

## فهرست مطالب

۲	راهنمای گام به گام پیاده سازی سرویس برای موجودیت جدید .....
۲	مرحله ی اول : آماده کردن متادیتای موجودیت .....
۲	ساختار دیتابیزی جدول نویسنده .....
۲	ساختار دیتابیزی جدول کتاب .....
۲	مرحله ی دوم : تولید فایل های مورد نیاز برای موجودیت نویسنده .....
۴	مرحله ی سوم : تولید فایل های مورد نیاز برای موجودیت کتاب .....
۶	مرحله ی چهارم : انجام دستی تغییرات مورد نیاز .....
۶	اضافه کردن کلاس های تبدیل دیتابیزی به EF Core .....
۶	معرفی کلاس پیاده سازی لایه دیتابیزی در کلاس TypeContainer .....
۷	اضافه کردن امکان تبدیل مدل های اطلاعاتی و نمایشی به یکدیگر .....
۷	اضافه کردن ارتباطات دیتابیزی یک به چند (one-to-many) .....
۹	تغییر سورس برای ایجاد لاگ های عملیاتی .....
۱۰	مرحله ی پنجم : تولید اسکریپت های دیتابیزی برای متادیتا .....

## راهنمای گام به گام پیاده سازی سرویس برای موجودیت جدید

### مرحله اول : آماده کردن متادیتای موجودیت

چون به فایل های متعددی در لایه های مختلف نیاز داریم، لازم است متادیتای موجودیت جدید را در بخش "Entities" در ابزار طراح سیستم (System Designer) وارد کنیم تا بتوانیم به کمک ابزارهای دیگر، کلاس ها و اسکریپت های مورد نیاز را تولید کنیم. در گام های بعدی فرض بر این است که متادیتای مورد نیاز را برای موجودیت های آزمایشی "کتاب" و "نویسنده"، طبق طراحی زیر، تعریف کرده ایم.

#### ساختار دیتابیزی جدول نویسنده

ردیف	نام فیلد	نوع داده ای	اجباری / اختیاری	شرح فیلد
۱	AuthorID	INT	NOT NULL	شناسه نویسنده (کلید اصلی)
۲	FirstName	NCARCHAR(32)	NOT NULL	نام نویسنده
۳	LastName	NCARCHAR(32)	NOT NULL	نام خانوادگی نویسنده
۴	Bio	NCARCHAR(1024)	NULL	شرح یا زندگینامه کوتاه نویسنده
۵	SiteUrl	NCARCHAR(128)	NULL	آدرس سایت اینترنتی نویسنده
۶	Twitter	NCARCHAR(64)	NULL	شناسه توییتر نویسنده
۷	Instagram	NCARCHAR(64)	NULL	شناسه اینستاگرام نویسنده

#### ساختار دیتابیزی جدول کتاب

ردیف	نام فیلد	نوع داده ای	اجباری / اختیاری	شرح فیلد
۱	BookID	INT	NOT NULL	شناسه کتاب (کلید اصلی)
۲	AuthorID	INT	NOT NULL	شناسه نویسنده
۳	Title	NVARCHAR(128)	NOT NULL	عنوان کتاب
۴	Genre	NVARCHAR(64)	NOT NULL	نوع داستانی یا غیرداستانی کتاب
۵	Publisher	NVARCHAR(64)	NOT NULL	نام ناشر
۶	Isbn	NVARCHAR(16)	NULL	شماره بین المللی کتاب (ISBN 13)
۷	Language	NVARCHAR(16)	NOT NULL	زبان کتاب
۸	ReleaseDate	DATE	NULL	تاریخ انتشار کتاب
۹	PageCount	INT	NOT NULL	تعداد صفحات کتاب
۱۰	Discontinued	BIT	NOT NULL	آخرین چاپ کتاب تمام شده یا نه؟

### مرحله دوم : تولید فایل های مورد نیاز برای موجودیت نویسنده

برای صرفه جویی در زمان، می توانیم کلاس ها و اسکریپت های مورد نیاز را به کمک ابزار طراح سیستم تولید کنیم. چون موجودیت کتاب به موجودیت نویسنده وابسته است، بهتر است ابتدا کلاس های مربوط به نویسنده را تولید کنیم. برای این کار مراحل زیر را به ترتیب انجام می دهیم :

۱. ابزار داخلی طراح سیستم را اجرا کرده و از منوی "Wizards" روی گزینه‌ی "CRUD Manager Wizard" کلیک می‌کنیم.

۲. در صفحه‌ی اول فرم چندمرحله‌ای (ویزارد) اطلاعات اولیه را برای موجودیت مورد نظر وارد می‌کنیم. از کامبوباکس "نام موجودیت" گزینه‌ی "Author" را برای موجودیت نویسنده انتخاب می‌کنیم. برای درست و کامل بودن مستندات کلاس‌ها، ویژگی‌ها و توابع در کدهای تولیدشده، لازم است نام فارسی موجودیت را به صورت مفرد و جمع وارد کنیم. گزینه‌های بعدی در صفحه‌ی اول فرم، مربوط به برنامه‌ی تدبیر است و برای موجودیت‌های آزمایشی این راهنما کاربردی ندارد. توضیح این دو گزینه به صورت زیر است :

- وابستگی موجودیت به شعبه و دوره‌ی مالی : اکثر موجودیت‌ها در برنامه‌ی تدبیر به شعبه و دوره‌ی مالی وابسته هستند. هنگام تولید کد برای این موجودیت‌ها، لازم است این گزینه انتخاب شده باشد.
- سیستمی بودن موجودیت : در برنامه‌ی تدبیر، اطلاعات موجودیت‌هایی که مستقل از شرکت‌ها هستند، در دیتابیس سیستمی برنامه نگهداری و مدیریت می‌شوند، مثل کاربران و نقش‌ها. برای این نوع موجودیت‌ها لازم است این گزینه انتخاب شده باشد.

روی دکمه‌ی بعدی کلیک کنید.

۳. در صفحه‌ی دوم می‌توانیم فایل‌هایی را که می‌خواهیم توسط ویزارد تولید شوند انتخاب کنیم. به طور پیش فرض، اکثر گزینه‌های موجود انتخاب شده‌اند. دو گزینه‌ی آخر مربوط به تولید کلاس‌های تایپ اسکریپت هستند و در حال حاضر حتی در صورت انتخاب، کدی برای آنها توسط ویزارد تولید نمی‌شود.

چون برای این راهنما نیازی به مدیریت اطلاعات نویسنده در دیتابیس نداریم، لازم است گزینه‌های زیر انتخاب نشده باشند :

- Generate API Controller
- Generate Repository Interface
- Generate Repository Implementation
- Generate API Routing
- Generate Permissions Enum
- Generate TypeScript View Model
- Generate TypeScript API Routing

۴. روی دکمه‌ی "Finish" کلیک کنید. فایل‌های مورد نیاز برای استفاده از موجودیت نویسنده در عملیات مربوط به کتاب تولید می‌شوند. فایل‌های زیر باید در مسیرهای درست ایجاد شده باشند :

- کلاس مدل اطلاعاتی نویسنده در فایل `Author.Generated.cs`، در مسیر جاری پروژه مدل‌های اطلاعاتی و در پوشه‌ی همنام با حوزه (Area) تعریف‌شده برای موجودیت که برای موجودیت‌های این راهنما، پوشه `Core` خواهد بود.
- کلاس مدل نمایشی نویسنده در فایل `AuthorViewModel.Generated.cs`، در مسیر جاری پروژه مدل‌های نمایشی موجودیت و در پوشه‌ی `Core` قرار می‌گیرد.
- کلاس مورد نیاز برای تبدیل ساختار دیتابیزی موجودیت به ساختار شیء‌گرای آن و بالعکس که برای پیاده‌سازی عملیات دیتابیزی توسط `EF Core` ضروری است. این کلاس در فایل `AuthorMap.Generated.cs`، در مسیر جاری پروژه‌ی لایه‌ی دیتابیزی و در پوشه‌ی `Mapping\Core` قرار می‌گیرد.
- اسکریپت دیتابیزی مورد نیاز برای ساختن جدول موجودیت داخل دیتابیس در فایل `CreateDbObjects.Generated.sql` و در پوشه‌ی `_res\_codegen` قرار می‌گیرد.

۵. ابزار طراح سیستم را ببندید و برای اطمینان از تولید صحیح کد برای کلاس‌های جدید، یک بار پروژه‌های برنامه را به کمک دستور `Build Solution` کامپایل کنید. تمام پروژه‌ها باید بدون خطا کامپایل شوند.

۶. اسکریپت دیتابیزی تولیدشده برای نویسنده را در یکی از دیتابیس‌های شرکتی موجود اجرا کنید تا جدول نویسنده ساخته شود.

### مرحله‌ی سوم : تولید فایل‌های مورد نیاز برای موجودیت کتاب

۱. ابزار طراح سیستم را اجرا کرده و از منوی "Wizards" روی گزینه‌ی "CRUD Manager Wizard" کلیک می‌کنیم.

۲. در صفحه‌ی اول ویزارد از کامبوباکس نام موجودیت، گزینه‌ی `Book` را انتخاب می‌کنیم. نام فارسی موجودیت را به صورت مفرد و جمع وارد می‌کنیم. پیش از رفتن به صفحه‌ی بعدی ویزارد، از انتخاب نشدن سایر گزینه‌ها اطمینان حاصل می‌کنیم. روی دکمه‌ی بعدی کلیک می‌کنیم.

۳. چون می‌خواهیم سرویس جدیدی برای مدیریت اطلاعات کتاب‌ها اضافه کنیم، لازم است در صفحه‌ی دوم ویزارد تمام گزینه‌ها انتخاب شده باشند. توضیح کوتاه گزینه‌های صفحه‌ی دوم به صورت زیر است :

- گزینه‌ی `Generate API Controller` : این گزینه در سه حالت مختلف قابل استفاده است. وقتی فقط همین گزینه به‌تنهایی انتخاب شده باشد، کلاس کنترلر مورد نیاز برای پیاده‌سازی عملیات اصلی سرویس با کمترین امکانات تولید می‌شود. در این حالت باید تمام عملیات را خودمان به صورت دستی پیاده‌سازی کنیم.

- گزینه‌ی Starter CRUD Methods : در صورت انتخاب این گزینه، متدهای مورد نیاز بدون پیاده‌سازی به کلاس کنترلر اضافه می‌شوند. در این حالت نیز لازم است تمام عملیات را به صورت دستی پیاده‌سازی کنیم.
- گزینه‌ی Starter CRUD Implementation : در صورت انتخاب این گزینه، تمام متدها با پیاده‌سازی استاندارد به کنترلر سرویس جدید اضافه می‌شوند.
- گزینه‌ی Generate Model : برای تولید کلاس مدل اطلاعاتی موجودیت، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate View Model : برای تولید کلاس مدل نمایشی موجودیت، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate Db Mapping : برای تولید کلاس تبدیل ساختار اطلاعاتی به ساختار شیء‌گرا، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate CREATE TABLE Script : برای تولید اسکریپت دیتابیزی مورد نیاز برای ایجاد جدول موجودیت در دیتابیس، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate Repository Interface : برای تولید اینترفیس استاندارد لایه‌ی دیتابیزی موجودیت جدید، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate Repository Implementation : برای تولید کلاس استاندارد لایه‌ی دیتابیزی موجودیت جدید، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate API Routing : برای تولید کلاس کمکی مورد نیاز کلاس کنترلر که آدرس متدهای سرویس را تعریف می‌کند، از این گزینه استفاده می‌کنیم.
- گزینه‌ی Generate Permissions Enum : برای تولید نوع شمارشی مورد نیاز کلاس کنترلر که دسترسی‌های امنیتی به عملیات را تعریف می‌کند، از این گزینه استفاده می‌کنیم.

هر گاه گزینه‌ی Generate API Routing انتخاب شده باشد، می‌توانیم در صفحه‌ی سوم ویزارد، عملیاتی را که نیاز به کنترل امنیتی دارند تعریف کنیم. چون در این راهنما گزینه را انتخاب کردیم، روی دکمه‌ی بعدی کلیک می‌کنیم.

۴. در صفحه‌ی سوم ویزارد می‌توانیم عملیات مورد نیاز برای موجودیت جدید را معرفی کنیم. به طور پیش‌فرض، هنگام اولین ورود به این صفحه تمام عملیات انتخاب شده‌اند. چون این عملیات برای بیشتر موجودیت‌ها مورد نیاز و به نوعی در برنامه‌ی تدبیر استاندارد هستند، معمولاً لازم است تمام گزینه‌ها انتخاب شده باشند. اگر برای موجودیت جدید عملیات خاص دیگری مورد نیاز باشد، می‌توانیم در لیست پایین فرم آنها را یکی یکی اضافه کنیم.

برای این راهنما، عملیات استاندارد کافی هستند و می‌توانیم بدون تغییر گزینه‌ها مرحله‌ی نهایی ویزارد را اجرا کنیم. برای این کار روی دکمه‌ی "Finish" کلیک می‌کنیم.

۵. ابزار طراح سیستم را ببندید و برای اطمینان از تولید صحیح کد برای کلاس‌های جدید، یک بار پروژه‌های برنامه را به کمک دستور Build Solution کامپایل کنید. تمام پروژه‌ها باید بدون خطا کامپایل شوند.

۶. اسکریپت دیتابیزی تولیدشده برای کتاب را در دیتابیس انتخاب‌شده برای نویسنده اجرا کنید تا جدول کتاب ساخته شود.

### مرحله‌ی چهارم : انجام دستی تغییرات مورد نیاز

ویزارد مورد استفاده در مراحل دوم و سوم، تا حد امکان کدهای اولیه را بدون خطا تولید می‌کند. در این مرحله لازم است تغییرات تکمیلی را در فایل‌های تولیدشده به صورت دستی اعمال کنیم. در گام‌های بعدی این مرحله، تغییرات مورد نیاز را یک به یک توضیح خواهیم داد.

### اضافه کردن کلاس‌های تبدیل دیتابیزی به EF Core

در پروژه‌ی لایه‌ی دیتابیزی (SPPC.Tadbir.Persistence) وارد پوشه Context می‌شویم. لازم است کلاس‌های تولیدشده برای نویسنده و کتاب (کلاس‌های AuthorMap و BookMap) را در کلاس محتوایی دیتابیس شرکتی (TadbirContext) ثبت کنیم. لازم به یادآوری است که اگر موجودیت جدید از نوع سیستمی باشد، لازم است آن را در کلاس محتوایی دیتابیس سیستمی (SystemContext) ثبت کنیم. برای این کار وارد متد OnModelCreating می‌شویم و دستورات زیر را به آن اضافه می‌کنیم :

```
AuthorMap.BuildMapping(modelBuilder.Entity<Author>());  
BookMap.BuildMapping(modelBuilder.Entity<Book>());
```

نکته : چون دستورات مشابهی به تدریج برای موجودیت‌های دیگر اضافه می‌شوند، برای اینکه بتوانیم به سرعت بودن یا نبودن دستورات مورد نیاز را کنترل کنیم، بهتر است هنگام اضافه کردن این دستورات آنها را با توجه به ترتیب حروف انگلیسی در جای مناسب قرار دهیم، به صورتی که در نهایت این دستورات به صورت صعودی مرتب (sort) شده باشند.

### معرفی کلاس پیاده‌سازی لایه دیتابیزی در کلاس TypeContainer

در پروژه‌ی سرویس وب (SPPC.Tadbir.Web.Api) وارد کلاس TypeContainer می‌شویم و دستور زیر را در انتهای پیاده‌سازی متد AddPersistenceTypes اضافه می‌کنیم :

```
_services.AddTransient<IBookRepository, BookRepository>();
```

اضافه کردن امکان تبدیل مدل های اطلاعاتی و نمایشی به یکدیگر

در پروژه نگاشت مدل های اطلاعاتی (SPPC.Tadbir.Mapper) وارد کلاس DomainMapper می شویم و با توجه به حوزه (Area) انتخاب شده هنگام ایجاد متادیتای موجودیت، در متد مناسب دستورات مورد نیاز را اضافه می کنیم. برای موجودیت های این راهنما، نام متد مورد نظر MapCoreTypes خواهد بود. دستورات زیر را به انتهای پیاده سازی این متد اضافه می کنیم :

```
mapperConfig.CreateMap<Book, BookViewModel>();  
mapperConfig.CreateMap<BookViewModel, Book>();
```

اضافه کردن ارتباطات دیتابیزی یک به چند (one-to-many)

برای اینکه بتوانیم ارتباط صحیحی بین موجودیت های کتاب و نویسنده برقرار کنیم، لازم است شناسه ی دیتابیزی موجودیت پایه (نویسنده) را به موجودیت وابسته (کتاب) اضافه کنیم. این ارتباط هنگام تولید کلاس مدل اطلاعاتی کتاب، به صورت یک رفرنس به مدل اطلاعاتی نویسنده برقرار شده است. شناسه دیتابیزی نویسنده هم در مدل اطلاعاتی (Book) و هم در مدل نمایشی (BookViewModel) کتاب مورد نیاز است. بنابراین لازم است ویژگی جدیدی به نام AuthorId به هر دو کلاس اضافه کنیم.

نکته ی ۱ : چون کلاس های مدل اطلاعاتی و مدل نمایشی توسط ویزارد تولید شده اند، بنا به قرارداد رایج در پروژه های مشابه، لازم است فایل جدیدی برای هر کلاس ایجاد کنیم. در واقع وقتی بعضی از فایل های پروژه توسط ابزارها تولید می شوند، لازم است هر گونه تغییرات دستی در فایل های جدید انجام شود تا در صورت تولید دوباره ی کلاس ها با ابزار، تغییرات دستی از بین نروند.

\* در پروژه ی مدل های اطلاعاتی (SPPC.Tadbir.Model) داخل پوشه ی مناسب (برای این راهنما، پوشه ی Core) از طریق تمپلیت کلاس جدید در ویژوال استودیو (Add New Class) فایل جدیدی به نام Book.cs اضافه می کنیم. در سورس کلاس ایجاد شده، پیش از کلمه ی کلیدی class عبارت public partial را اضافه می کنیم تا خطای کامپایلری برطرف شود. سپس ویژگی جدیدی برای شناسه نویسنده از نوع عدد صحیح و با قابلیت خواندن و نوشتن مقدار ویژگی (read/write) اضافه می کنیم. نتیجه ی نهایی به صورت زیر می شود :

```
public partial class Book  
{  
    /// <summary>  
    /// شناسه دیتابیزی نویسنده این کتاب  
    /// </summary>
```

```

    public virtual int AuthorId { get; set; }
}

```

نکته ۲ : در کد بالا کلمه‌ی کلیدی **virtual** در واقع برای تعریف ویژگی ضروری نیست. ولی چون ابزارهای تولید کد از این مشخصه برای کلاس‌های مدل استفاده می‌کنند، بهتر است برای حفظ یکدستی با کدهای موجود به همین شکل کار کنیم.

\* مشابه گام قبلی، در پروژه‌ی مدل‌های نمایشی (**SPPC.Tadbir.ViewModel**) فایل جدیدی به نام **BookViewModel.cs** اضافه می‌کنیم. سپس ویژگی جدیدی برای شناسه‌ی نویسنده اضافه می‌کنیم. نتیجه‌ی نهایی به صورت زیر می‌شود :

```

public partial class BookViewModel
{
    /// <summary>
    /// شناسه دیتابسی نویسنده این کتاب
    /// </summary>
    public int AuthorId { get; set; }
}

```

نکته ۳ : برای تعریف ویژگی‌ها در مدل‌های نمایشی، نیازی به استفاده از کلمه‌ی کلیدی **virtual** نیست.

\* پس از اضافه کردن شناسه‌ی دیتابسی نویسنده در مدل‌های کتاب، لازم است ارتباط مورد نیاز **EF Core** را در کلاس نگاشت دیتابسی معرفی کنیم. برای این کار در پروژه‌ی لایه‌ی دیتابسی (**SPPC.Tadbir.Persistence**) ابتدا وارد پوشه‌ی **Mapping** و سپس وارد پوشه‌ی **Core** می‌شویم و در انتهای کلاس **BookMap** دستور **HasForeignKey** را با دستور زیر جایگزین می‌کنیم :

```

.HasForeignKey(e => e.AuthorId)

```

\* در گام آخر این مرحله، لازم است امکان تغییر نویسنده را در کلاس لایه‌ی دیتابسی کتاب فراهم کنیم. برای این کار، در پروژه‌ی لایه‌ی دیتابسی وارد پوشه‌ی **Repository** شده و کلاس **BookRepository** را باز می‌کنیم. در متد **UpdateExisting** که برای کپی تغییرات از مدل نمایشی به مدل اطلاعاتی فراخوانی می‌شود، دستور زیر را اضافه می‌کنیم :



```
book.AuthorId = bookViewModel.AuthorId;
```

پس از انجام تغییرات دستی که در گام‌های ۱ تا ۴ توضیح دادیم، امکان انجام عملیات اصلی CRUD در دیتابیس برای موجودیت کتاب فراهم می‌شود. تا پیش از تکمیل مراحل بعدی، امکانات استاندارد دیگر (مثل کنترل دسترسی امنیتی، ایجاد لاگ‌های عملیاتی، گزارش فوری و ارسال به اکسل) هنوز عملیاتی نخواهند بود. در این مرحله می‌توانیم درستی عملکرد سرویس برای عملیات استاندارد دیتابیزی را در یکی از برنامه‌های HTTP Client (مانند Chrome PostMan و Insomnia) آزمایش کنیم.

### تغییر سورس برای ایجاد لاگ‌های عملیاتی

در این گام مقدمات ایجاد لاگ‌های عملیاتی را در لایه دیتابیزی فراهم می‌کنیم. این کار در دو مرحله انجام می‌شود.

\* برای اضافه کردن امکان چندزبانه به شرح لاگ‌های عملیاتی، لازم است نام فیلدهای اطلاعاتی موجودیت اصلی (کتاب) را در ریسورس زبان‌های مختلف برنامه اضافه کنیم. این کار را در فایل‌های AppStrings.resx و AppStrings.en.resx از پروژه‌ی اصلی ریسورس‌ها (SPPC.Tadbir.Resources) انجام می‌دهیم. پس از تکمیل این مرحله، موارد اضافه‌شده به ریسورس‌ها مطابق شکل‌های زیر می‌شود:

Widget	ویجت
WidgetAccess	دسترسی به ویجت
ZeroDebitAndCreditNotAllowed	مبلغ بدهکار و بستانکار صفر در آرتیکل نادرست است.
Genre	دسته بندی
Publisher	ناشر
Isbn	شماره بین المللی
Language	زبان
ReleaseDate	تاریخ انتشار
PageCount	تعداد صفحات
Discontinued	اتمام چاپ
*	

WeekX	Week {0}
Widget	Widget
WidgetAccess	Widget access
ZeroDebitAndCreditNotAllowed	Zero amount in both debit and credit is not allowed.
Genre	Genre
Publisher	Publisher
Isbn	ISBN
Language	Language
ReleaseDate	Release Date
PageCount	Page Count
Discontinued	Out of Print
*	

\* در شرح لاگ برای عملیات ایجاد، اصلاح و حذف، جزئیات سطر اطلاعاتی مورد نیاز است. یکی از متدهای مجازی تولیدشده در کلاس لایه دیتابسی، برای این منظور پیش‌بینی شده است که باید به طور دستی آن را پیاده‌سازی کنیم. متن‌های چندزبانه‌ای که برای ویژگی‌های موجودیت کتاب در مرحله‌ی قبل اضافه کردیم، در این متد مورد استفاده قرار می‌گیرند. در نمونه کد زیر می‌توانیم پیاده‌سازی مرسوم برای این متد را ببینیم :

```
protected override string GetState(Book entity)
{
    var releaseDate = entity.ReleaseDate.HasValue
        ? _config.GetDateDisplayAsync(entity.ReleaseDate.Value).Result
        : String.Empty;
    var discontinued = entity.Discontinued
        ? AppStrings.BooleanYes
        : AppStrings.BooleanNo;
    return entity != null
        ? $"{AppStrings.Title}: {entity.Title}, {AppStrings.Genre}:
{entity.Genre}, " +
        $"{AppStrings.Publisher}: {entity.Publisher}, {AppStrings.Isbn}:
{entity.Isbn}, " +
        $"{AppStrings.Language}: {entity.Language}, {AppStrings.ReleaseDate}:
{releaseDate}, " +
        $"{AppStrings.PageCount}: {entity.PageCount}, {AppStrings.Discontinued}:
{discontinued}"
        : String.Empty;
}
```

مرحله‌ی پنجم: تولید اسکریپت‌های دیتابسی برای متادیتا

#### تاریخچه تغییرات

ردیف	نسخه	تهیه کننده	تغییرات
۱	۱,۰	بابک اسلامی	نسخه اولیه (پس از چند مرحله ویرایش و تکمیل)
۲			
۳			
۴			