عنوان: **روش های ارث بری در Entity Framework - قسمت اول**
نویسنده: بهروز راد
تاریخ: ۱۳۹۲/۰۲/۱۷ ۸:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Inheritance Strategies, Table per type, TPT

بخش هایی از کتاب "مرجع کامل Entity Framework 6.0"
ترجمه و تالیف: بهروز راد
وضعیت: در حال نگارش

پیشتر، آقای نصیری در بخشی از مباحث مربوط به Code First در مورد روش‌های مختلف ارث بری در EF و در روش Code First صحبت کرده اند. در این مقاله‌ی دو قسمتی، در مورد دو تا از این روش‌ها در حالت Database First می‌خوانید.

چرا باید از ارث بری استفاده کنیم؟

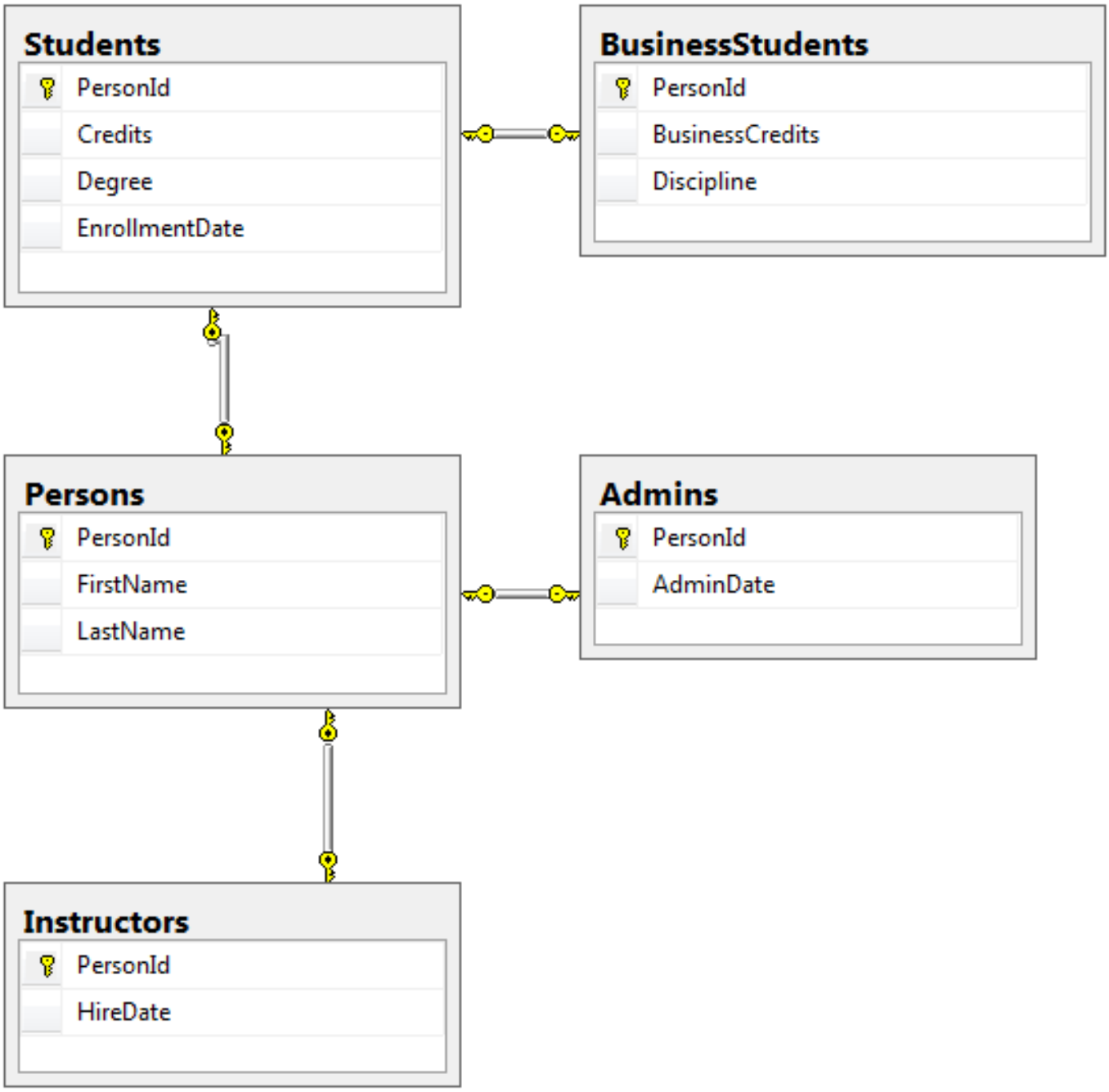
یکی از اهداف اصلی ORMها این است که با ایجاد یک مدل مفهومی از پایگاه داده، آن را هر چه بیشتر به طرز تفکر ما از مدل شی گرای برنامه مان نزدیکتر کنند. از آنجا که ما توسعه گران از مفاهیم شی گرایی مانند "ارث بری" در کدهای خود استفاده می‌کنیم، نیاز داریم تا این مفهوم را در سطح پایگاه داده نیز داشته باشیم. آیا این کار امکان پذیر است؟ EF چه امکاناتی برای رسیدن به این هدف برای ما فراهم کرده است؟ در این قسمت به این سوال پاسخ خواهیم داد.

ارث- بری جداول مفهومی است که در EF به راحتی قابل پیاده- سازی است. سه روش برای پیاده- سازی این مفهوم در مدل وجود دارد.

Table Per Type یا TPT: خصیصه‌های مشترک در جدول پایه قرار دارند و به ازای هر زیر مجموعه نیز یک جدول جدا ایجاد می‌شود.
Table Per Hierarchy یا TPH: تمامی خصیصه‌ها در یک جدول وجود دارند.
Table Per Concrete Type یا TPC: جدول پایه ای وجود ندارد و به ازای هر موجودیت دقیقاً یک جدول همراه با خصیصه‌های موجودیت در آن ایجاد می‌شود.

روش TPT

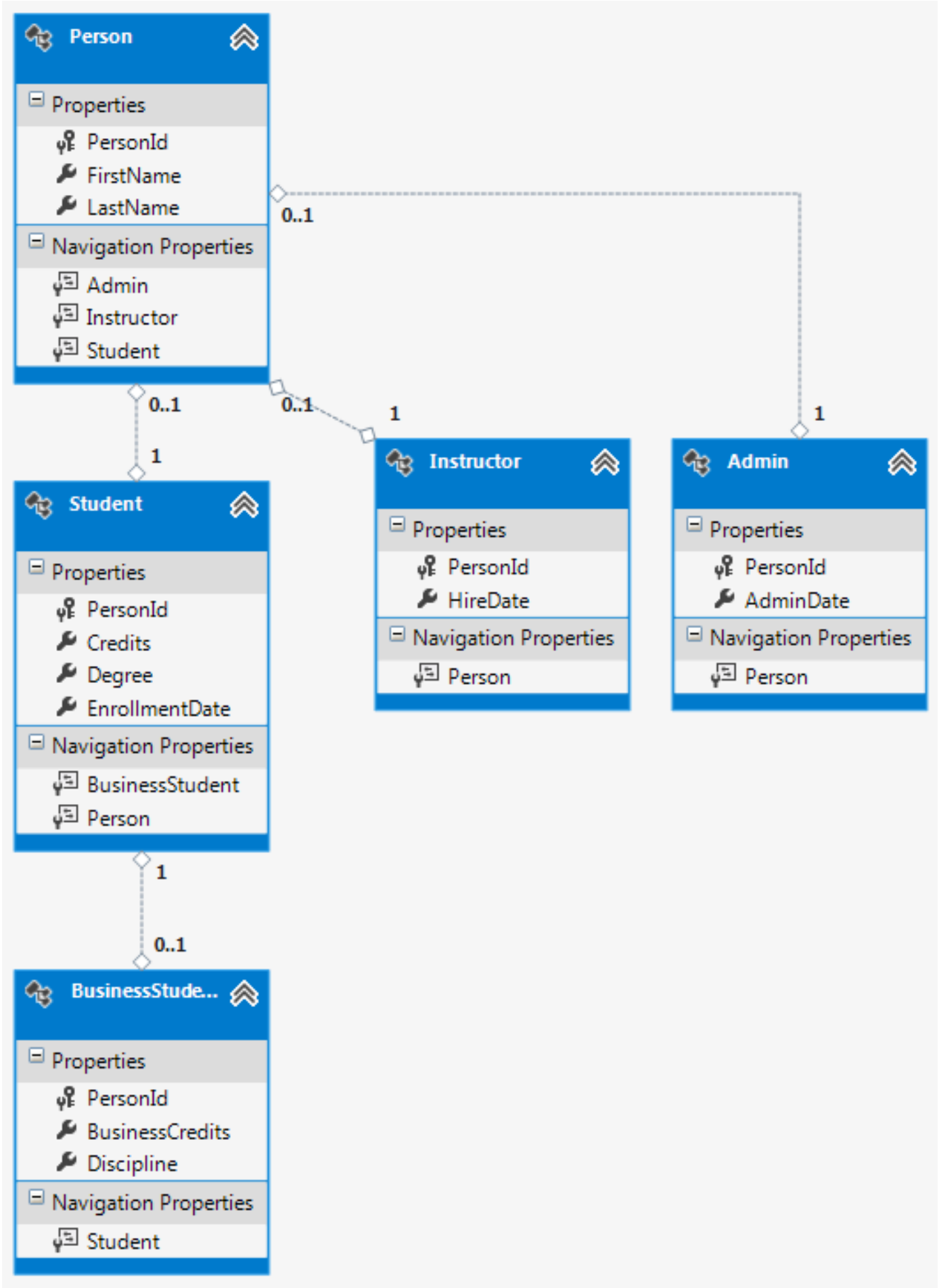
در این روش، خصیصه‌های مشترک در یک جدول پایه قرار دارند و به ازای هر زیر مجموعه از جدول پایه، یک جدول با خصیصه‌های منحصر به آن نوع ایجاد می‌شود. ابتدا جداول و ارتباطات بین آنها که در توضیح مثال برای این روش با آنها کار می‌کنیم را ببینیم.



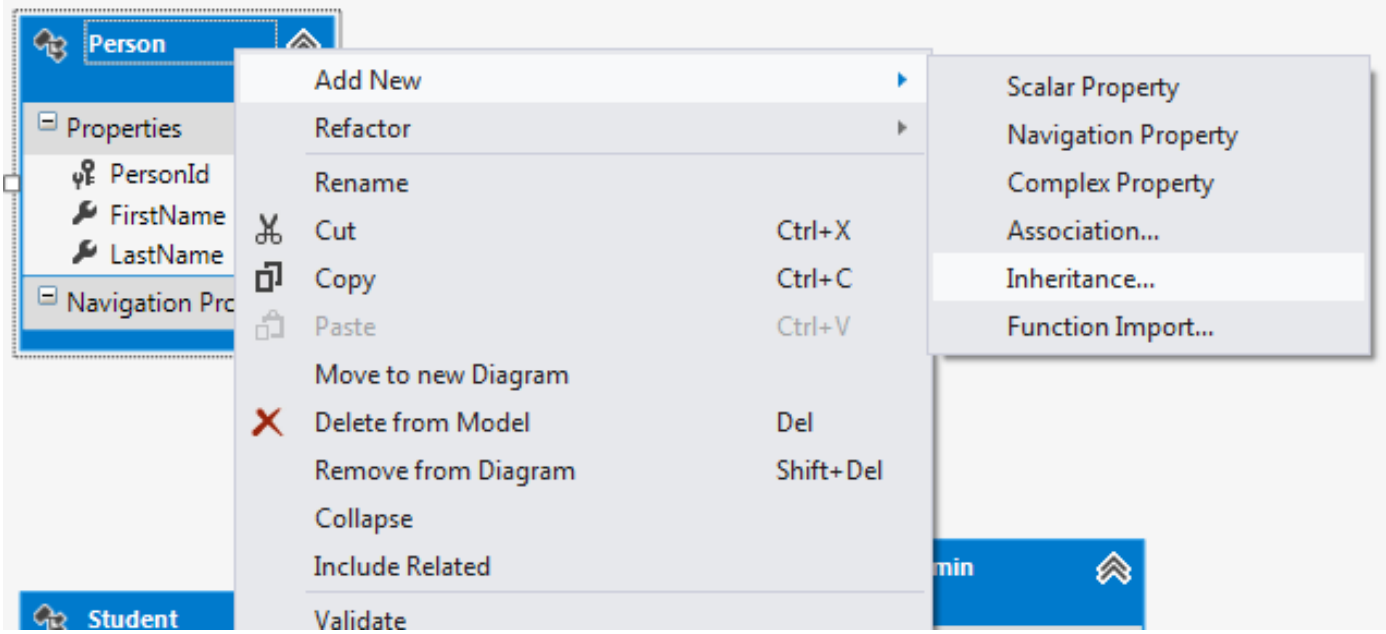
فرض کنید قصد داریم تا در هنگام ثبت مشخصات یک دانش آموز، مقطع تحصیلی او نیز حتماً ذخیره شود. در این حالت، فیلدی با نام Degree ایجاد و تیک گزینه‌ی Allow Nulls را از روبروی آن بر میداریم. با این حال اگر مشخصات دانش آموزان را در جدولی عمومی مثلاً با نام People ذخیره کنیم و این جدول را مکانی برای ذخیره‌ی مشخصات افراد دیگری مانند مدیران و معلمان نیز در نظر بگیریم، از آنجا که قصد ثبت مقطع تحصیلی برای مدیران و معلمان را نداریم، وجود فیلد Degree در کار ما اختلال ایجاد می‌کند. اما با ذخیره‌ی اطلاعات مدیران و معلمان در جداول مختص به خود، می‌توان قانون غیر قابل Null بودن فیلد Degree برای دانش آموزان را به راحتی پیاده سازی کرد.

همان طور که در شکل قبل نیز مشخص است، ما یک جدول پایه با نام Persons ایجاد کرده ایم و خصیصه‌های مشترک بین زیر مجموعه‌ها (FirstName و LastName) را در آن قرار داده ایم. سه موجودیت (Student، Admin و Instructor) از Persons ارث می‌برند و موجودیت BusinessStudent نیز از Student ارث می‌بَرَد.

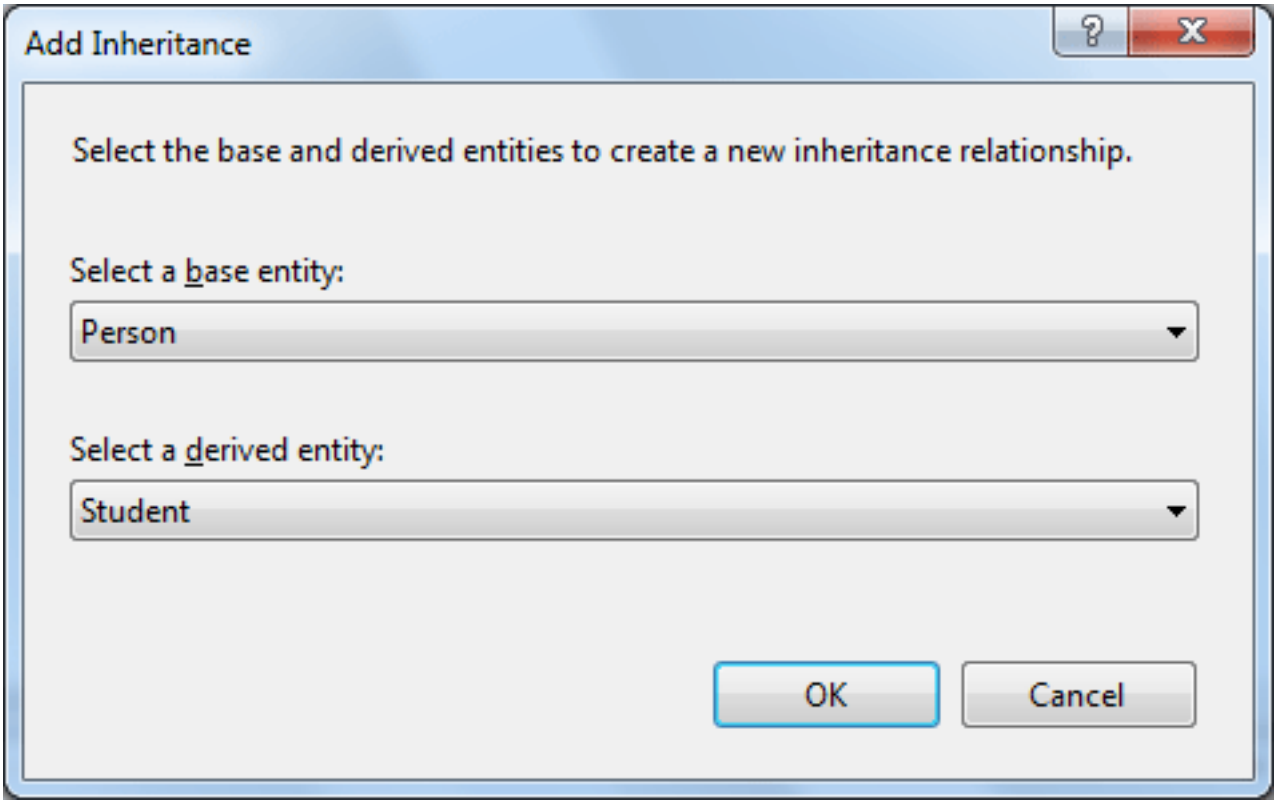
جداول ایجاد شده، پس از ایجاد مدل به روش Database First، به شکل زیر تبدیل می‌شوند.


Person
Properties
PersonId
FirstName
LastName
Navigation Properties
Admin
Instructor
Student
0..1
0..1
1
1
0..1
1
Student
Properties
PersonId
Credits
Degree
EnrollmentDate
Navigation Properties
BusinessStudent
Person
Instructor
Properties
PersonId
HireDate
Navigation Properties
Person
Admin
Properties
PersonId
AdminDate
Navigation Properties
Person
1
0..1
BusinessStude...
Properties
PersonId
BusinessCredits
Discipline
Navigation Properties
Student

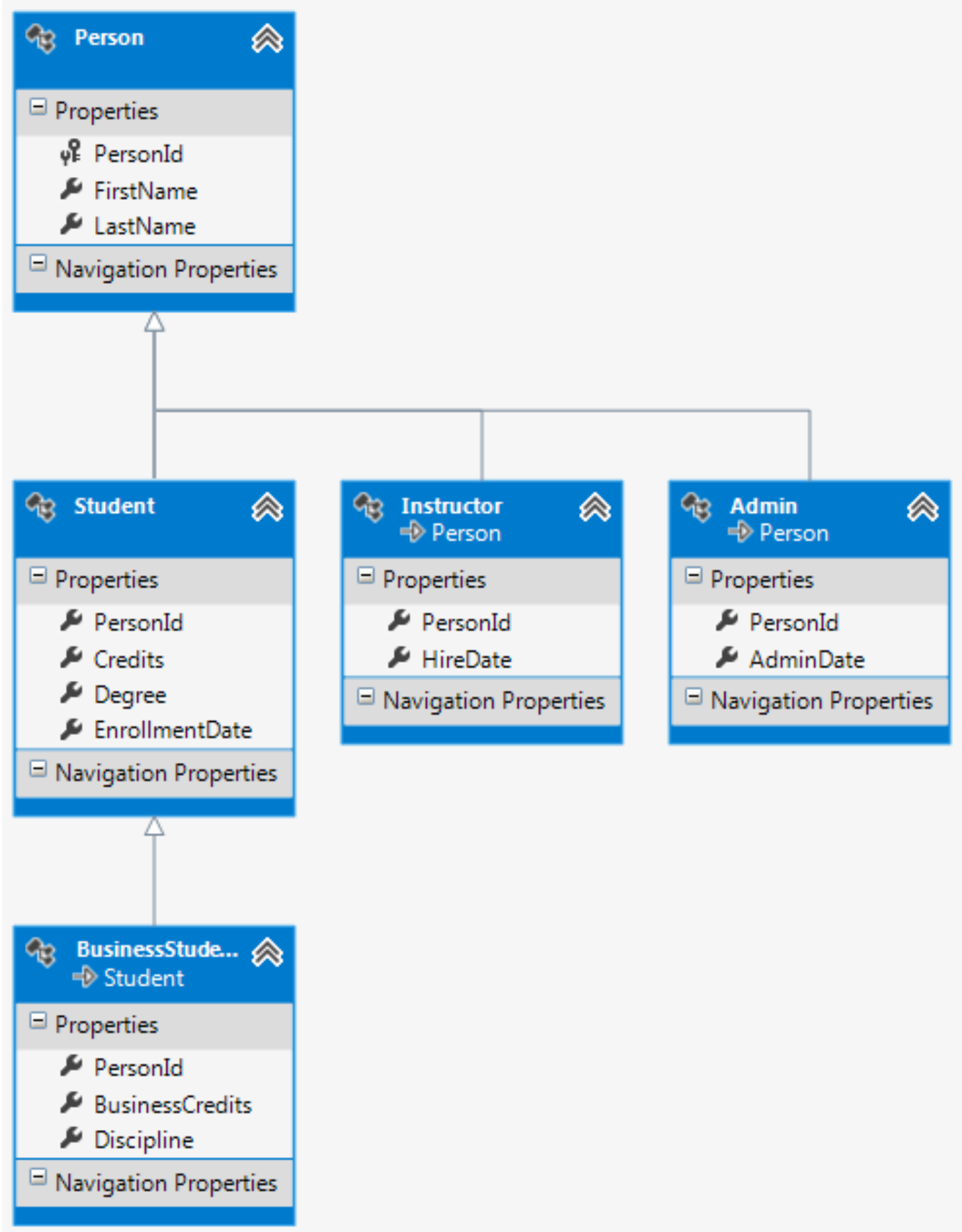
از آنجا که قصد داریم ارتباطات ارث بری شده ایجاد کنیم، باید ارتباطات پیش فرض شکل گرفته بین موجودیت‌ها را حذف کنیم. بدین منظور، بر روی هر خط ارتباطی در EDM Designer کلیک راست و گزینه‌ی Delete from Model را انتخاب کنید. سپس بر روی موجودیت Person، کلیک راست کرده و از منوی Add New، گزینه‌ی Inheritance را انتخاب کنید (شکل زیر).



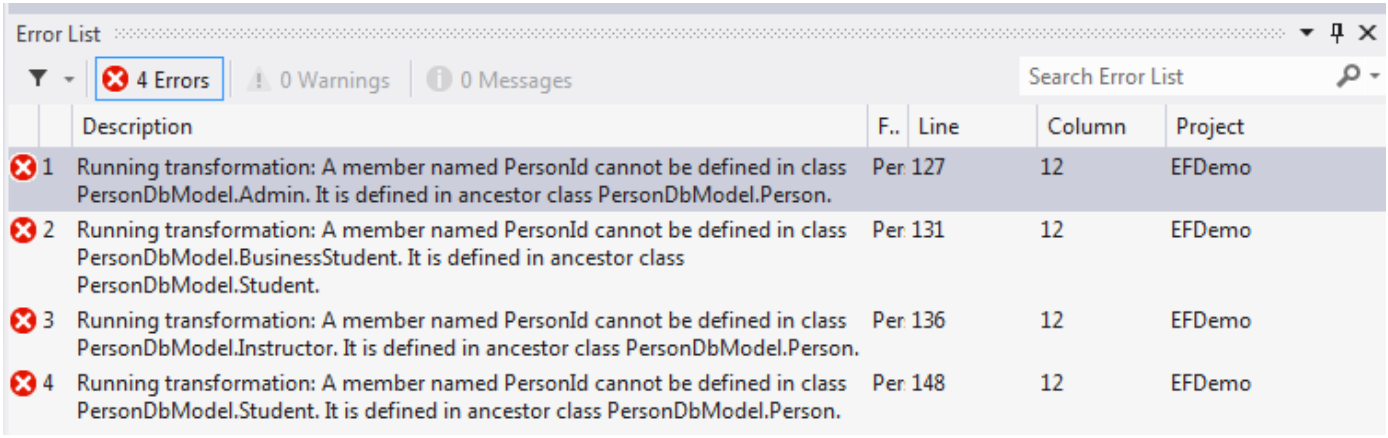
شکل زیر ظاهر می‌شود.



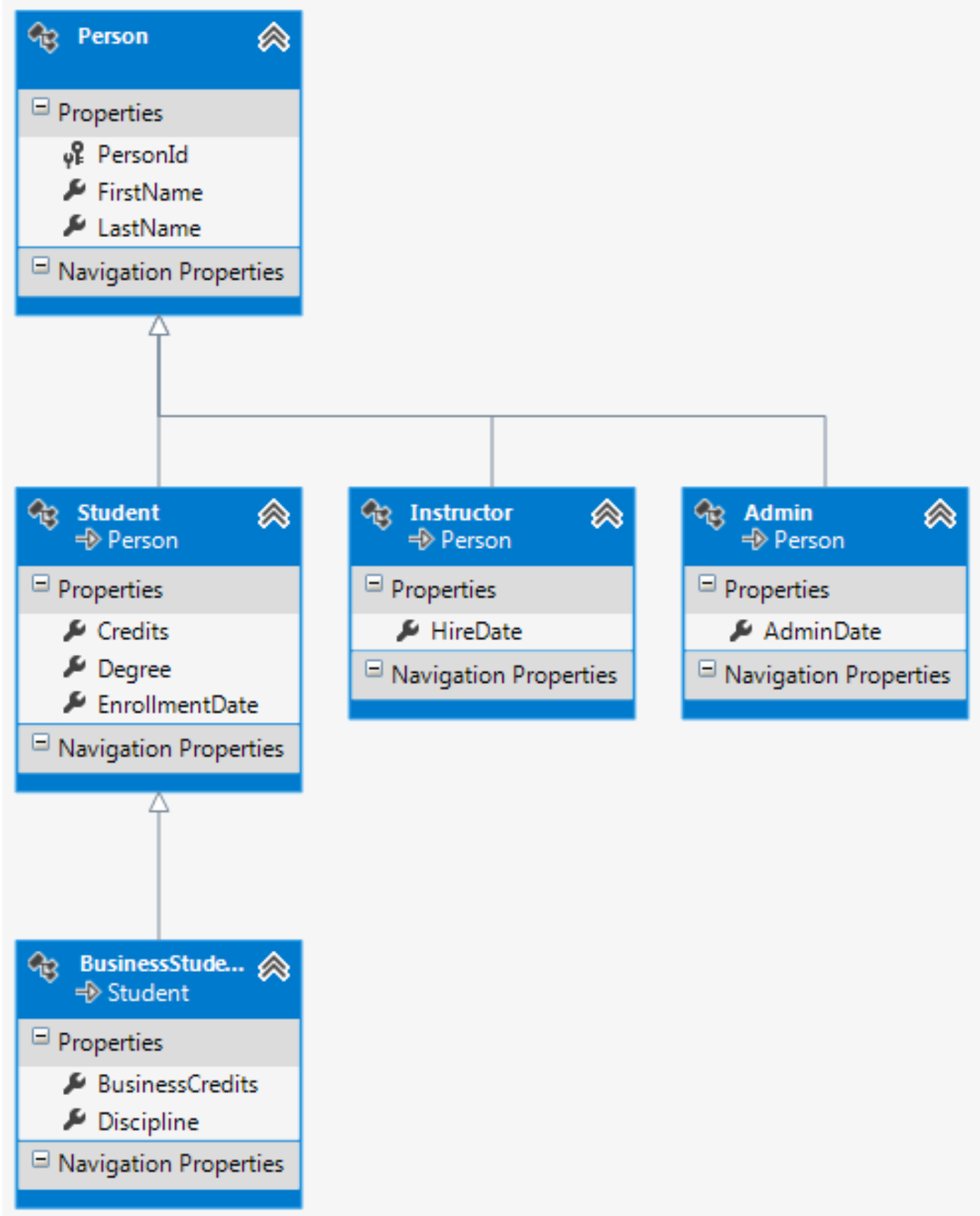
قسمت بالا، موجودیت پایه، و قسمت پایین، موجودیت مشتق شده را مشخص می‌کند. این کار را سه مرتبه برای ایجاد ارتباط ارث بری شده بین موجودیت Person به عنوان موجودیت پایه و موجودیت‌های Instructor، Student و Admin به عنوان موجودیت‌های مشتق شده ایجاد کنید. همچنین یک ارتباط نیز بین موجودیت Student به عنوان موجودیت پایه و موجودیت BusinessStudent به عنوان موجودیت مشتق شده ایجاد کنید. نتیجه‌ی کار را در شکل زیر ملاحظه می‌کنید.



اگر بر روی دکمه‌ی Save در نوار ابزار Visual Studio کلیک کنید، چهار خطا در پنجره‌ی Error List نمایش داده می‌شود



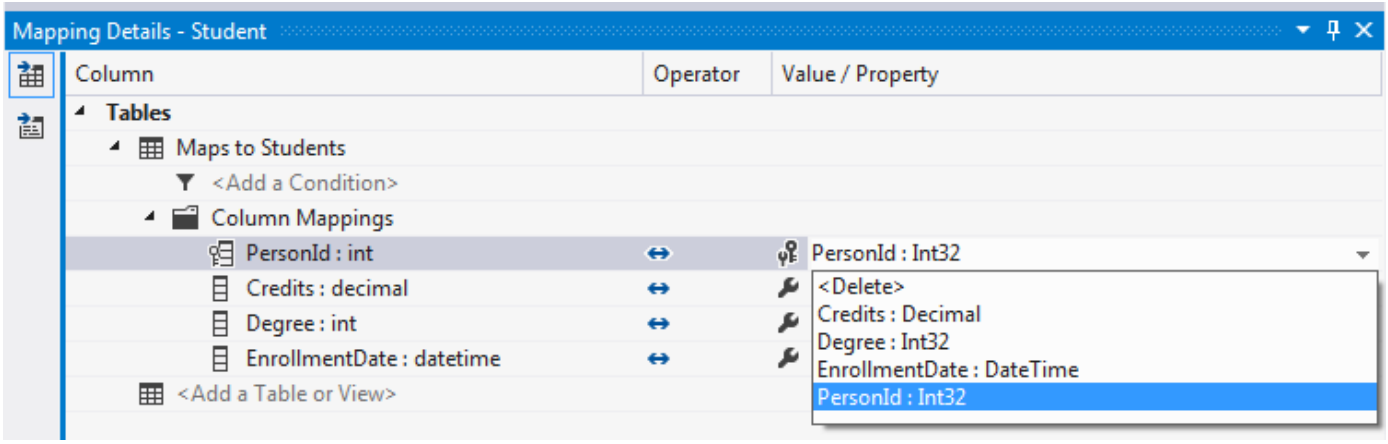
این خطاها بیانگر این هستند که خصیصه‌ی PersonId به دلیل اینکه در موجودیت پایه‌ی Person تعریف شده است، نباید در موجودیت‌های مشتق شده از آن نیز وجود داشته باشد چون موجودیت‌های مشتق شده، خصیصه‌ی PersonId را به ارث برده اند. وجود این خصیصه در زمان طراحی جدول در مدل فیزیکی الزامی بوده است اما اکنون ما با مدل مفهومی و قوانین شی گرایی سر و کار داریم. بنابراین خصیصه‌ی PersonId را از موجودیت‌های Admin، Instructor، Student و BusinessStudent حذف کنید.
شکل زیر، نتیجه‌ی کار را نشان می‌دهد.



اکنون اگر بر روی دکمه‌ی Save کلیک کنید، خطاها از بین می‌روند.

ما خصیصه‌ی PersonId را از موجودیت‌های مشتق شده به این دلیل که آن را از موجودیت پایه ارث می‌برند حذف کردیم. حال این خصیصه برای موجودیت‌های مشتق شده وجود دارد اما باید مشخص کنیم که به کدام خصیصه از کلاس پایه تناظر دارد. شاید

انتظار این باشد که EF، خود تشخیص بدهد که PersonId در موجودیت‌های مشتق شده باید به PersonId کلاس پایه‌ی خود تناظر داشته باشد اما در حال حاضر این کاری است که خود باید انجام دهیم. بدین منظور، بر روی هر یک از موجودیت‌های مشتق شده کلیک راست کرده و گزینه‌ی Table Mapping را انتخاب کنید. سپس همان طور که در شکل زیر مشاهده می‌کنید، تناظر را ایجاد کنید.



مدل ما آماده است. آن را امتحان می‌کنیم. در زیر، یک کوئری LINQ ساده بر روی مدل ایجاد شده را ملاحظه می‌کنید.

```
using (PersonDbEntities context = new PersonDbEntities())
{

    var people = from p in context.Persons
                 select p;

    foreach (Person person in people)
    {
        Console.WriteLine("{0}, {1}",
            person.LastName,
            person.FirstName);
    }

    Console.ReadLine();
}
```

قضیه به همین جا ختم نمی‌شود! ما الان یک مدل ارث بری شده داریم. بهتر است مزایای آن را در عمل ببینیم. شاید دوست داشته باشیم تا فقط اطلاعات زیر مجموعه‌ی BusinessStudent را بازیابی کنیم.

```
using (PersonDbEntities context = new PersonDbEntities())
{

    var students = from p in context.Persons.OfType<BusinessStudent>()
                   select p;

    foreach (BusinessStudent student in students)
    {
        Console.WriteLine("{0}, {1}: Degree {2}, Discipline {3}",
            student.LastName,
            student.FirstName,
            student.Degree,
            student.Discipline);
    }

    Console.ReadLine();
}
```

همان طور که در کدهای قبل نیز مشخص است، خصیصه‌های LastName و FirstName از موجودیت پایه یعنی Person، خصیصه‌ی Degree از موجودیت مشتق شده‌ی Student (که البته در نقش موجودیت پایه برای BusinessStudent است) و Discipline از موجودیت مشتق شده یعنی BusinessStudent خوانده می‌شوند.
یک روش دیگر نیز برای کار با این سلسه مراتب ارث بری وجود دارد. کوئری اول را دست نزنیم (اطلاعات موجودیت پایه را بازیابی کنیم) و پیش از انجام عملیاتی خاص، نوع موجودیت مشتق شده را بررسی کنیم. مثالی در این زمینه:

```
using (PersonDbEntities context = new PersonDbEntities())
{

    var people = from p in context.Persons
                 select p;

    foreach (Person person in people)
    {
        Console.WriteLine("{0}, {1}",
            person.LastName,
            person.FirstName);

        if (person is Student)
            Console.WriteLine("    Degree: {0}",
                ((Student)person).Degree);

        if (person is BusinessStudent)
            Console.WriteLine("    Discipline: {0}",
                ((BusinessStudent)person).Discipline);
    }

    Console.ReadLine();
}
```

مزایای روش TPT

امکان نرمال سازی سطح 3 در این روش به خوبی وجود دارد
افزونگی در جداول وجود ندارد.
اصلاح مدل آسان است (برای اضافه یا حذف کردن یک موجودیت به/از مدل فقط کافی است تا جدول متناظر با آن را از پایگاه داده حذف کنید)

معایب روش TPT

سرعت عملیات CRUD (ایجاد، بازیابی، آپدیت، حذف) داده‌ها با افزایش تعداد موجودیت‌های شرکت کننده در سلسله مراتب ارث بری کاهش می‌یابد. به عنوان مثال، کوئری‌های SELECT، حاوی عبارت‌های JOIN خواهند بود و عدم توجه صحیح به کوئری نوشته شده می‌تواند منجر به حضور چندین عبارت JOIN که برای ارتباط بین جداول به کار می‌رود در اسکریپت تولیدی و کاهش زمان اجرای بازیابی داده‌ها شود.
تعداد جداول در پایگاه داده زیاد می‌شود

در قسمت بعد با روش TPH آشنا می‌شوید.

نظرات خوانندگان

نویسنده: سید امیر سجادی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۲۳:۲۷

با سلام
به نظر شما برای پروژه‌های بزرگ اگه از LINQ TO SQL استفاده بشه کارایی و سرعت رو پایین می‌آره یا نه؟

نویسنده: مهمان
تاریخ: ۱۳۹۲/۰۲/۱۹ ۱۲:۶

سلام آقای راد

ضمن تشکر بابت مطالبی که نوشتید راستی کتاب مرجع کامل EF 6 کی به بازار میاد؟

نویسنده: بهروز راد
تاریخ: ۱۳۹۲/۰۲/۲۰ ۹:۲۴

از LINQ To Entities و Entity Framework استفاده کنید. بهینه‌تر هست.

نویسنده: بهروز راد
تاریخ: ۱۳۹۲/۰۲/۲۰ ۹:۲۵

حدوداً یک ماه پس از انتشار نسخه‌ی نهایی EF 6.0 توسط مایکروسافت.

نویسنده: کوروش شاهی
تاریخ: ۱۳۹۳/۰۲/۲۴ ۱۷:۱۴

سلام
هنوز کتاب انتشار نیافته است ؟