState:
    {0}",studentEntry.State);

    Console.WriteLine("StudentAddress EntityState:
    {0}",addressEntry.State);
}
```

Output:

Student EntityState: Added

StudentAddress EntityState: Added

Như đoạn mã mẫu trên đây, chúng ta thêm disconnectedStudent entity graph sử dụng phương thức ctx.Students.Add. Ở đây thực thể cha là Student vì vậy chúng ta đã thêm toàn bộ entity graph trong Students DbSet. Chúng ta rồi sau đó nhận thể hiện của DbEntityEntry cho những thực thể Student và StudentAddress để kiểm tra trạng thái của mỗi thực thể. Như bạn thấy trong phần output cả hai thực thể có trạng thái Added.

Như vậy sử dụng phương thức Add của thực thể cha DbSet để gắn toàn bộ entity graph tới context instance mới với trạng thái Added cho mỗi thực thể. Phương thức này sẽ thực thi lệnh insert cho tất cả những thực thể và sẽ insert những dòng mới trong những bảng CSDL thích hợp.

DbSet.Attach():

Phương thức DbSet.Attach gắn toàn bộ entity graph tới context mới với trạng thái thực thể Unchanged.

Xem xét đoạn mã sau.

```
//disconnected entity graph
Student disconnectedStudent = new Student() { StudentName = "New
Student" };
disconnectedStudent.StudentAddress = new StudentAddress() { Address1
= "Address", City = "City1" };

using (var ctx = new SchoolDBEntities())
{
    //attach disconnected Student entity graph to new context
instance - ctx
    ctx.Students.Attach(disconnectedStudent);

    // get DbEntityEntry instance to check the EntityState of
specified entity
    var studentEntry = ctx.Entry(disconnectedStudent);
    var addressEntry =
ctx.Entry(disconnectedStudent.StudentAddress);

    Console.WriteLine("Student EntityState:
{0}",studentEntry.State);

    Console.WriteLine("StudentAddress EntityState:
{0}",addressEntry.State);
}
```

Output:

Student EntityState: Unchanged

StudentAddress EntityState: Unchanged

Như đoạn mã ở trên chúng ta có thể gắn disconnected entity graph sử dụng phương thức DbSet.Attach. Điều này sẽ gắn toàn bộ entity graph tới context mới với trạng thái thực thể Unchanged tới tất cả những thực thể.

Như vậy phương thức Attach sẽ chỉ gắn entity graph tới context vì vậy chúng ta cần tìm trạng thái thực thể thích hợp cho mỗi thực thể và áp dụng nó thủ công.

DbContext.Entry():

Phương thức Entry of DbContext trả về thể hiện của DbEntityEntry cho một thực thể cụ thể. DbEntityEntry có thể sử dụng để thay đổi trạng thái của một thực thể.

```
DbContext.Entry(disconnectedEntity).state =  
EntityState.Added/Modified/Deleted/Unchanged
```

Phương thức này gắn toàn bộ entity graph tới context với trạng thái cụ thể tới thực thể cha và cài đặt trạng thái của những thực thể khác như trình bày trong bảng sau.

| Parent Entity State | Entity State of child entities |
|---------------------|--------------------------------|
| Added | Added |
| Modified | Unchanged |
| Deleted | Tất cả thực thể con sẽ là null |

Phần 15 – Disconnected Scenario #2

I. Thêm thực thể mới sử dụng DbContext trong Disconnected Scenario

Trong mục này bạn sẽ học được làm thế nào để thêm một thực thể mới trong DbContext trong disconnected scenario, lần lượt insert một dòng mới trong bảng CSDL.

Nếu bạn sử dụng phương thức tiếp cận Database-First sau đó tạo một Entity Data Model như trình bày trong phần Tạo Entity Data Model cho CSDL mẫu SchoolDB. Hoặc nếu bạn sử dụng phương thức tiếp cận Code-First hoặc Model-First sau đó tạo những thực thể và những lớp context. Trong bất kỳ trường hợp nào những thực thể và những lớp context đều sẽ tương tự

Ở đây chúng ta sẽ thấy làm thế nào để thêm một thực thể đơn Student (không phải entity graph) . Sau đây là một thực thể Student.

```
using System;  
using System.Collections.Generic;  
  
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; }  
}
```

Đây là lớp context.

```
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)
    {
    }

    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; }
}
```

Đoạn mã sau trình bày làm thế nào để lưu một thực thể đơn.

```
class Program
{
    static void Main(string[] args)
    {
        // create new Student entity object in disconnected scenario
        (out of the scope of DbContext)
        var newStudent = new Student();

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

        //create DbContext object
        using (var dbContext = new SchoolDBEntities())
        {
            //Add Student object into Students DbSet
            dbContext.Students.Add(newStudent);

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

Như bạn có thể thấy trong đoạn mã trên, đầu tiên chúng ta đã tạo một đối tượng thực thể mới Student và cài đặt StudentName là 'Bill'. Thứ hai chúng ta đã tạo một đối tượng DbContext mới và thêm newStudent vào Students EntityState. Thứ ba chúng ta gọi phương thức SaveChanges của DbContext và nó sẽ thực thi truy vấn insert sau tới CSDL.

```
exec sp_executesql N'INSERT [dbo].[Student]([StudentName], [StandardId])
VALUES (@@, NULL)
SELECT [StudentID], [RowVersion]
FROM [dbo].[Student]
WHERE @@ROWCOUNT > 0 AND [StudentID] = scope_identity(),@@='Bill'
```

Ngoài ra chúng ta cũng có thể thêm thực thể vào DbContext.Entry và đánh dấu nó như Added và nó có kết quả truy vấn insert tương tự:

```
class Program
{
    static void Main(string[] args)
    {
        // create new Student entity object in disconnected scenario
        (out of the scope of DbContext)
        var newStudent = new Student();

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

        //create DbContext object
        using (var dbContext = new SchoolDBEntities())
        {
            //Add newStudent entity into DbSet and mark
            EntityState to Added
            dbContext.Entry(newStudent).State =
            System.Data.Entity.EntityState.Added;

            // call SaveChanges method to save new Student into database
            dbContext.SaveChanges();
        }
    }
}
```

Vì vậy theo cách này bạn có thể thêm một thực thể đơn mới trong disconnected scenario.

II. Cập nhật thực thể đã tồn tại sử dụng DbContext trong Disconnected Scenario

Chúng ta sẽ thấy làm thế nào để cập nhật một thực thể Student đơn (không phải entity graph). Sau đây là một thực thể Student.

```
using System;
using System.Collections.Generic;

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; }
}
```

Đây là một lớp context.

```
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)
    {
    }

    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; }
}
```

Ví dụ sau trình bày làm thế nào để cập nhật một thực thể Student trong disconnected scenario:

```
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 dbContext = new SchoolDBEntities())
{
    //3. Mark entity as modified
    dbContext.Entry(stud).State = System.Data.Entity.EntityState.Modified;

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

Như bạn nhìn thấy trong đoạn mã trên, we đang làm những bước sau:

1. Lấy một sinh viên đã tồn tại từ CSDL.
2. Thay đổi tên sinh viên bên ngoài phạm vi của Context (mô hình disconnected).
3. Chuyển thực thể đã chỉnh sửa vào phương thức Entry để lấy đối tượng DBEntityEntry của nó và sau đó đánh dấu trạng thái của nó là Modified.
4. Gọi phương thức SaveChanges() để cập nhật thông tin sinh viên vào CSDL.

SaveChanges sẽ gửi truy vấn update sau tới CSDL:

```
exec sp_executesql N'update [dbo].[Student]
set [StudentName] = @0, [StandardId] = @1
where ([StudentID] = @2)',N'@0 varchar(50),
@1 int,@2 int',@0='Updated Student1',@1=299,@2=267
```

Theo cách này chúng ta có thể dễ dàng cập nhật một thực thể đơn sử dụng DbContext trong mô hình disconnected.

Phương thức DbContext.Entry trả về một thể hiện của DBEntityEntry cho một thực thể cụ thể. Một thể hiện của lớp DbEntityEntry cung cấp một truy cập tới thông tin về một thực thể đã cho và trạng thái của nó. Bạn có thể thay đổi trạng thái của một thực thể thành Added, Updated hoặc Deleted.

III. Xóa thực thể sử dụng DbContext trong Disconnected Scenario

Chúng ta sử dụng phương thức Entry của DbContext để đánh dấu EntityState là Modified ở mục trên. Theo cách này chúng ta có thể sử dụng phương thức Entry để gắn một thực thể disconnected tới context và đánh dấu trạng thái của nó thành Deleted.

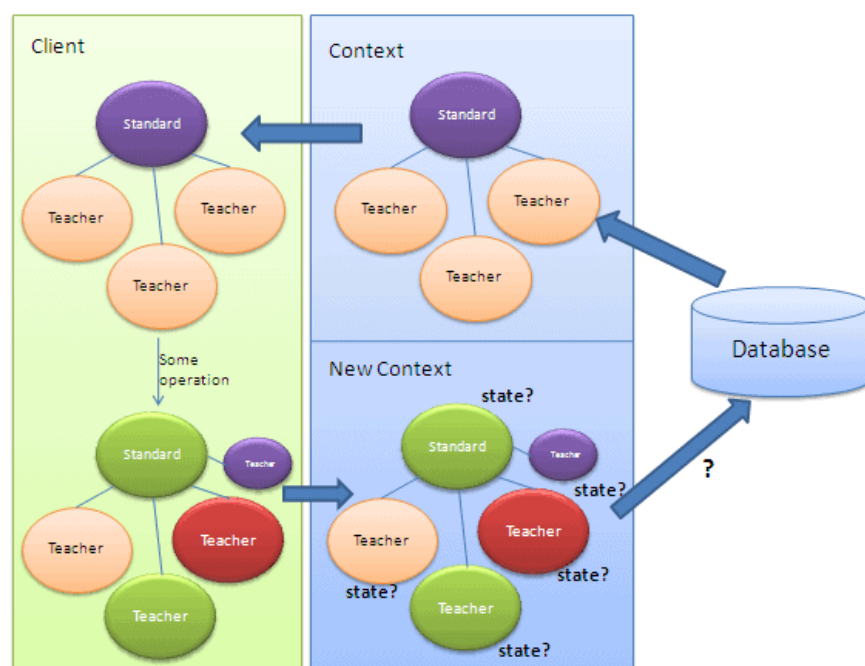
```
Student studentToDelete;  
//1. Get student from DB  
using (var ctx = new SchoolDBEntities())  
{  
    studentToDelete = ctx.Students.Where(s => s.StudentName ==  
    "Student1").FirstOrDefault<Student>();  
}  
  
//Create new context for disconnected scenario  
using (var newContext = new SchoolDBEntities())  
{  
    newContext.Entry(studentToDelete).State =  
    System.Data.Entity.EntityState.Deleted;  
  
    newContext.SaveChanges();  
}
```

Đoạn mã trình bày ở trên có kết quả trong truy vấn delete dòng từ bảng Teacher.

```
delete [dbo].[Student]  
where ([StudentId] = @0)',N'@0 int',@0=1
```

Như vậy bạn có thể xóa một thực thể đơn trong kịch bản disconnected.

Phần 16 – Disconnected Scenario #3



Entity Graph in Disconnected Scenario

I. Thêm Entity Graph sử dụng DbContext

Adding entity graph with all new entities is a simple task. We can use `DbSet.Add()` method to attach a whole entity graph to the context and set all the entity's states to Added as we have seen in the previous section.

Thêm entity graph với tất cả những thực thể mới là một tác vụ đơn giản. Chúng ta có thể sử dụng phương thức `DbSet.Add()` để gắn toàn bộ entity graph vào context và cài đặt trạng thái của thực thể thành Added như chúng ta đã nhìn thấy trong phần trước.

Ví dụ sau thêm một entity graph sinh viên trong mô hình disconnected sử dụng phương thức `DbSet.Add()` và nó đánh dấu trạng thái Added tới tất cả những thực thể:

```
//Create student in disconnected mode
Student newStudent = new Student() { StudentName = "New Single Student"
};

//Assign new standard to student entity
newStudent.Standard = new Standard() { StandardName = "New Standard" };

//add new course with new teacher into student.courses
newStudent.Courses.Add(new Course() { CourseName = "New Course for
single student", Teacher = new Teacher() { TeacherName = "New Teacher" }
});

using (var context = new SchoolDBEntities())
{
    context.Students.Add(newStudent);
    context.SaveChanges();

    Console.WriteLine("New Student Entity has been added with new
StudentId= " + newStudent.StudentID.ToString());
    Console.WriteLine("New Standard Entity has been added with new
StandardId= " + newStudent.StandardId.ToString());
    Console.WriteLine("New Course Entity has been added with new
CourseId= " + newStudent.Courses.ElementAt(0).CourseId.ToString());
    Console.WriteLine("New Teacher Entity has been added with new
TeacherId= " + newStudent.Courses.ElementAt(0).TeacherId.ToString());
}
```

Output:

New Student Entity has been added with new StudentId= 14

New Standard Entity has been added with new StandardId= 6

New Course Entity has been added with new CourseId= 7

New Teacher Entity has been added with new TeacherId= 9

Và thực thi những câu lệnh sau tới CSDL:

```

exec sp_executesql N'INSERT [dbo].[Standard]([StandardName],
[Description])
VALUES (@0, NULL)
SELECT [StandardId]
FROM [dbo].[Standard]
WHERE @@ROWCOUNT > 0 AND [StandardId] = scope_identity()',N'@0
varchar(50)',@0='New Standard'
go

exec sp_executesql N'INSERT [dbo].[Student]([StudentName], [StandardId])
VALUES (@0, @1)
SELECT [StudentID]
FROM [dbo].[Student]
WHERE @@ROWCOUNT > 0 AND [StudentID] = scope_identity()',N'@0
varchar(50),@1 int',@0='New Single Student',@1=61
go

```

```

exec sp_executesql N'INSERT [dbo].[Teacher]([TeacherName], [StandardId])
VALUES (@0, NULL)
SELECT [TeacherId]
FROM [dbo].[Teacher]
WHERE @@ROWCOUNT > 0 AND [TeacherId] = scope_identity()',N'@0
varchar(50)',@0='New Teacher'
go

exec sp_executesql N'INSERT [dbo].[Course]([CourseName], [Location],
[TeacherId])
VALUES (@0, NULL, @1)
SELECT [CourseId]
FROM [dbo].[Course]
WHERE @@ROWCOUNT > 0 AND [CourseId] = scope_identity()',N'@0
varchar(50),@1 int',@0='New Course for single student',@1=93
go

exec sp_executesql N'INSERT [dbo].[StudentCourse]([StudentId],
[CourseId])
VALUES (@0, @1)
',N'@0 int,@1 int',@0=43,@1=72
go

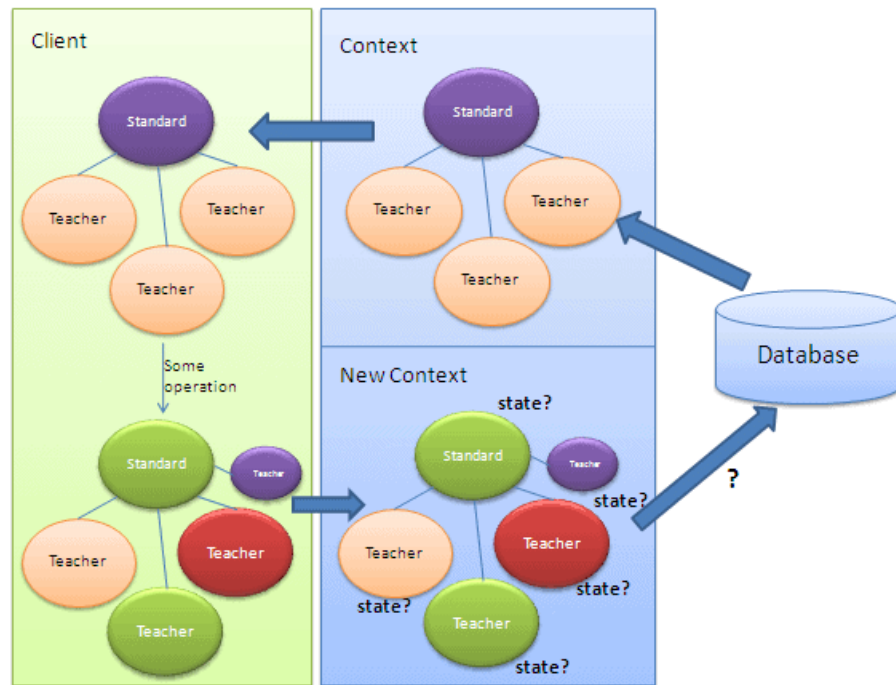
```

Như vậy không có sự khác biệt khi bạn thêm một thực thể đơn hoặc entity graph trong kịch bản disconnected hoặc connected. Chắc chắn rằng tất cả các thực thể đều mới.

II. Cập nhật Entity Graph sử dụng DbContext

Cập nhật một entity graph trong disconnected scenario là một tác vụ phức tạp. Nó là dễ dàng để thêm một entity graph mới trong mô hình disconnected nhưng để cập nhật một entity graph cần cân nhắc thiết kế cẩn thận.

Vấn đề trong việc cập nhật một entity graph in the disconnected scenario là context không biết thao tác gì được thực hiện trên nó ở phía client. Như hình minh họa sau context mới không biết trạng thái của mỗi thực thể:



Entity Graph in Disconnected Scenario

Chúng ta cần xác định trạng thái của mỗi thực thể trong entity graph trước khi gọi phương thức SaveChanges() của context. Có những kiểu mẫu khác nhau để xác định trạng thái của thực thể mà chúng ta cần cân nhắc trong khi thiết kế lớp kết nối dữ liệu với Entity Framework.

Những kiểu mẫu của việc xác định trạng thái thực thể trong disconnected scenario:

Có vài cách (trình bày bên dưới) để xác định một trạng thái thực thể trong disconnected scenario:

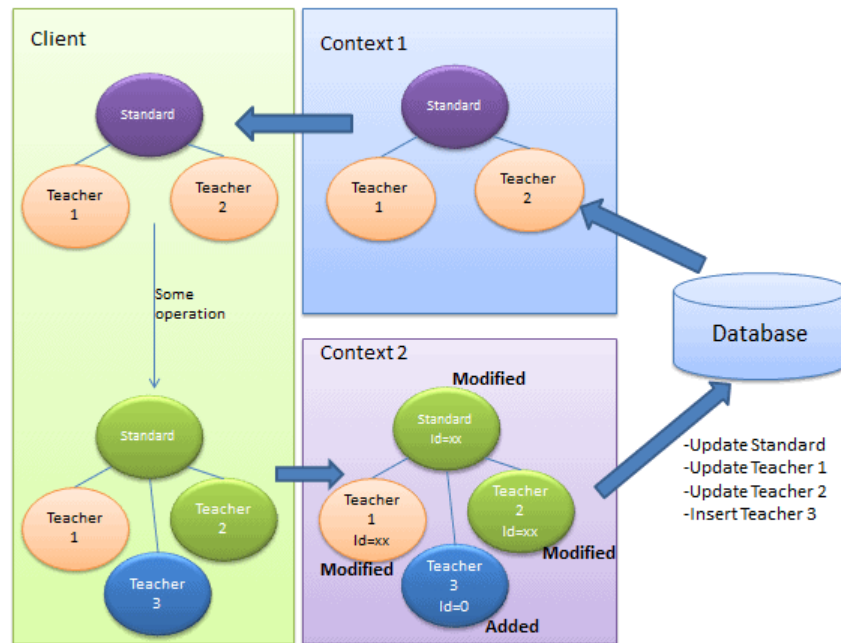
1. Sử dụng thuộc tính PrimaryKey (Id) của một thực thể.
2. Có thuộc tính State trong tập thực thể.

Note: Bạn có thể tạo thiết kế riêng của bạn để xác định một trạng thái thực thể dựa trên cấu trúc của bạn. Điều này chỉ dành cho mục đích học hỏi.

1) Sử dụng thuộc tính PrimaryKey của một thực thể:

Bạn có thể sử dụng thuộc tính PrimaryKey (Id) của mỗi thực thể để xác định trạng thái thực thể của nó. Tuy nhiên bạn phải quyết định sử dụng quy tắc cấu trúc nào sau đây:

- Mỗi kiểu thực thể phải có thuộc tính Id (PK).
- Giá trị mặc định của thuộc tính Id nên bằng 0.



Update Entity Graph in Disconnected Scenario using Id Property

Như bạn thấy trong hình minh họa trên đây, client tìm nạp entity graph Standard và Teacher và thực hiện vài thao tác trên đó rồi sau đó gửi nó tới Context 2 để lưu lại những thay đổi.

Trong disconnected scenario, context 2 không biết trạng thái của mỗi thực thể. Nó phải xác định trạng thái của thực thể Standard bằng cách sử dụng StandardId và trạng thái của thực thể Teacher bằng cách sử dụng thuộc tính TeacherId. Nếu giá trị của StandardId và TeacherID là 0 nó có nghĩa là một thực thể mới và nếu nó không bằng 0 thì nó có nghĩa là một thực thể đã chỉnh sửa.

Trong hình minh họa trên Standard, Teacher 1, và Teacher 2 có một thuộc tính Id không bằng 0 vì vậy chúng được đánh dấu là Modified và Teacher 3 bằng 0 được đánh dấu là Added.

```
Standard disconnectedStandard = null;

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

    disconnectedStandard = context.Standards.Where(s => s.StandardId == 58).Include(s => s.Teachers).FirstOrDefault<Standard>();
}
//Update Standard in disconnected mode
disconnectedStandard.StandardName = "Edited Standard Name";

//Update teachers collection by editing first teacher and adding new teacher
disconnectedStandard.Teachers.ElementAt(0).TeacherName = "Edited Teacher Name";
disconnectedStandard.Teachers.Add(new Teacher() { TeacherName = "New Teacher", StandardId = disconnectedStandard.StandardId });

using (var newContext = new SchoolDBEntities())
{

```

```

//mark standard based on StandardId
newContext.Entry(disconnectedStandard).State =
disconnectedStandard.StandardId == 0 ? EntityState.Added :
EntityState.Modified;

//mark teacher based on StandardId
foreach (Teacher tchr in disconnectedStandard.Teachers)
    newContext.Entry(tchr).State = tchr.TeacherId == 0 ?
EntityState.Added : EntityState.Modified;

newContext.SaveChanges();
}

```

Thuận lợi:

- Không cần viết mã hoặc xử lý thêm để xác định trạng thái thực thể.
- Hiệu năng tốt.

Bất lợi:

- Mỗi kiểu thực thể cần có một thuộc tính Id. Nó không thể xác định trạng thái của một thực thể không có thuộc tính Id.
- Không thể định danh một thực thể Unchanged. Nó được cài đặt thành trạng thái Modified thậm chí nếu thực thể không có bất kỳ thay đổi nào. Vì vậy có một câu lệnh update CSDL không cần thiết cho một thực thể unchanged.
- Không thể kiểm soát được kịch bản xóa thực thể. Nó cần kiểm soát sự xóa bỏ riêng.

2) Có thuộc tính State trong mỗi thực thể:

Một cách khác để xác định trạng thái thực thể là có một thuộc tính State trong mỗi kiểu thực thể và cài đặt một trạng thái thích hợp từ phía client trong mô hình disconnected. Sau đó chúng ta chỉ cần cài đặt trạng thái thực thể như thuộc tính trạng thái của một đối tượng thực thể trước khi gọi SaveChanges của context mới.

VD:

Đầu tiên tạo một giao diện IEntityState với thuộc tính enum gọi là EntityState:

```

interface IEntityObjectState
{
    EntityState ObjectState { get; set; }
}

public enum EntityState
{
    Added,
    Modified,
    Deleted,
    Unchanged
}

```

Giờ cài đặt một IEntityObjectState trong mỗi lớp thực thể. Vd: thực thể Standard và Teacher như bên dưới:

```
public partial class Standard:IEntityObjectState
{
    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; }
    [NotMapped]
    public EntityState ObjectState
    {
        get;
        set;
    }
}
```

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

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

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

    [NotMapped]
    public EntityState ObjectState
    {
        get;
        set;
    }
}
```

Cài đặt trạng thái thích hợp trong mô hình disconnected từ phía client:

```
Teacher existingTeacher = null;

using (var context = new SchoolDBEntities())
{
    context.Configuration.ProxyCreationEnabled = false;
    existingTeacher = context.Teachers.FirstOrDefault<Teacher>();
}

Standard disconnectedStandard = new Standard() { StandardName = "New
Standard", ObjectState = EntityState.Added };
existingTeacher.ObjectState = EntityState.Modified;
//add existing teacher(in db) to standard
disconnectedStandard.Teachers.Add(existingTeacher);
//add new standard
disconnectedStandard.Teachers.Add(new Teacher() { TeacherName = "New
teacher", StandardId = disconnectedStandard.StandardId, ObjectState =
EntityState.Added });
```

Cài đặt trạng thái thực thể theo EntityState trước khi gọi.

```
using (var newContext = new SchoolDBEntities())
{
    //check the EntityState property and mark appropriate EntityState
    if (disconnectedStandard.ObjectState == EntityState.Added)
        newContext.Entry(disconnectedStandard).State =
System.Data.Entity.EntityState.Added;
    else if (disconnectedStandard.ObjectState ==
EntityState.Modified)
        newContext.Entry(disconnectedStandard).State =
System.Data.Entity.EntityState.Modified;
    else if (disconnectedStandard.ObjectState ==
EntityState.Deleted)
        newContext.Entry(disconnectedStandard).State =
System.Data.Entity.EntityState.Deleted;
    else
        newContext.Entry(disconnectedStandard).State =
System.Data.Entity.EntityState.Unchanged;

    //check the EntityState property of each teacher and mark
appropriate EntityState
    foreach (Teacher tchr in disconnectedStandard.Teachers)
    {
```



```

        if (tchr.ObjectState == EntityState.Added)
            newContext.Entry(tchr).State =
System.Data.Entity.EntityState.Added;
        else if (tchr.ObjectState == EntityState.Modified)
            newContext.Entry(tchr).State =
System.Data.Entity.EntityState.Modified;
        else if (tchr.ObjectState == EntityState.Deleted)
            newContext.Entry(tchr).State =
System.Data.Entity.EntityState.Deleted;
        else
            newContext.Entry(tchr).State =
System.Data.Entity.EntityState.Unchanged;
    }
    //save changes
    newContext.SaveChanges();
}

```



Thuận lợi:

- Không cần viết mã hoặc xử lý thêm để xác định trạng thái thực thể.
- Kiểm soát được thuộc tính trạng thái Added, Modified, Deleted và Unchanged.
- Không cần gọi cập nhật không cần thiết cho những thực thể unchanged.

Bất lợi:

- Cần cài đặt những trạng thái thích hợp của mỗi thực thể trong mô hình disconnected. Vì vậy cần đặc biệt cẩn thận trong mô hình disconnected.

Phần 17 – Concurrency trong Entity Framework

| | Column Name | Data Type | Allow Nulls |
|---|-------------|-------------|-------------------------------------|
|  | StudentID | int | <input type="checkbox"/> |
| | StudentName | varchar(50) | <input checked="" type="checkbox"/> |
| | StandardId | int | <input checked="" type="checkbox"/> |
|  | RowVersion | timestamp | <input type="checkbox"/> |
| | | | <input type="checkbox"/> |

Mặc định Entity Framework hỗ trợ Optimistic Concurrency. Trong optimistic concurrency EF lưu thực thể vào CSDL, giả định rằng dữ liệu tương tự không thay đổi từ lúc thực thể được tải. Nếu nó xác định rằng dữ liệu đã thay đổi sau đó một exception được đưa ra và bạn phải loại bỏ xung đột trước khi thử lưu nó lại lần nữa.

Chúng ta hãy xem làm thế nào để kiểm soát optimistic concurrency với thực thể Student.

Đầu tiên bạn cần có một cột rowversion trong bảng Student để kiểm soát concurrency với thực thể Student. Rowversion là một kiểu dữ liệu trong SQL Server tự động khởi tạo số nhị phân duy nhất bất

cứ khi nào thao tác insert hoặc update được thực hiện trong một bảng. Kiểu dữ liệu rowversion đơn giản là một số tự tăng. Rowversion tương tự kiểu dữ liệu timestamp.

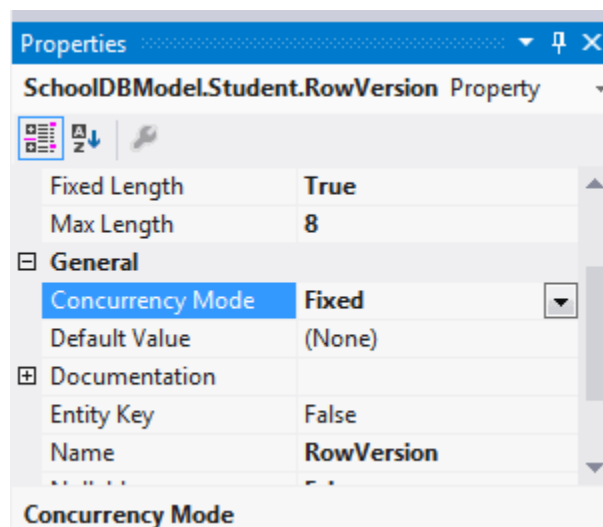
Tạo một cột mới RowVersion trong bảng Student với kiểu dữ liệu timestamp như trình bày bên dưới:

| | Column Name | Data Type | Allow Nulls |
|---|-------------|-------------|-------------------------------------|
| 🔑 | StudentID | int | <input type="checkbox"/> |
| | StudentName | varchar(50) | <input checked="" type="checkbox"/> |
| | StandardId | int | <input checked="" type="checkbox"/> |
| ▶ | RowVersion | timestamp | <input type="checkbox"/> |
| | | | <input type="checkbox"/> |

Note: Giá trị của RowVersion sẽ được thêm và cập nhật tự động bởi CSDL trong khi thao tác Insert và Update.

Giờ tạo một Entity Data Model như trình bày trong phần Tạo Entity Data Model hoặc nếu bạn đã có một EDM thì cập nhật nó bằng cách click chuột phải vào phần thiết kế -> Update Model From Database -> Refresh Student table. Và bạn sẽ nhìn thấy thuộc tính RowVersion được thêm vào thực thể Student.

Sau đó bạn cần cài đặt concurrency mode để sửa đổi bằng cách click chuột phải trên thuộc tính RowVersion trong thực thể Student (click chuột phải vào thuộc tính RowVersion chứ không phải thực thể Student) -> select Property. Đổi Concurrency Mode thành Fixed từ None trong cửa sổ thuộc tính như bên dưới:



EF giờ sẽ chèn một cột RowVersion trong mệnh đề where bất cứ khi nào bạn thực hiện một thao tác update và nếu giá trị rowversion là khác biệt hơn trong mệnh đề where thì nó sẽ ném ra DbUpdateConcurrencyException.

Đoạn mã sau chỉ ra rằng User1 và User2 lấy về cùng một sinh viên và update StudentName ở cùng một thời điểm:

```
Student student1WithUser1 = null;
Student student1WithUser2 = null;

//User 1 gets student
using (var context = new SchoolDBEntities())
{
    context.Configuration.ProxyCreationEnabled = false;
    student1WithUser1 = context.Students.Where(s => s.StudentID ==
1).Single();
}
//User 2 also get the same student
using (var context = new SchoolDBEntities())
{
    context.Configuration.ProxyCreationEnabled = false;
    student1WithUser2 = context.Students.Where(s => s.StudentID ==
1).Single();
}
//User 1 updates Student name
student1WithUser1.StudentName = "Edited from user1";

//User 2 updates Student name
student1WithUser2.StudentName = "Edited from user2";
```

User1 lưu thay đổi trước User2. Vì vậy khi User2 cố lưu những thay đổi sẽ nhận về một concurrency exception:

```

//User 1 saves changes first
using (var context = new SchoolDBEntities())
{
    try
    {
        context.Entry(student1WithUser1).State = EntityState.Modified;
        context.SaveChanges();
    }
    catch (DbUpdateConcurrencyException ex)
    {
        Console.WriteLine("Optimistic Concurrency exception occurred");
    }
}

//User 2 saves changes after User 1.
//User 2 will get concurrency exection
//because CreateOrModifiedDate is different in the database
using (var context = new SchoolDBEntities())
{
    try
    {
        context.Entry(student1WithUser2).State = EntityState.Modified;
        context.SaveChanges();
    }
    catch (DbUpdateConcurrencyException ex)
    {
        Console.WriteLine("Optimistic Concurrency exception occurred");
    }
}

```

Concurrency trong Code-First:

Bạn có thể tạo một đặc tính timestamp trong Code-First bằng cách sử dụng thuộc tính [Timestamp]. Chắc chắn rằng kiểu đặc tính là byte[] bởi vì timestamp là nhị phân trong C#.

```

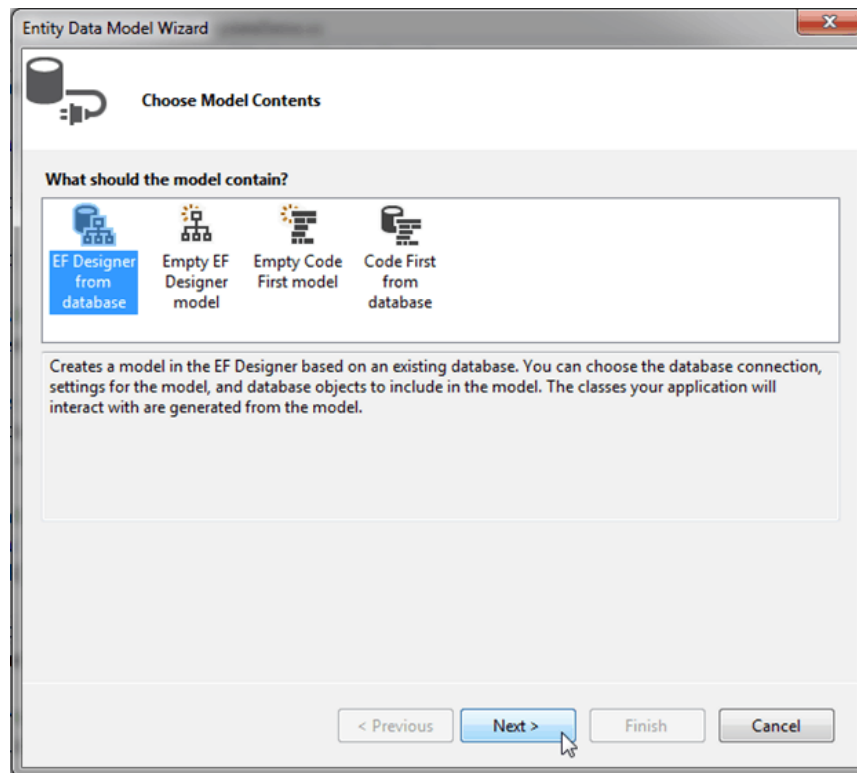
[Timestamp]
public byte[] RowVersion { get; set; }

```

EF chèn một đặc tính trong mệnh đề where trong khi thao tác update, nếu đặc tính được đánh dấu với thuộc tính Timestamp

You can resolve concurrency exceptions many ways. Visit [msdn](https://msdn.microsoft.com/en-us/library/gg695140.aspx) for detailed information on how to resolve optimistic concurrency.

Phần 18 – Stored Procedure trong Entity Framework



Entity Framework có khả năng tự động xây dựng những lệnh riêng cho CSDL dựa trên những truy vấn LINQ to Entities hoặc Entity SQL của bạn cũng như xây dựng những câu lệnh cho việc thêm, chỉnh sửa hoặc xóa dữ liệu. Bạn có thể muốn ghi đè những bước này và sử dụng những stored procedure riêng của bạn. Bạn có thể sử dụng stored procedures để lấy dữ liệu hoặc add/update/delete những record vào một hoặc nhiều bảng CSDL.

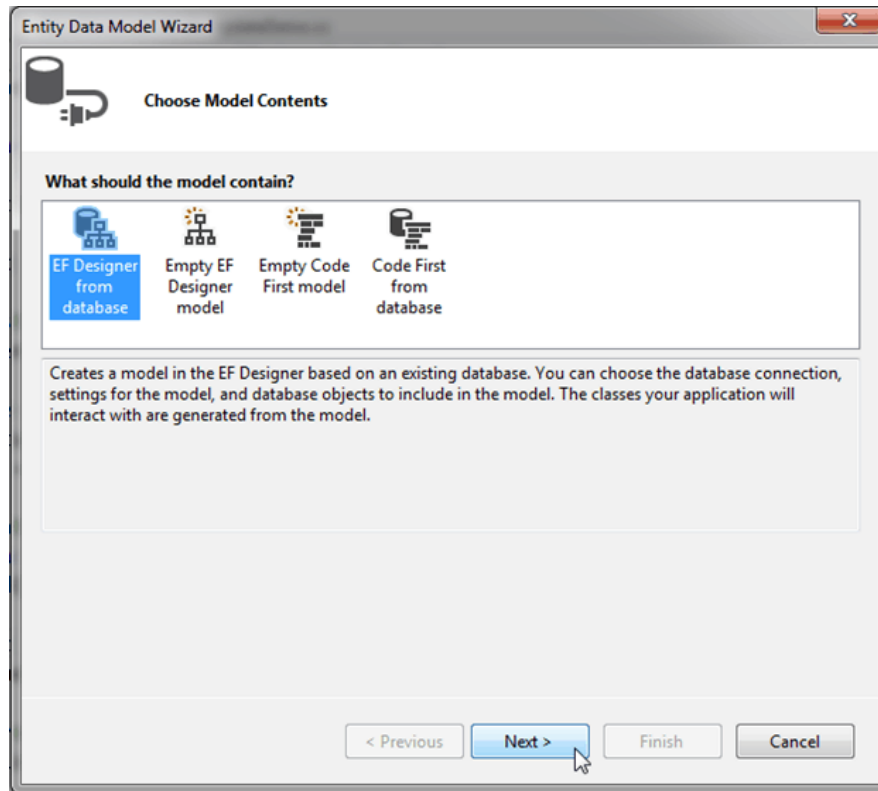
Stored procedures và user-defined functions (UDFs) trong CSDL được trình bày như những hàm trong Entity Framework. EDM sẽ không có bất kỳ thực thể nào cho stored procedures trong EDM designer.

Ở đây chúng ta sẽ thêm stored procedure sau `GetCoursesByStudentId` vào EDM. Procedure này trả về tất cả các khóa học được gán tới một sinh viên riêng biệt:

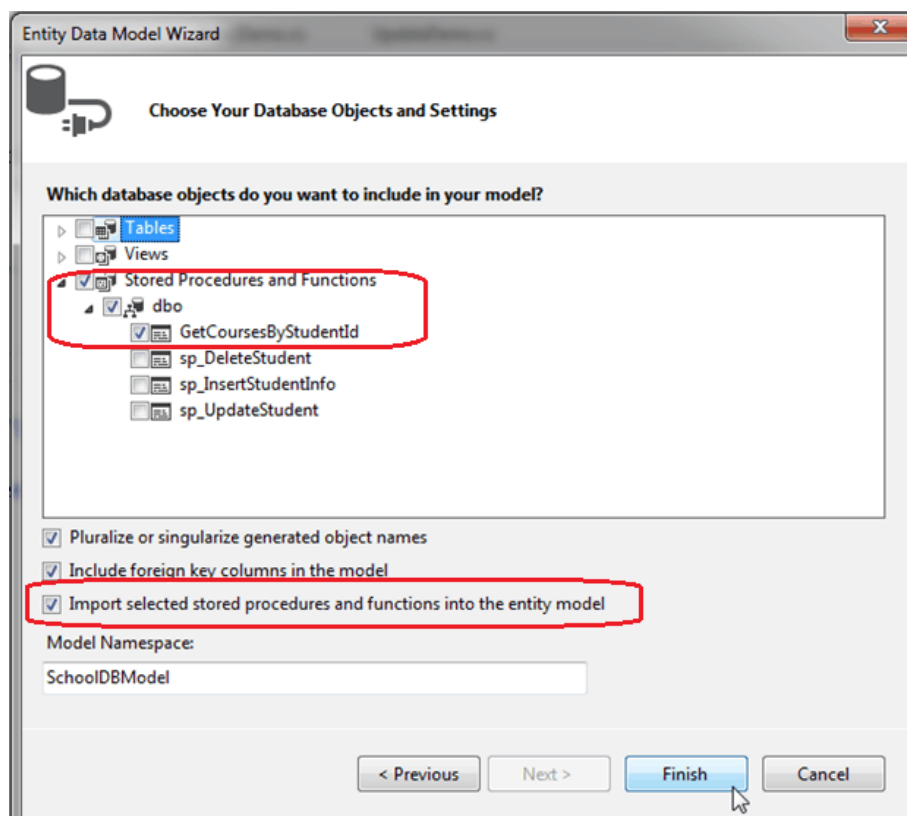
```
CREATE PROCEDURE [dbo].[GetCoursesByStudentId]
-- Add the parameters for the stored procedure here
    @StudentId int = null
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    select c.courseid, c.coursename, c.Location, c.TeacherId
    from student s
    left outer join studentcourse sc on sc.studentid = s.studentid
    left outer join course c on c.courseid = sc.courseid
    where s.studentid = @StudentId
END
```

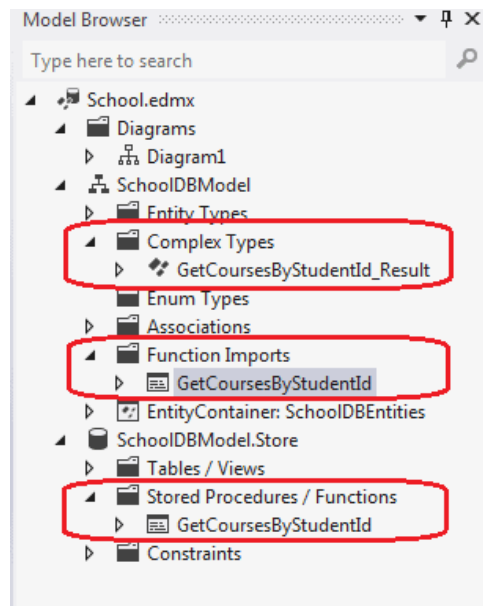
Đầu tiên tạo một ADO.Net Entity Data Model mới sử dụng EF Designer from database.



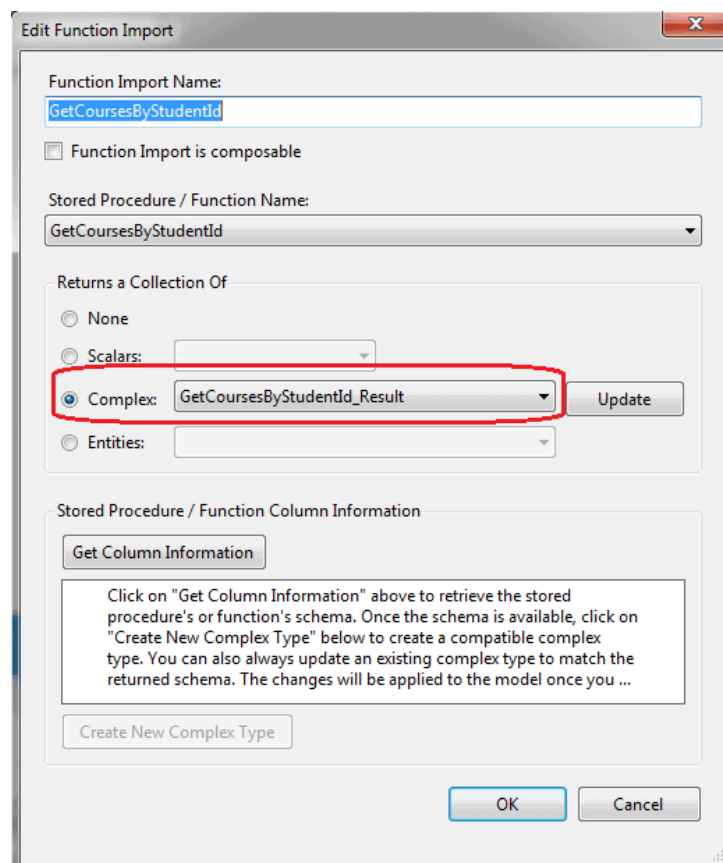
Chọn GetCoursesByStudentId. Chắc chắn rằng Import selected stored procedures and functions into the entity model checkbox được chọn và click Finish.

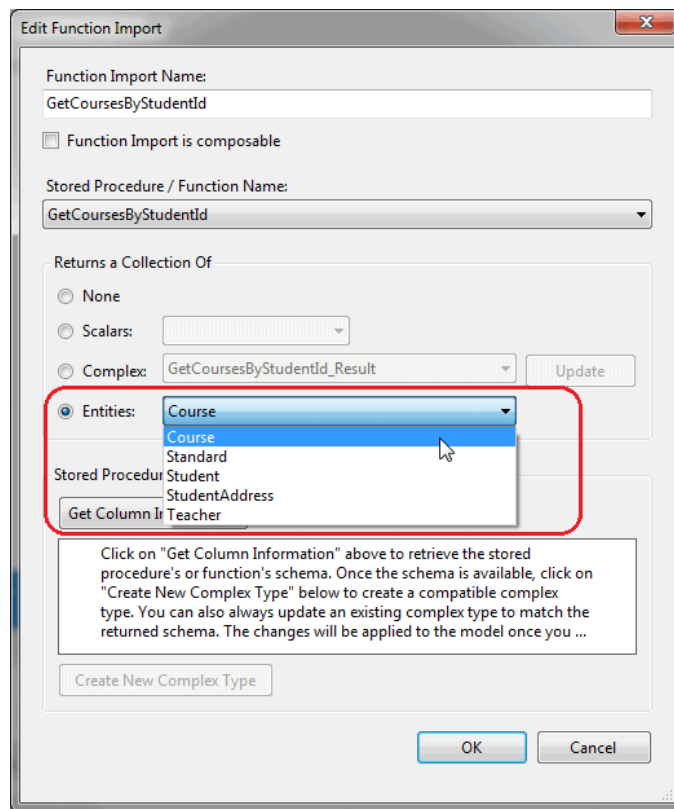


Bạn sẽ thấy `GetCoursesByStudentId` được thêm trong Function Imports với complex type mới `GetCoursesByStudentId_Result` trong Model Browser. Bất cứ khi nào bạn import một stored procedure vào một model, nó sẽ tạo một complex type mới với tên `{sp name}_Result` theo mặc định.



`GetCoursesByStudentId` trả về những trường giống nhau được định nghĩa trong thực thể `Course`. Vì vậy chúng ta không cần thêm một complex type mới cho dữ liệu trả về từ `GetCoursesByStudentId`. Bạn có thể thay đổi nó bằng cách click chuột phải vào `GetCoursesByStudentId` trong function imports và chọn `Edit`. Kiểm tra `Entities` và chọn `Course` từ dropdown trong cửa sổ popup như bên dưới:





Bạn sẽ nhìn thấy hàm trong lớp context cho GetCoursesByStudentId như bên dưới:

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

    protected override void OnModelCreating(DbModelBuilder modelBuilder) {...}

    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 ObjectResult<Course> GetCoursesByStudentId(Nullable<int> studentId)
    {
        var studentIdParameter = studentId.HasValue ?
            new ObjectParameter("StudentId", studentId) :
            new ObjectParameter("StudentId", typeof(int));

        return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction<Course>("GetCoursesByStudentId", studentIdParameter);
    }
}
```

Giờ GetCoursesByStudentId có thể được gọi và kết quả như bên dưới sẽ trả về:

```
using (var context = new SchoolDBEntities())
{
    var courses = context.GetCoursesByStudentId(1);

    foreach (Course cs in courses)
        Console.WriteLine(cs.CourseName);
}
```

The code shown above will execute the following statement: Đoạn mã trên đây sẽ thực thi những lệnh sau:

```
exec [dbo].[GetCoursesByStudentId] @StudentId=1
```

CRUD sử dụng Stored Procedure:

Ở mục này chúng ta sẽ sử dụng stored procedures cho thao tác CUD (create, update, delete) sử dụng DbContext. Có nghĩa rằng context sẽ thực thi những stored procedure thay vì những lệnh DDL trên context.SaveChanges().

Chúng ta sẽ sử dụng những stored procedure sau:

1. sp_InsertStudentInfo để thêm một sinh viên mới vào CSDL.
2. sp_UpdateStudent để cập nhật sinh viên.
3. sp_DeleteStudent để xóa một sinh viên trong CSDL.

Sp_InsertStudentInfo:

```
CREATE PROCEDURE [dbo].[sp_InsertStudentInfo]
-- Add the parameters for the stored procedure here
    @StandardId int = null,
    @StudentName varchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    INSERT INTO [SchoolDB].[dbo].[Student]([StudentName],
    [StandardId])
        VALUES(@StudentName, @StandardId)

    SELECT SCOPE_IDENTITY() AS StudentId

END
```

sp_UpdateStudent:

```
CREATE PROCEDURE [dbo].[sp_UpdateStudent]
-- Add the parameters for the stored procedure here
    @StudentId int,
    @StandardId int = null,
    @StudentName varchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    Update [SchoolDB].[dbo].[Student]
    set StudentName = @StudentName, StandardId = @StandardId
    where StudentID = @StudentId;

END
```

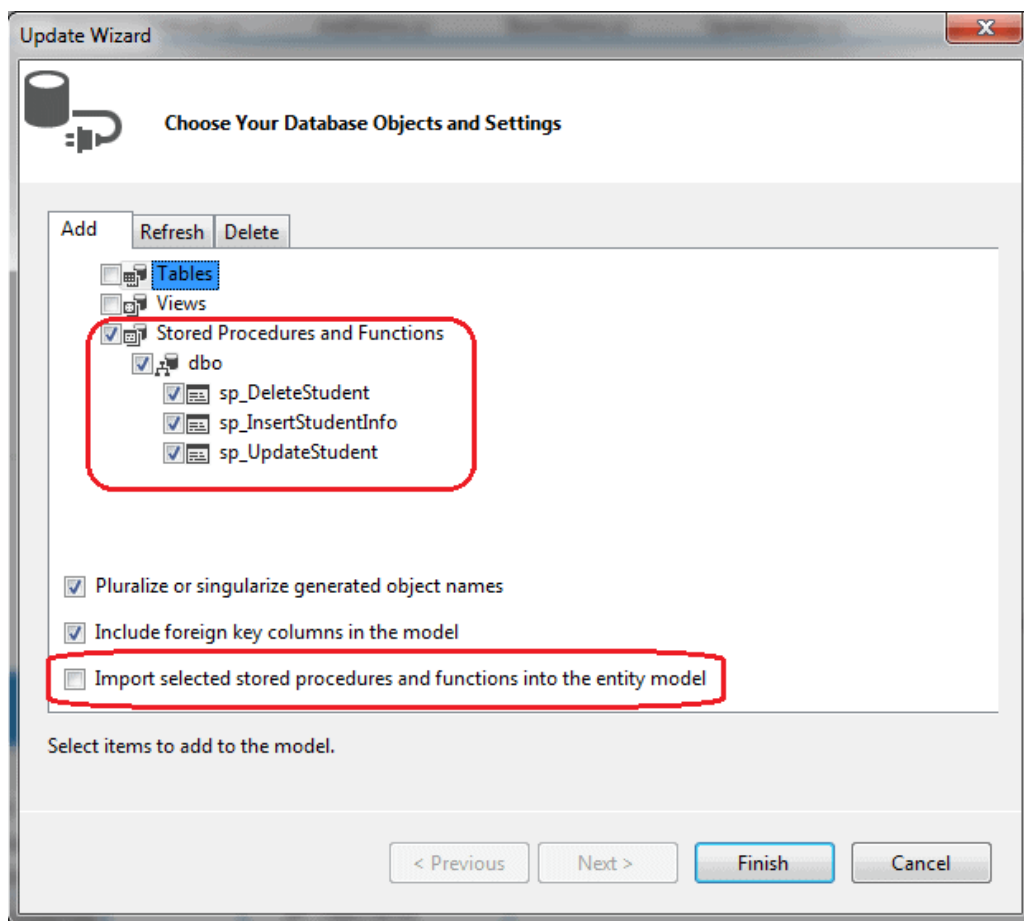
sp_DeleteStudent

```
CREATE PROCEDURE [dbo].[sp_DeleteStudent]
-- Add the parameters for the stored procedure here
    @StudentId int
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

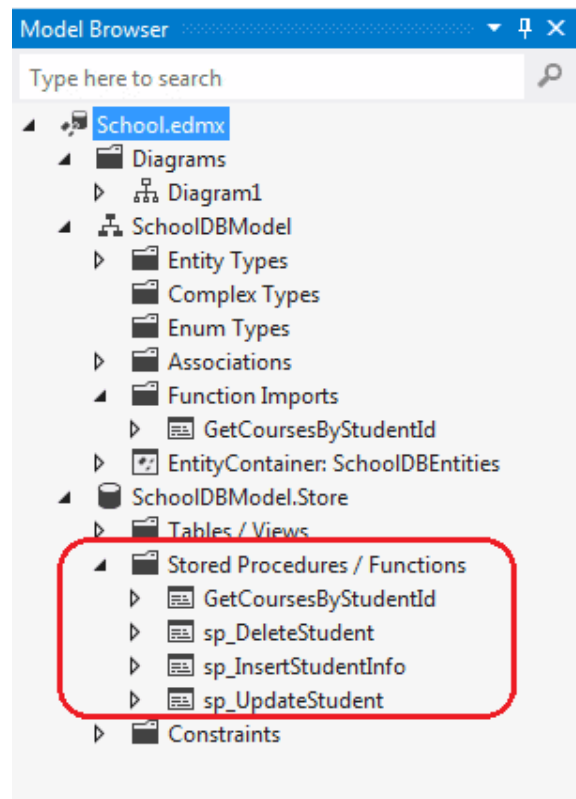
    DELETE FROM [dbo].[Student]
    where StudentID = @StudentId

END
```

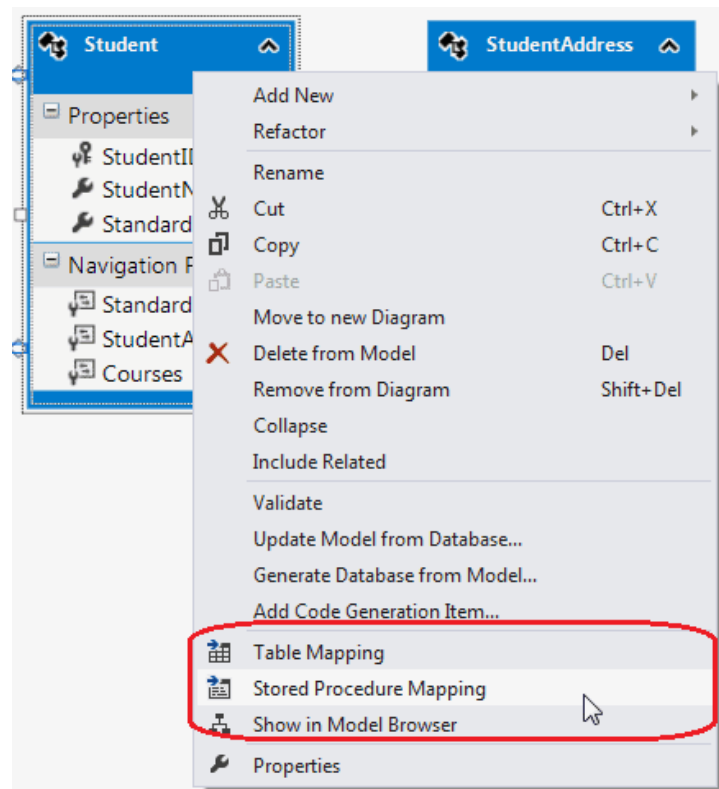
Đầu tiên thêm những stored procedure này vào EDM và chắc chắn rằng Import selected stored procedures and function into the entity model checkbox là unchecked vì chúng ta sẽ nối những procedure này với thực thể Student trực tiếp.



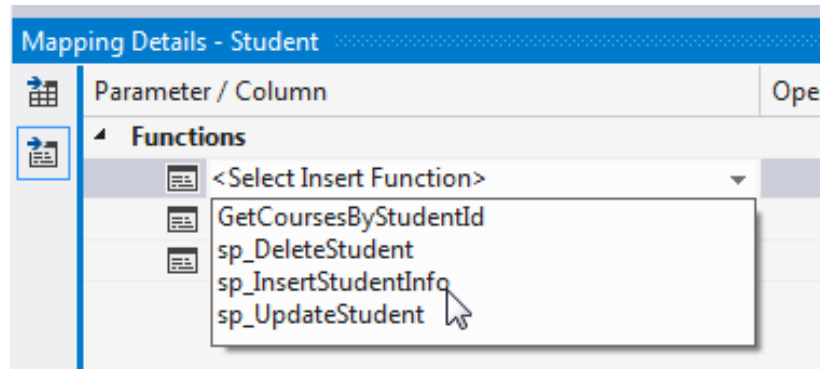
Trong Model Browser sẽ thêm những procedure vào Storage model mà không phải trong Function Imports.



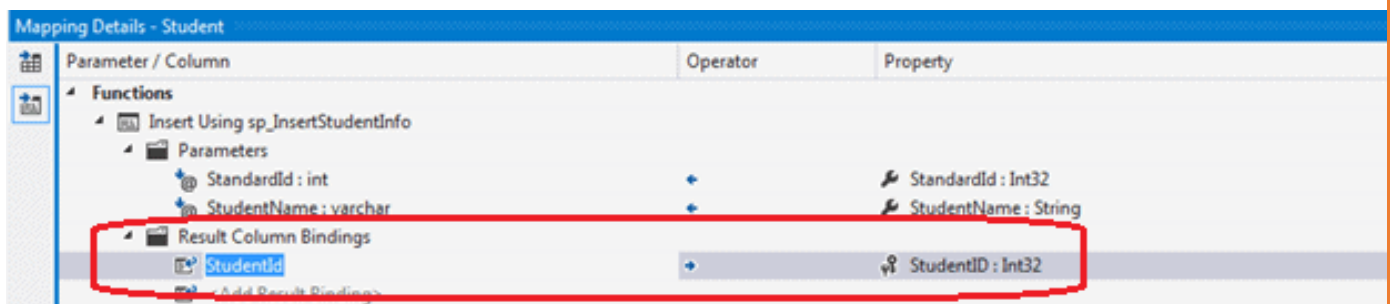
Ở EDM designer, click chuột phải vào thực thể Student và chọn Stored Procedure Mapping để mở Mapping details:



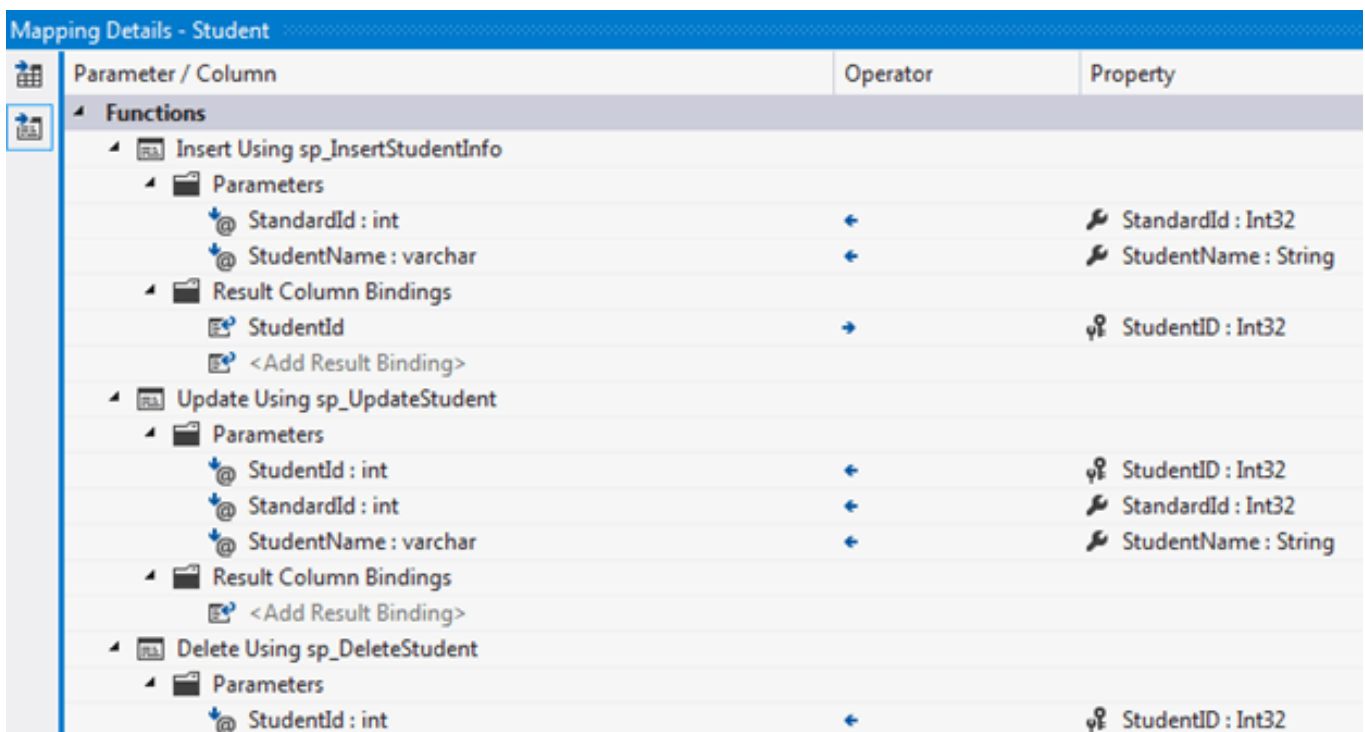
Trong Mapping Details, bạn sẽ thấy <Select Insert Function>, <Select Update Function> và <Select Delete Function>. Chọn stored procedure phù hợp cho mỗi cái. Vd: Chọn sp_InsertStudentInfo cho hàm Insert như bên dưới:



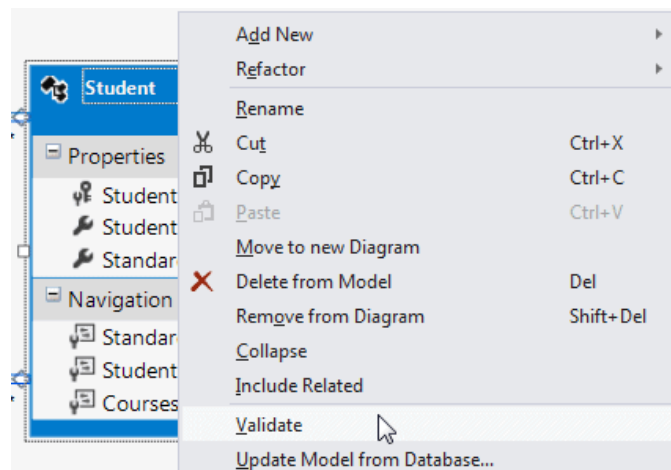
sp_InsertStudentInfo trả về StudentId mới khởi tạo. Nối cái này với StudentID của thực thể Student như bên dưới:



Hoàn thành việc nối những procedure Insert, Update và Delete như bên dưới:



Giờ chúng ta cần kiểm tra nó trước khi thực thi để đảm bảo rằng sẽ không có lỗi ở thời gian thực thi. Để thực hiện việc này, click chuột phải vào thực thể Student trong phần thiết kế và click **Validate** và chắc chắn rằng không có cảnh báo hoặc lỗi nào:



Bạn có thể thêm, cập nhật và xóa sinh viên như bên dưới:

```
using (var context = new SchoolDBEntities())
{
    Student newStudent = new Student() { StudentName = "New Student using SP" };

    context.Students.Add(newStudent);
    //will execute sp_InsertStudentInfo
    context.SaveChanges();

    newStudent.StudentName = "Edited student using SP";
    //will execute sp_UpdateStudent
    context.SaveChanges();

    context.Students.Remove(newStudent);
    //will execute sp_DeleteStudentInfo
    context.SaveChanges();
}
```

Đoạn mã trên sẽ thực thi những stored procedure sau trên mỗi SaveChanges():

```
exec [dbo].[sp_InsertStudentInfo] @StandardId=NULL,@StudentName='New Student using SP'
go

exec [dbo].[sp_UpdateStudent]
@StudentId=47,@StandardId=NULL,@StudentName='Edited student using SP'
go

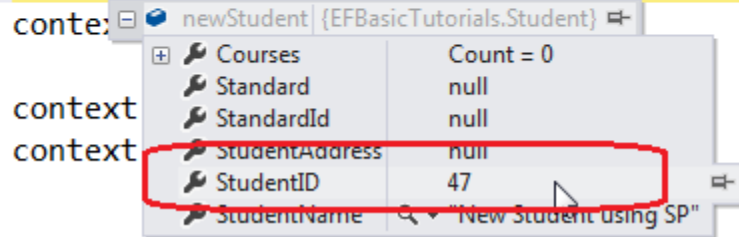
exec [dbo].[sp_DeleteStudent] @StudentId=47
go
```

Note: Một khi context gọi SaveChanges sau khi thêm một sinh viên mới, nó sẽ gán StudentID mới vào thuộc tính StudentID của thực thể Student bởi vì sp_InsertStudentInfo trả về StudentId. Điều này là cần thiết để sử dụng đối tượng thực thể đó cho những thao tác xa hơn.

```
Student newStudent = new Student() { StudentName = "New

context.Students.Add(newStudent);
context.SaveChanges();
```

```
newStudent.StudentName = "Edited student using SP";
```

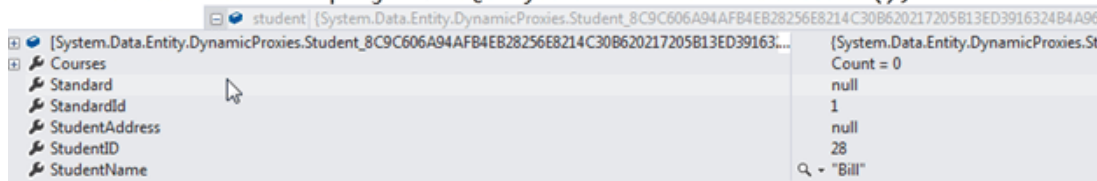


Phần 19 – Eager Loading, Lazy Loading và Explicit Loading trong Entity Framework

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

    var projectionQuery = from s in context.Students
                          where s.StudentName == "Bill"
                          select s;
```

```
Student student = projectionQuery.FirstOrDefault<Student>();
```



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

```
}
```

I. Eager Loading

Eager loading là quá trình nhờ đó một truy vấn cho một kiểu của thực thể cũng tải những thực thể liên quan như một phần của truy vấn. Eager loading được thực hiện bằng cách sử dụng phương thức Include().

Trong ví dụ sau nó lấy tất cả những sinh viên từ CSDL cùng với trình độ của nó sử dụng phương thức Include()

LINQ Query Syntax:

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

LINQ Method Syntax:

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

Đoạn mã trên sẽ có kết quả như truy vấn SQL sau:

```
SELECT TOP (1)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent2].[StandardId] AS [StandardId],
[Extent2].[StandardName] AS [StandardName],
[Extent2].[Description] AS [Description]
FROM [dbo].[Student] AS [Extent1]
LEFT OUTER JOIN [dbo].[Standard] AS [Extent2] ON [Extent1].[StandardId]
= [Extent2].[StandardId]
WHERE 'Student1' = [Extent1].[StudentName]
```

Use Lambda Expression:

Bạn cũng có thể sử dụng LINQ lambda expression trong phương thức Include. Để làm điều này lấy một tham chiếu của System.Data.Entity namespace và sử dụng lambda expression như dưới đây:

```
using System;
using System.Data.Entity;

class Program
{
    static void Main(string[] args)
    {
        using (var ctx = new SchoolDBEntities())
        {
            stud = ctx.Students.Include(s => s.Standard)
                                .Where(s => s.StudentName == "Student1")
                                .FirstOrDefault<Student>();
        }
    }
}
```


Đoạn mã trên sẽ có kết quả như truy vấn SQL sau:

```
SELECT TOP (1)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent2].[StandardId] AS [StandardId],
[Extent2].[StandardName] AS [StandardName],
[Extent2].[Description] AS [Description]
FROM [dbo].[Student] AS [Extent1]
LEFT OUTER JOIN [dbo].[Standard] AS [Extent2] ON [Extent1].[StandardId]
= [Extent2].[StandardId]
WHERE 'Student1' = [Extent1].[StudentName]
```

Tải nhiều cấp của những thực thể liên quan:

Bạn cũng có thể eagerly load nhiều cấp của những thực thể liên quan. Đoạn mã sau tải Student, Standard và Teachers liên quan:

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

Hoặc sử dụng lambda expression như bên dưới:

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

Đoạn mã trên có kết quả như truy vấn SQL sau:

```
SELECT [Project2].[StudentID] AS [StudentID],
[Project2].[StudentName] AS [StudentName],
[Project2].[StandardId] AS [StandardId],
[Project2].[StandardName] AS [StandardName],
[Project2].[Description] AS [Description],
[Project2].[C1] AS [C1],
[Project2].[TeacherId] AS [TeacherId],
[Project2].[TeacherName] AS [TeacherName],
[Project2].[StandardId1] AS [StandardId1]
FROM ( SELECT
[Limit1].[StudentID] AS [StudentID],
[Limit1].[StudentName] AS [StudentName],
[Limit1].[StandardId1] AS [StandardId],
[Limit1].[StandardName] AS [StandardName],
[Limit1].[Description] AS [Description],
[Project1].[TeacherId] AS [TeacherId],
[Project1].[TeacherName] AS [TeacherName],
[Project1].[StandardId] AS [StandardId1],
CASE WHEN ([Project1].[TeacherId] IS NULL) THEN CAST(NULL AS int)
ELSE 1 END AS [C1]
FROM [Project1] AS [Project1]
LEFT OUTER JOIN [Project2] AS [Project2] ON [Project1].[StandardId] = [Project2].[StandardId]
WHERE [Project1].[StudentName] = 'Student1')
```

```

FROM (SELECT TOP (1) [Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName], [Extent1].[StandardId] AS
[StandardId2], [Extent2].[StandardId] AS [StandardId1], [Extent2].
[StandardName] AS [StandardName], [Extent2].[Description] AS
[Description]
FROM [dbo].[Student] AS [Extent1]
LEFT OUTER JOIN [dbo].[Standard] AS [Extent2] ON [Extent1].
[StandardId] = [Extent2].[StandardId]
WHERE 'updated student' = [Extent1].[StudentName] ) AS [Limit1]
LEFT OUTER JOIN (SELECT
[Extent3].[TeacherId] AS [TeacherId],
[Extent3].[TeacherName] AS [TeacherName],
[Extent3].[StandardId] AS [StandardId]
FROM [dbo].[Teacher] AS [Extent3]
WHERE [Extent3].[StandardId] IS NOT NULL ) AS [Project1] ON
[Limit1].[StandardId2] = [Project1].[StandardId]
) AS [Project2]
ORDER BY [Project2].[StudentID] ASC, [Project2].[StandardId] ASC,
[Project2].[C1] ASC

```

II. Lazy Loading

Một trong những chức năng quan trọng của Entity Framework là lazy loading. Lazy loading có nghĩa là trì hoãn việc tải những dữ liệu liên quan cho tới khi bạn đưa ra yêu cụ thể cho nó. VD: Lớp Student chứa StudentAddress như một thuộc tính complex. Vì vậy đầu tiên context tải tất cả những sinh viên từ CSDL rồi sau đó nó sẽ tải địa chỉ của một sinh viên riêng biệt khi chúng ta truy cập thuộc tính StudentAddress như bên dưới:

```

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;
}

```

Đoạn mã trên sẽ có kết quả trong hai truy vấn SQL. Đầu tiên nó sẽ tìm nạp tất cả sinh viên:

```

SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[StandardId] AS [StandardId]
FROM [dbo].[Student] AS [Extent1]

```

Rồi sau đó nó sẽ gửi truy vấn sau khi chúng ta nhận tham chiếu của StudentAddress:

```
exec sp_executesql N'SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[Address1] AS [Address1],
[Extent1].[Address2] AS [Address2],
[Extent1].[City] AS [City],
[Extent1].[State] AS [State]
FROM [dbo].[StudentAddress] AS [Extent1]
WHERE [Extent1].[StudentID] = @EntityKeyValue1,N'@EntityKeyValue1
int',@EntityKeyValue1=1
```

Tuy nhiên bạn cũng có thể tắt lazy loading cho một thuộc tính riêng biệt hoặc toàn bộ context. Để tắt lazy loading cho một thuộc tính riêng biệt đừng cài nó là virtual. Để tắt lazy loading cho tất cả những thực thể trong context, cài thuộc tính configuration của nó thành false:

```
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")
    {
        this.Configuration.LazyLoadingEnabled = false;
    }

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

Những quy tắc cho lazy loading:

1. context.Configuration.ProxyCreationEnabled nên là true.
2. context.Configuration.LazyLoadingEnabled nên là true.
3. Thuộc tính điều hướng nên được định nghĩa là public, virtual. Context sẽ không thực hiện lazy loading nếu thuộc tính không được định nghĩa là virtual.

III. Explicit Loading với DbContext

Thậm chí với lazy loading bị tắt, nó vẫn có thể lazy load những thực thể liên quan nhưng nó phải được thực hiện với một explicit call. Sử dụng phương thức Load của đối tượng DBEntityEntry để thực hiện việc này.

Đoạn mã sau explicitly loads Standard của Student riêng biệt sử dụng phương thức Reference() của DbEntityEntry:

```
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.Standard).Load();
}
```

Nếu bạn chạy đoạn mã trên bạn có thể nhìn thấy đầu tiên nó tải sinh viên mà không phải standard như bên dưới:

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

    var projectionQuery = from s in context.Students
                          where s.StudentName == "Bill"
                          select s;

    Student student = projectionQuery.FirstOrDefault<Student>();
```

| | |
|--|---------------------------------------|
| student (System.Data.Entity.DynamicProxies.Student_8C9C606A94AFB4EB28256E8214C30B620217205B13ED3916324B4A96) | |
| [System.Data.Entity.DynamicProxies.Student_8C9C606A94AFB4EB28256E8214C30B620217205B13ED3916324B4A96] | (System.Data.Entity.DynamicProxies.St |
| + | Count = 0 |
| + | Standard |
| + | StandardId |
| + | StudentAddress |
| + | StudentID |
| + | StudentName |

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

```
}
```

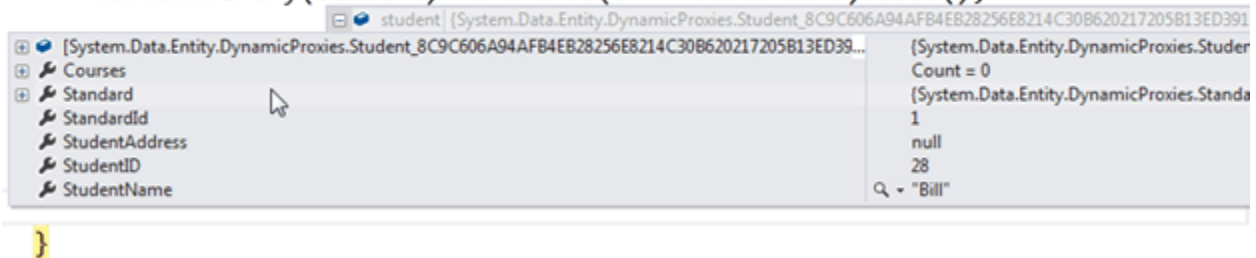
Phương thức load để nhận thực thể Standard là như bên dưới:

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

    var projectionQuery = from s in context.Students
                          where s.StudentName == "Bill"
                          select s;

    Student student = projectionQuery.FirstOrDefault<Student>();

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



Đoạn mã trên sẽ thực thi hai truy vấn CSDL khác nhau. Truy vấn đầu tiên nhận Student và truy vấn thứ hai nhận Standard.

Load collection:

Sử dụng phương thức Collection() thay vì phương thức Reference() để tải tập hợp thuộc tính điều hướng. Ví dụ sau tải những khóa học của sinh viên.

```
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();
}
```

Note: Phương thức Load mở rộng làm việc giống ToList ngoại trừ nó tránh việc tạo toàn bộ danh sách.

Phần 20 – Thực thi truy vấn Native SQL

Bạn có thể thực thi truy vấn native SQL thô dựa vào CSDL sử dụng DbContext. Bạn có thể thực thi những kiểu truy vấn sau:

1. Truy vấn SQL cho những kiểu thực thể trả về những kiểu riêng của thực thể.
2. Truy vấn SQL cho những kiểu không phải thực thể trả về một kiểu dữ liệu nguyên thủy.
3. Những lệnh SQL thô với CSDL.

Truy vấn SQL cho những kiểu thực thể:

Cũng như chúng ta đã nhìn thấy trong những phần trước DbSet có phương thức SqlQuery() để viết những truy vấn SQL thô và trả về những thể hiện của thực thể. Những đối tượng trả về sẽ được theo dõi bởi context, cũng giống như chúng sẽ được nếu trả về bởi một truy vấn LINQ. VD:

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

Tuy nhiên những cột trả về bởi truy vấn SQL nên khớp với thuộc tính của một tập thực thể của DbSet nếu không nó sẽ ném ra một exception. VD:

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

Nếu bạn thay đổi tên cột trong truy vấn thì nó sẽ ném ra một exception bởi vì nó phải khớp với tên cột:

```
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();
}
```

Truy vấn SQL cho những kiểu không phải thực thể:

Một truy vấn SQL trả về thể hiện của bất kỳ kiểu nào bao gồm những kiểu nguyên thủy có thể được tạo bằng cách dùng phương thức SqlQuery của lớp Database. VD:

```
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>();
}
```

Những lệnh SQL thô tới CSDL:

Phương thức ExecuteSqlCommnad là hữu ích trong việc gửi những lệnh non-query tới CSDL như là lệnh Insert, Update và Delete. VD:

```
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");
}
```

Phần 21 – Validate Entity

Bạn có thể viết validation tùy chỉnh phía server cho bất kỳ thực thể. Để thực hiện điều này override phương thức ValidateEntity của DbContext như bên dưới:

```
protected override
System.Data.Entity.Validation.DbEntityValidationResult
ValidateEntity(DbEntityEntry entityEntry,
System.Collections.Generic.IDictionary<object, object> items)
{
    if (entityEntry.Entity is Student)
    {
        if (entityEntry.CurrentValues.GetValue<string>("StudentName") ==
        "")
        {
            var list = new
List<System.Data.Entity.Validation.DbValidationError>();
            list.Add(new
System.Data.Entity.Validation.DbValidationError("StudentName",
"StudentName is required"));

            return new
System.Data.Entity.Validation.DbEntityValidationResult(entityEntry,
list);
        }
    }
    return base.ValidateEntity(entityEntry, items);
}
```

Như bạn nhìn thấy trong đoạn mã trên, chúng ta đang xác thực thực thể sinh viên. Nếu StudentName là để trống rồi sau đó chúng ta thêm DBValidationError vào DBEntityValidationResult. Vì vậy bất cứ khi nào bạn gọi phương thức DbContext.SaveChanges và thử lưu thực thể Student mà không có StudentName thì nó sẽ ném ra DbEntityValidationException. VD:

```
try
{
    using (var ctx = new SchoolDBEntities())
    {
        ctx.Students.Add(new Student() { StudentName = "" });
        ctx.Standards.Add(new Standard() { StandardName = "" });

        ctx.SaveChanges();
    }
}
catch (DbEntityValidationException dbEx)
{
    foreach (DbEntityValidationResult entityErr in
dbEx.EntityValidationErrors)
    {
        foreach (DbValidationError error in entityErr.ValidationErrors)
        {
            Console.WriteLine("Error Property Name {0} : Error Message:
{1}",
                                error.PropertyName, error.ErrorMessage);
        }
    }
}
```