Elaboração de Arquiteturas de Software – FASE 2



Guia de Desenvolvimento para ORM

Preparado para

Secretaria da Fazenda do Estado de São Paulo

16-mai-22

Versão 1.1 Final

Preparado por

**Diego Baroffio**

diego.baroffio@microsoft.com

*Contribuidores*

**Carlos Dietrich**, **Ernesto Guimarães**, **Gleisson Bezerra**, **Hamilton de Paula**

Histórico de Atualizações e Revisões

Registro de Atualizações

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Autor | Versão | Descrição |
| 15/04/2013 | Caio Dietrich | 0.1 |  |
| 23/04/2013 | Diego Baroffio  Vanessa Oliveira | 0.2 |  |
| 26/04/2013 | Diego Baroffio | 0.3 |  |
| 29/04/2013 | Diego Baroffio | 0.4 |  |
| 30/04/2013 | Diego Baroffio | 0.5 |  |
| 02/05/2013 | Diego Baroffio | 0.6 |  |
| 03/05/2013 | Diego Baroffio | 0.7 |  |
| 05/05/2013 | Diego Baroffio | 0.8 |  |
| 06/05/2013 | Diego Baroffio | 1.0 |  |
| 18/06/2013 | Diego Baroffio | 1.1 |  |
|  |  |  |  |

Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| Nome | Versão Aprovada | Posição | Data |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Conteúdo

[1 Objetivo 1](#_Toc367883496)

[2 Público Alvo 2](#_Toc367883497)

[3 Requisitos de Ambiente 2](#_Toc367883498)

[4 Siglas 2](#_Toc367883499)

[5 Visão Geral de Arquitetura 4](#_Toc367883500)

[5.1 ORM & Aplicações 4](#_Toc367883501)

[5.2 O Entity Framework 7](#_Toc367883502)

[5.3 Sefaz.Fw 10](#_Toc367883503)

[6 Recomendações 12](#_Toc367883504)

[6.1 Quando Usar ORM? 12](#_Toc367883505)

[6.2 O Entity Data Model 13](#_Toc367883506)

[6.2.1 Criando o Modelo via Database First 13](#_Toc367883507)

[6.2.2 Gerando as Classes das Entidades e do Contexto 17](#_Toc367883508)

[6.2.3 Alterando o Mapeamento das Entidades 18](#_Toc367883509)

[6.2.4 Refletindo Alterações da Base de Dados no Modelo 19](#_Toc367883510)

[6.2.5 Gerando as Classes das Entidades em Outro Projeto 20](#_Toc367883511)

[6.3 Uso do Entity Framework em Aplicações Distribuídas 22](#_Toc367883512)

[6.4 Entidades e DTO’s 23](#_Toc367883513)

[6.4.1 Entidades POCO 23](#_Toc367883514)

[6.4.2 Carregando Navigation Properties 24](#_Toc367883515)

[6.4.3 Data Transfer Objects 27](#_Toc367883516)

[6.5 Stored Procedures 29](#_Toc367883517)

[6.5.1 Mapeando uma Stored Procedure 29](#_Toc367883518)

[6.6 Rastreamento de Alterações 31](#_Toc367883519)

[6.7 Concorrência 31](#_Toc367883520)

[6.7.1 Configurando as Propriedades para Concorrência 32](#_Toc367883521)

[6.7.2 Tratando Situações de Concorrência 33](#_Toc367883522)

[6.8 Desempenho 35](#_Toc367883523)

[6.8.1 .NET Framework 4.0 versus 4.5 36](#_Toc367883524)

[6.8.2 Auto-Compiled LINQ Queries 36](#_Toc367883525)

[6.8.3 Pre-generated Views 36](#_Toc367883526)

[6.8.4 Consultas Não-rastreáveis 37](#_Toc367883527)

[7 Referências 38](#_Toc367883528)

[8 Apêndice A – Modelos de Programação do EF 39](#_Toc367883529)

[9 Apêndice B – Code First 40](#_Toc367883530)

[9.1 Criando o Modelo 40](#_Toc367883531)

[9.2 Configurando o Mapeamento 41](#_Toc367883532)

[9.3 Alterando o Modelo 43](#_Toc367883533)

[10 Apêndice C – Data Providers para o Entity Framework 45](#_Toc367883534)

Índice de Figuras

[Figura 1: Arquitetura das aplicações sem ORM 4](#_Toc359418589)

[Figura 2: Arquitetura das aplicações com ORM 6](#_Toc359418590)

[Figura 3: Distribuição física de uma aplicação 7](#_Toc359418591)

[Figura 4: Integração do EF nas camadas físicas da aplicação 8](#_Toc359418592)

[Figura 5: Classes geradas pelos templates T4 17](#_Toc359418593)

[Figura 6: Templates T4 inseridos pelo modelo EDMX 17](#_Toc359418594)

[Figura 7: Propriedades do template T4 18](#_Toc359418595)

[Figura 8: Concurrency Mode 32](#_Toc359418596)

Índice de Tabelas

[Tabela 1: Tabela de requisitos de ambiente 2](#_Toc359418597)

[Tabela 2: Siglas 3](#_Toc359418598)

[Tabela 3: Tabela de decisão sobre o uso do ORM 13](#_Toc359418599)

[Tabela 4: Referências 38](#_Toc359418600)

[Tabela 5: Data Providers para o Entity Framework 46](#_Toc359418601)

Índice de Listagens

[Listagem 1: Exemplo de entidade 23](#_Toc359418630)

[Listagem 2: Exemplo de leitura usando o método Include() 24](#_Toc359418631)

[Listagem 3: Exemplo de leitura que carrega somente a *navigation property* Customer 24](#_Toc359418632)

[Listagem 4: Exemplo de SQL gerado pelo EF usando o método Include() 25](#_Toc359418633)

[Listagem 5: Exemplo de leitura que carrega as *navigation properties* Customer e Order\_Details 25](#_Toc359418634)

[Listagem 6: Exemplo de SQL gerado pelo EF carregando as *navigation properties* Customer e Order\_Details 27](#_Toc359418635)

[Listagem 7: Exemplo de DTO combinando informações de duas entidades 27](#_Toc359418636)

[Listagem 8: Exemplo de comando LINQ para ler DTO’s 28](#_Toc359418637)

[Listagem 9: Exemplo de SQL gerado pelo EF para carregar DTO’s 28](#_Toc359418638)

[Listagem 10: Exemplo de *stored procedure* 30](#_Toc359418639)

[Listagem 11: Exemplo de tipo gerado para os resultados de uma *stored procedure* 30](#_Toc359418640)

[Listagem 12: Exemplo de método gerado para chamada de uma *stored procedure* 30](#_Toc359418641)

[Listagem 13: Exemplo de atualização com os valores originais 31](#_Toc359418642)

[Listagem 14: Exemplo de atualização sem os valores originais 31](#_Toc359418643)

[Listagem 15: Exemplo de atualização com abordagem **Base de dados ganha** 33](#_Toc359418644)

[Listagem 16: Exemplo de atualização com abordagem **Cliente ganha** 34](#_Toc359418645)

[Listagem 17: Exemplo de atualização com abordagem **Customizada** 35](#_Toc359418646)

[Listagem 18: Exemplo do uso do método AsNoTracking() 37](#_Toc359418647)

[Listagem 19: Entidade de negócio Order 40](#_Toc359418648)

[Listagem 20: Entidade de Negócio Order\_Detail 40](#_Toc359418649)

[Listagem 21: Classe de contexto 41](#_Toc359418650)

[Listagem 22: Entidade Order com atributo *[Key]* 41](#_Toc359418651)

[Listagem 23: Entidade Order\_Detail com atributo *[Key]* 42](#_Toc359418652)

[Listagem 24: Exemplo de método *OnModelCreating* 42](#_Toc359418653)

[Listagem 25: Exemplo de classe de configuração para a entidade Order 42](#_Toc359418654)

[Listagem 26: Exemplo de classe de configuração para a entidade Order\_Detail 43](#_Toc359418655)

[Listagem 27: Exemplo de classe *Migration* gerada 44](#_Toc359418656)

1. Objetivo

Este documento é parte integrante do projeto de Elaboração de Arquiteturas de Software – FASE 2, desenvolvido segundo o Escopo de Trabalho e Formato de Entregas especificados no documento de Visão Escopo (**SEFAZ - ITAP Arquiteturas de Software - FASE 2 - Visão e Escopo v1.0.docx**).

O projeto visa estabelecer uma referência para o desenvolvimento padronizado de sistemas que permita o planejamento da adoção de práticas para reduzir riscos de segurança e aumentar a disponibilidade de sistemas web da instituição.

Dentro desta proposta foram priorizados os aspectos referentes as seguintes frentes de trabalho:

**APLICAÇÕES WEB**: Documento de orientação para o desenvolvimento Interfaces WEB na Plataforma Microsoft .NET utilizando Web Forms contemplando: Acessibilidade, Interface com o Usuário, Arquitetura e Segurança.

**ORM (Mapeamento Objeto-Relacional)**: Documento com recomendações para a utilização de O/RM (Object/Relational Mapping) em projetos listando prós e contras da abordagem ORM para diferentes cenários e, quando decidido pelo seu uso, fazer uma série de recomendações do uso do Entity Framework para a persistência de dados das aplicações da SEFAZ.

**TESTES**: Documento de orientação para a execução de testes com Visual Studio 2012 considerando: Uso do Microsoft Fakes, Compatibilização do Moles com .NET 4.5, Recursos de automação do Visual Studio para testes de carga e de longa duração e Elaboração de infraestrutura reutilizável para testes de carga utilizando os serviços de testes do Visual Studio.

**GOVERNANÇA**: Criação de documento compreendendo: Disciplina de Desenho de Arquitetura e Revisão alinhado com a fase de Desenho da Solução e Acompanhamento do CPPR, Papéis para execução da Disciplina de Arquitetura e Administração dos artefatos gerados pela fase 1 e fase 2 (Guias, Templates e POC)

Este guia diz respeito à frente de trabalho de ORM e tem como objetivo orientar o desenvolvedor da plataforma Microsoft .Net com recomendações para a utilização de O/RM (Object/Relational Mapping) no design de seus projetos listando prós e contras da abordagem ORM para diferentes cenários e, caso se decida por ela, fazer uma série de recomendações no uso do produto Entity Framework para a persistência de dados das aplicações da SEFAZ.

1. Público Alvo

O público alvo deste guia são todos os desenvolvedores e arquitetos da SEFAZ envolvidos com a criação de sistemas na plataforma .Net 4.0 e 4.5, utilizando o C# como linguagem de programação.

1. Requisitos de Ambiente

Este tópico apresenta os requisitos de ambiente necessários para o acompanhamento deste guia.

|  |  |
| --- | --- |
| Recurso | Descrição |
| Visual Studio 2012 | Ambiente de desenvolvimento integrado Microsoft para a plataforma .NET |
| .NET Framework 4.5 | *Runtime* de execução .NET |
| Entity Framework 5.0 | Ferramenta ORM da Microsoft. |

Tabela 1: Tabela de requisitos de ambiente

1. Siglas

|  |  |
| --- | --- |
| Sigla | Descrição |
| O/RM ou ORM | **O**bject/**R**elational **M**apping.  Ferramenta ou framework para mapeamento entre o mundo orientado a objeto e o mundo relacional. |
| LINQ | **L**anguage-**In**tegrated **Q**uery.  Fornece funcionalidades de consulta a diversas estruturas de dados de forma integrada à linguagem de programação, oferecendo inclusive a compilação prévia das consultas. |
| EF | **E**ntity **F**ramework.  Conjunto de classes e ferramentas ORM da Microsoft. |
| POCO | **P**lain-**o**ld **C**LR **O**bject.  Uma classe POCO é uma classe que não herda funcionalidade de nenhum framework ou tecnologia. É uma simples classe pública que contém propriedades e métodos. Estes métodos não implementam nenhuma lógica de persistência como salvar e ler dados de uma base. Por este motivo estas classes são chamadas de Agnósticas de Persistência. |
| SoC | **S**eparation **o**f **C**oncerns.  Separação de Conceitos, ou ainda Separação de Responsabilidades, que é um princípio de design para a separação de uma aplicação em blocos distintos, onde cada bloco tem um objetivo e uma responsabilidade bem definidos. |
| DTO | **D**ata **T**ransfer **O**bject.  Classes cuja principal responsabilidade é trafegar dados entre as camadas da aplicação. Possuem apenas propriedades. |
| T4 | **T**ext **T**emplate **T**ransformation **T**oolkit  Um template de texto T4 é uma mistura de blocos de texto e lógica de controle que pode gerar um arquivo texto. A lógica de controle é escrita na forma de fragmentos de código em C# ou Visual Studio. |

Tabela 2: Siglas

1. Visão Geral de Arquitetura
   1. ORM e as Aplicações

Atualmente na SEFAZ, as aplicações corporativas seguem uma arquitetura distribuída em camadas lógicas e físicas:

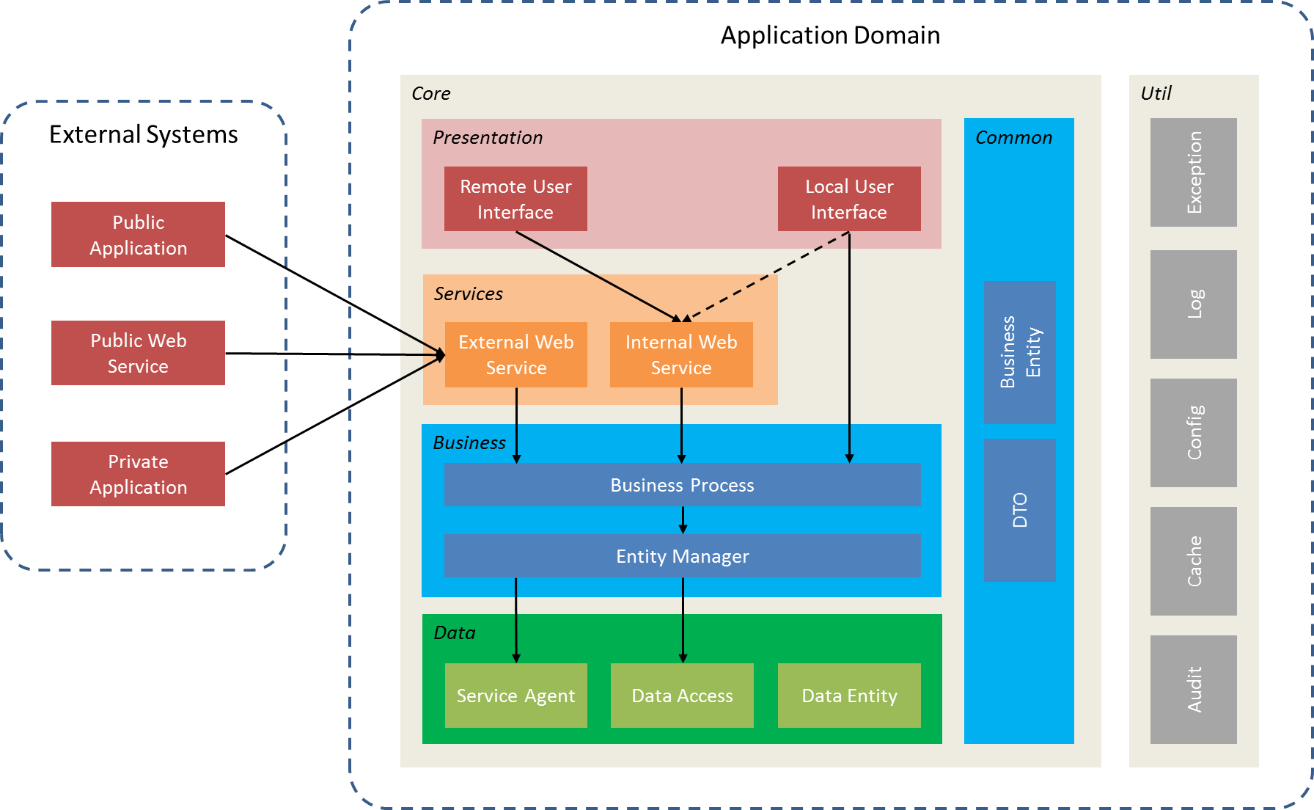


Figura 1: Arquitetura das aplicações sem ORM

Esta arquitetura multicamadas leva em consideração a separação de responsabilidades (SoC) de cada camada, assim como encapsulamento e desacoplamento de tecnologias usadas.

Normalmente, o acesso à base de dados da aplicação é de responsabilidade do bloco *Data Access*. Nele deve-se tentar encapsular todo e qualquer uso de tecnologia de persistência de dados. Isto ajuda em um cenário de evolução das tecnologias de persistência e também nos testes unitários, onde toda esta camada pode ser substituída por simuladores, por exemplo, usando o Microsoft Fakes Framework.

O bloco *Data Access* repassa para as camadas superiores apenas *Business Entities* e/ou DTO’s, ou seja, é responsável por todo tipo de conversão entre objetos *Business Entities* e estruturas para persistência em banco de dados, e é aqui que o ORM entra em cena.

De fato, a maior parte das aplicações escritas hoje em dia usam um banco relacional para armazenamento de dados. Essa tecnologia é vigente no mercado, por sua maturidade e sua fundamentação em uma teoria matemática forte (baseada em conjuntos) e padronização. Quando usamos o modelo relacional, o objetivo é a normalização e a otimização de recursos no armazenamento de nossos dados ou coleções.

Da mesma forma, a maior parte das aplicações escritas hoje em dia usam linguagens de programação orientada a objetos em seu desenvolvimento, onde o objetivo é a modelagem de processos, como a definição de comportamentos e tipos de entidades na forma de classes.

Surge assim um problema chamado "*Impedance Mismatch*", que é a dificuldade em realizar o mapeamento entre os dois mundos, o mundo relacional e o mundo orientado a objetos.

Para este problema, uma solução é o uso de uma camada de tradução ou ORM – (Mapeamento Objeto-Relacional), que é responsável por esse mapeamento entre o mundo OO e o mundo relacional. O grande desejo é que essa camada de mapeamento ORM seja transparente para nossas aplicações.

A figura a seguir mostra a arquitetura das aplicações corporativas da SEFAZ usando ORM:

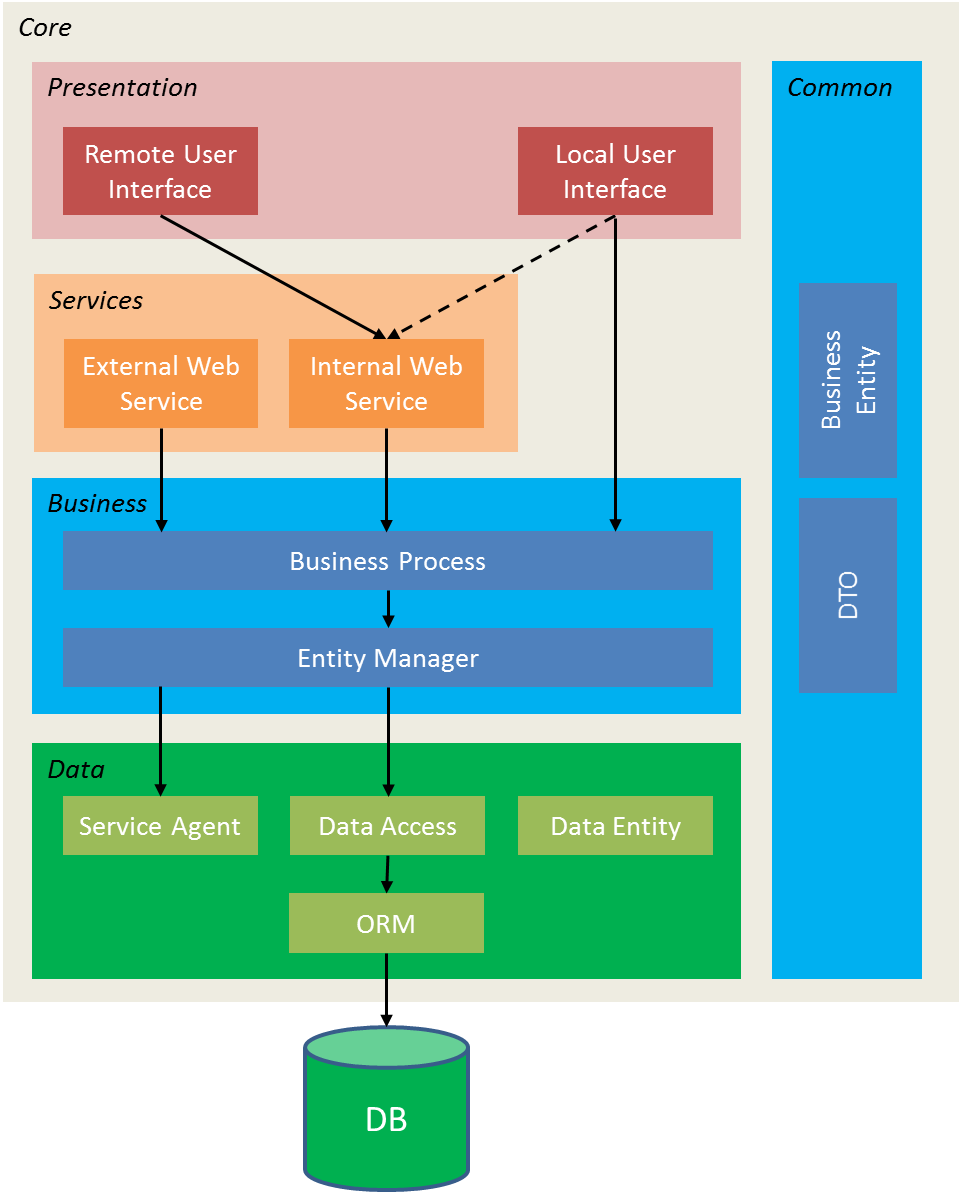


Figura 2: Arquitetura das aplicações com ORM

Um dos maiores benefícios do ORM é o ganho de produtividade da equipe de desenvolvimento, tanto na criação da aplicação quanto na sua manutenção. Ele gera automaticamente a maioria do código de acesso a dados (SQL) que você deveria escrever e manter. Além disso, objetos são automaticamente criados para você representando os dados relacionais da base de dados.

O ORM, como ferramenta de mapeamento, ajuda a isolar o esquema da base de dados do modelo de entidades da aplicação, minimizando o impacto de manutenção quando um dos dois passar por uma eventual alteração/evolução.

A integração do ORM com o .NET ainda traz outro grande benefício: o LINQ (Consulta Integrada à Linguagem). O LINQ fornece funcionalidades de consulta a diversas estruturas de dados de forma integrada à linguagem de programação, oferecendo inclusive a compilação prévia das consultas. Ao se escrever uma consulta usando LINQ, o ORM faz a tradução para a sintaxe SQL específica do banco de dados utilizado, diminuindo bastante a necessidade de conhecimento específico de SQL, já considerando aspectos como a performance, índices e plano de execução.

* 1. O Entity Framework

O Entity Framework (EF) é uma ferramenta de mapeamento objeto-relacional da Microsoft que permite que desenvolvedores .NET trabalhem com dados relacionais usando objetos específicos de domínio. Todas as recomendações deste documento estão baseadas no **Entity Framework 5.0**.

Para integrar o EF à arquitetura atual da SEFAZ, deve-se considerar que as aplicações da SEFAZ são baseadas em serviços e ficam fisicamente distribuídas em camadas como: Apresentação, Serviços, Negócio, Dados e Comum. Isto permite um reaproveitamento dos componentes por mais de uma aplicação, além de oferecer maior segurança ao manter as camadas de Negócio e de Dados apenas dentro de sua intranet.

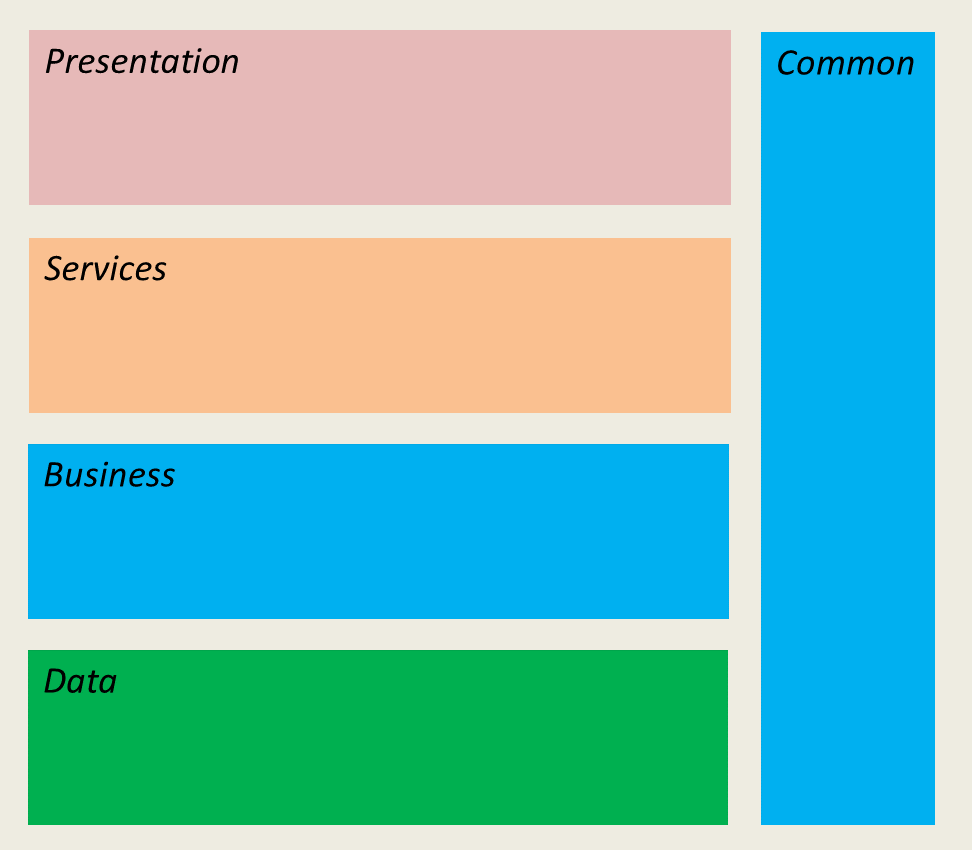


Figura 3: Distribuição física de uma aplicação

A inclusão do Entity Framework nesta arquitetura deve preservar todas as características e benefícios obtidos com a mesma.

Para atender a arquitetura de aplicações distribuídas recomendada pela SEFAZ e maximizar os benefícios que as funcionalidades do Entity Framework 5.0 oferecem, recomenda-se que ele seja parcialmente encapsulado pela camada de dados, expondo apenas os objetos entidades gerados por ele para as outras camadas. Todos os outros componentes relacionados ao EF devem ser referenciados e usados apenas pela camada de dados.

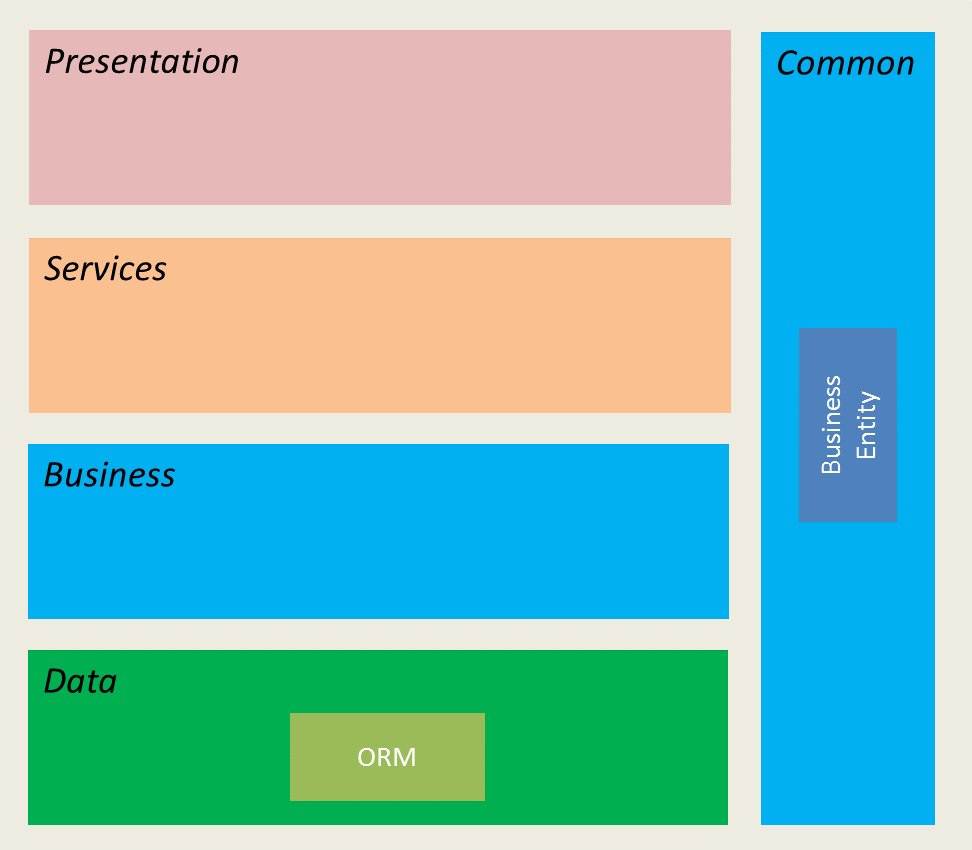


Figura 4: Integração do EF nas camadas físicas da aplicação

Esta abordagem traz as seguintes vantagens:

* É aderente ao conceito de separação de responsabilidades, não expondo componentes do EF para outras camadas e Nem requer referência ao mesmo.
* Todo o EF pode ser substituído. Isto pode ser importante no momento dos testes unitários, onde o EF pode ser substituído por um simulador; ou até mesmo no caso da evolução do EF através de novas versões.
* Não é necessária a criação de objetos adicionais para representar as entidades de negócio, uma vez que pode-se usar as entidades geradas pelo próprio EF.
* Composição de consultas via LINQ fora da camada de dados.

O Entity Framework ainda minimiza a dependência entre ele e a camada de negócio ao oferecer a possibilidade de trabalhar com classes POCO (Plain-old CLR Object) para suas entidades. Uma classe POCO é uma classe que não herda funcionalidade de nenhum framework ou tecnologia. É uma simples classe pública que contém propriedades e métodos. Estes métodos não implementam nenhuma lógica de persistência como salvar e ler dados de uma base. Por este motivo estas classes são chamadas de “Agnósticas de Persistência”.

* 1. ORM e o Sefaz.Fw

Na arquitetura proposta para o Sefaz.Fw, o acesso a dados é feito através da camada Sefaz.Fw.Data. Para exemplificar o funcionamento dessa camada, na prova de conceito foi desenvolvida uma classe base, abstrata e genérica, chamada *CRUDDataAccessBase*:

public abstract class CRUDDataAccessBase<BE> where BE : class, new()

{

private ICRUDProvider<BE> \_daProvider = null;

protected CRUDDataAccessBase(string name,

string configSectionName="dataAccessConfiguration") {...}

public BE Insert(BE entity) {...}

public virtual List<BE> Insert(List<BE> entities) {...}

public BE Update(BE entity) {...}

public virtual List<BE> Update(List<BE> entities) {...}

public void Delete(BE entity) {...}

public virtual void Delete(List<BE> entities) {...}

public virtual BE GetById(BE entity) {...}

public virtual QueryResult<BE> Get(QueryParams<BE> queryParams) {...}

public virtual List<BE> GetAll(){...}

}

Listagem 1: Classe CRUDDataAccessBase

Esta classe deve ser estendida pela aplicação cliente e fornece as operações básicas de CRUD (Create, Read, Update e Delete) para entidades POCO. O parâmetro genérico *BE* define o tipo da entidade que será utilizada nas operações da classe derivada.

Para o acesso efetivo aos dados, a classe *CRUDDataAccessBase* utiliza uma interface chamada *ICRUDProvider*, o que permite que diferentes implementações (ou *providers*) sejam utilizados. (O único requisito para a criação desse *provider* é que a classe em questão implemente a interface *ICRUDProvider*).

public interface ICRUDProvider<DE> : IUtilProvider where DE : class, new()

{

List<DE> Insert(List<DE> newEntities);

void Update(List<DE> updatedEntities, List<DE> originalEntities);

void Delete(List<DE> entities);

DE GetById(DE idEntity);

List<DE> Get(QueryParams<DE> queryParams, out int queryCount);

List<DE> GetAll();

}

Listagem 2: Interface ICRUDProvider

Para ilustrar o uso do ORM, como parte da prova de conceito do Sefaz.Fw, foi desenvolvido um *provider* que utiliza o *Entity Framework 5.0* como mecanismo de persistência.

Sua arquitetura física está descrita na figura abaixo:

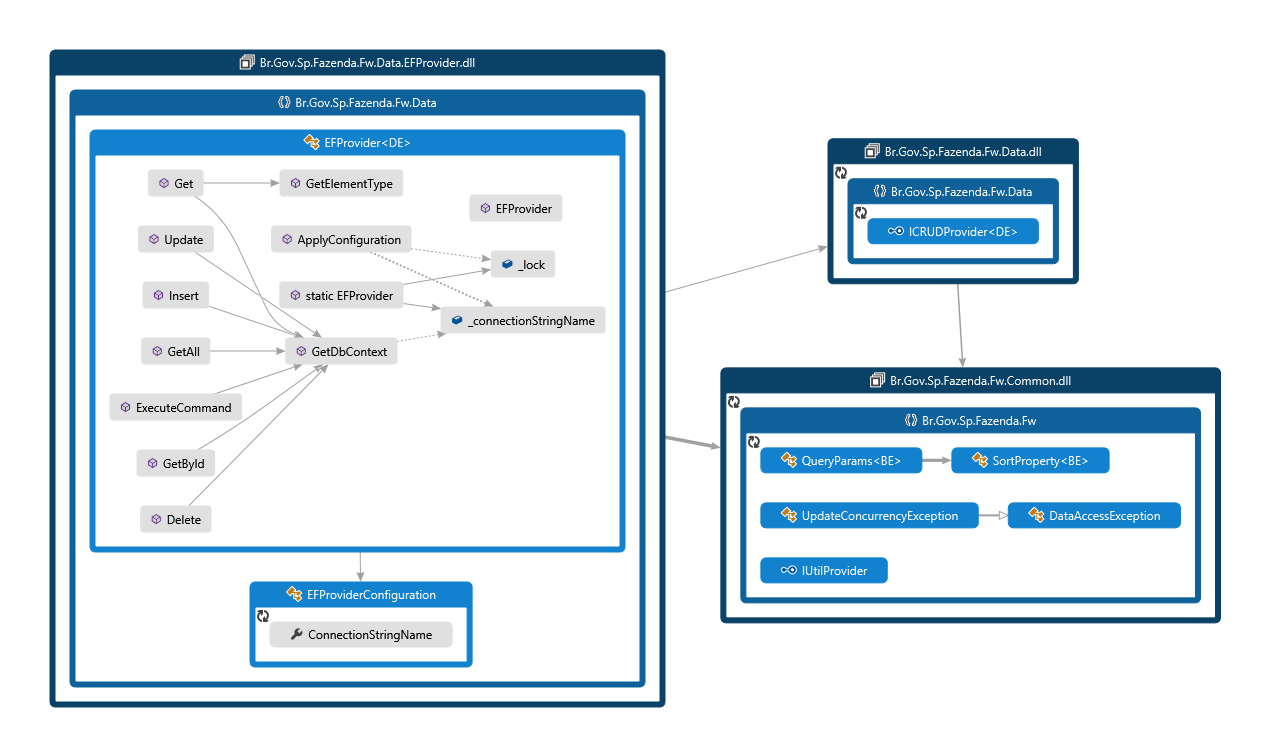


Figura 5: Arquitetura do provider EFProvider do Sefaz.Fw

Mais detalhes sobre esse *provider* ou o Sefaz.Fw podem ser encontrados no guia: “Arquitetura e Prova de Conceito do Framework para Aplicações .Net.docx”.

1. Recomendações
   1. Quando Usar ORM?

A primeira decisão a ser tomada com relação a um ORM é se ele se encaixa bem aos requisitos da aplicação a ser construída.

A decisão de usar ou não o Entity Framework pode ser baseada em alguns requisitos não-funcionais da aplicação em questão. A seguir são sugeridos alguns quesitos de avaliação a serem considerados:

|  |  |  |
| --- | --- | --- |
| Quesito de Avaliação | ORM Recomendado | Não recomendado |
| A aplicação não pode executar comandos SQL no banco de dados, apenas *stored procedures*. |  | **X**  [impeditivo] |
| Desenvolvedor não possui conhecimento em LINQ. |  | **X** |
| Existe a necessidade de definir todo e qualquer comando SQL previamente em tempo de desenho. |  | **X**  [impeditivo] |
| Aplicação deve evitar a criação de objetos em memória para representar seus dados. |  | **X** |
| Aplicação processa número grande de dados em lote, com pouca lógica de negócio. |  | **X** |
| Comandos SQL automaticamente gerados, com precauções contra *sql injection*. | **X** |  |
| Forma de programação única, independentemente do banco de dados usado. | **X** |  |
| Desenvolvedor não possui conhecimento em TSQL. | **X** |  |
| Ferramenta integrada ao ambiente de desenvolvimento para criar modelos de entidades ou importá-los a partir de uma base de dados existente. | **X** |  |
| Ferramenta integrada ao ambiente de desenvolvimento para atualizar um modelo de entidades a partir de uma base de dados existente. | **X** |  |
| Desenvolvimento que usa um modelo de objetos centrado na aplicação, incluindo tipos com herança, membros complexos, e relacionamentos. | **X** |  |
| Mudanças no mapeamento entre o modelo de objetos e o esquema da base de dados sem alterar o código da aplicação. | **X** |  |
| Validação da sintaxe dos comando SQL automática em *design time.* | **X** |  |
| Comandos SQL automaticamente gerados para operações CRUD. | **X** |  |
| Comandos SQL automaticamente gerados, considerando índices e um plano de execução otimizado. | **X** |  |

Tabela 3: Tabela de decisão sobre o uso do ORM

* 1. O Entity Data Model

No ciclo de vida da maioria das aplicações desenvolvidas na SEFAZ a base de dados é desenhada antes da construção da aplicação. Isto possibilita que administradores de dados examinem o modelo da base previamente. Por este motivo recomenda-se seguir a abordagem de desenvolvimento *Database First* para o EF.

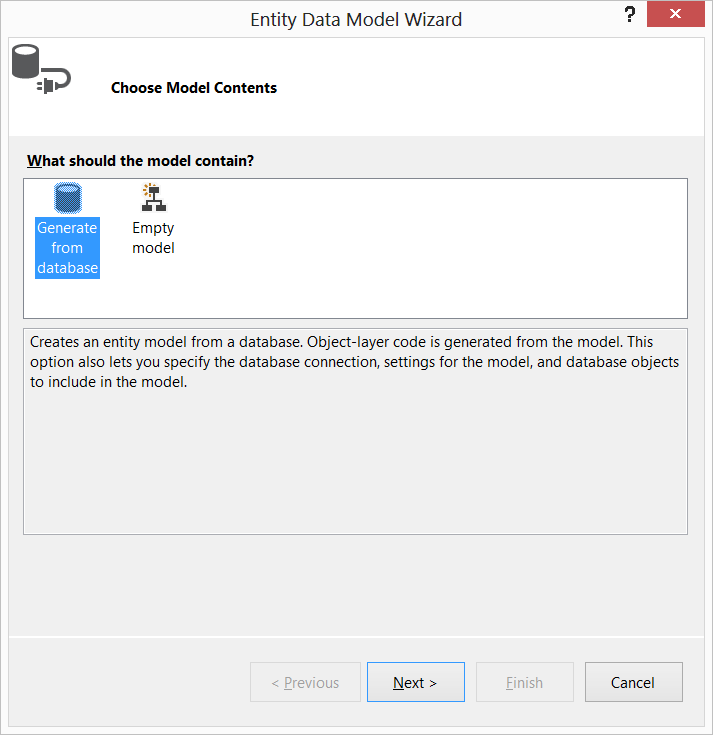
O Entity Framework trabalha com base em um *Entity Data Model*, um modelo de dados entidade-relacionamento cujo conceito central são as entidades e os relacionamentos. Quando a base de dados já está definida, o *Database First* permite realizar a engenharia reversa de um modelo a partir desta base existente. O modelo é armazenado em um arquivo EDMX (extensão .edmx), pode ser visualizado e editado no *Entity Framework Designer*. As classes que interagem com sua aplicação são geradas automaticamente a partir do arquivo EDMX.

É possível criar mais de um modelo EDMX caso você queira fazer um agrupamento lógico de suas entidades de banco de dados. Isso facilita a organização e manutenção.

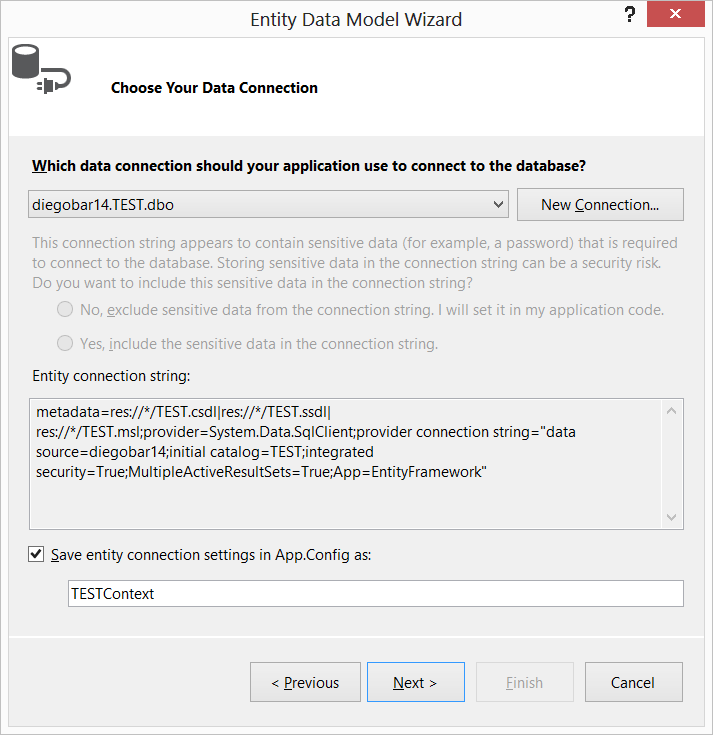
* + 1. Criando o Modelo via Database First

Veja o passo-a-passo para gerar um modelo a partir de uma base de dados existente:

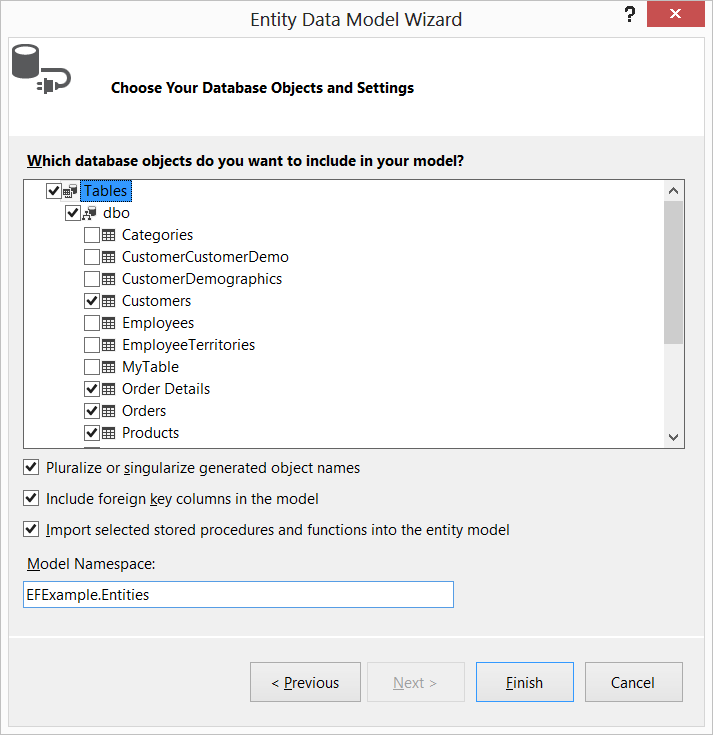
* Conexão com o banco de dados
  + No Visual Studio, abra a solução de seu projeto.
  + Selecione **View** -> **Server Explorer**
  + Certifique-se que em **Data Connections** existe uma conexão para a sua base de dados de trabalho. Caso não exista, adicione essa conexão.
* Criando o modelo
  + No seu projeto da camada *Data Access*, selecione **Project** -> **Add New Item…**
  + Selecione **Data** no menu à esquerda e então **ADO.NET Entity Data Model**
  + Digite o nome do modelo e clique **OK**
  + O **Entity Data Model Wizard** aparecerá
  + Selecione **Generate from Database** e clique **Next**



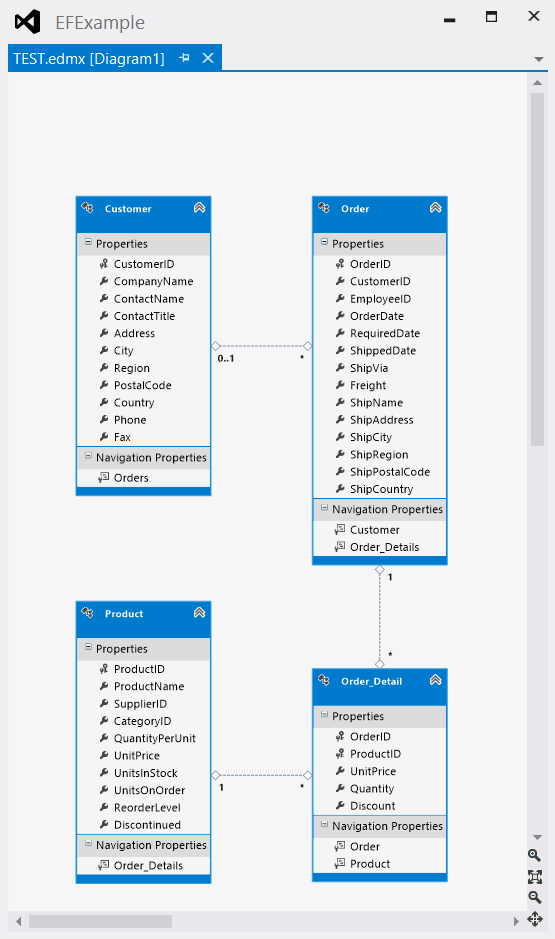
* + Selecione a conexão com a base de dados
  + Digite o nome da conexão e clique **Next**



* + Marque as tabelas, *views* e *stored procedures* que se deseja importar ao modelo.
  + Se necessário, altere o namespace do modelo



* + Clique **Finish**
  + O EF Designer apresentará o modelo recém importado



As classes geradas pelo modelo ficam logo abaixo dos templates T4 (extensão .tt).

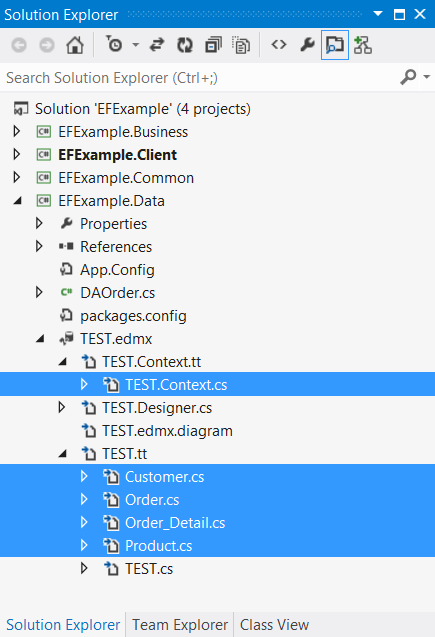


Figura 6: Classes geradas pelos templates T4

* + 1. Gerando as Classes das Entidades e do Contexto

Ao criar um EDMX no projeto, dois templates T4 (extensão .tt) são automaticamente adicionados. Eles são responsáveis pela geração da classe do contexto e da classe da entidade respectiva a cada tabela mapeada. Estas classes das entidades já são classes POCO.

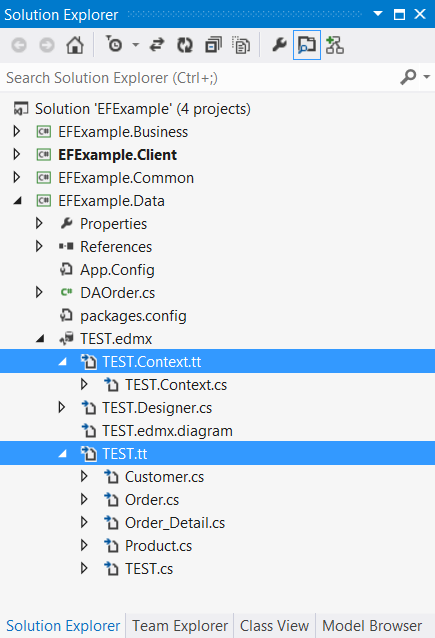


Figura 7: Templates T4 inseridos pelo modelo EDMX

Caso seja necessário alterar o namespace das classes geradas, basta alterar a propriedade **Custom Tool Namespace** do template T4 em questão.

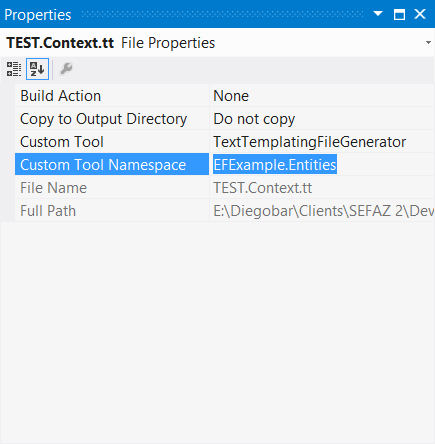


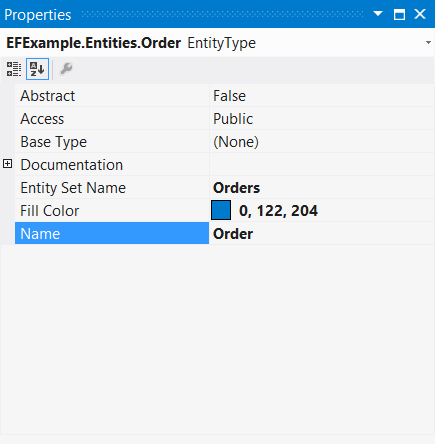
Figura 8: Propriedades do template T4

Os templates T4 são excelentes pontos de extensibilidade do EF, eles nos permitem customizar completamente as classes geradas a partir do modelo EDMX. Veja mais em <http://msdn.microsoft.com/en-us/data/gg558520.aspx> .

* + 1. Alterando o Mapeamento das Entidades

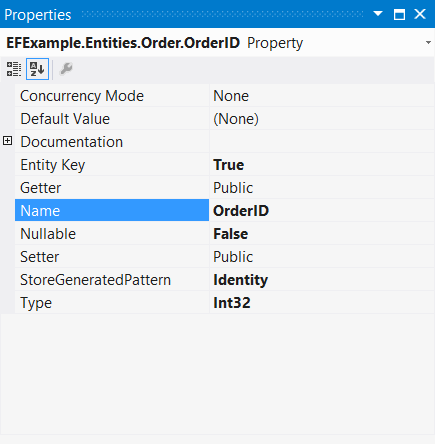
Assim como nas demais abordagens, existe a possibilidade de ajustar a entidade para melhor se adequar às necessidades do projeto, isto pode ser feito através da janela de propriedades do *EF Designer*:

* Ao selecionar uma entidade no modelo, a janela de propriedades será automaticamente preenchida com as informações da entidade.



Aqui pode-se alterar o nome da entidade, entre outras características da classe.

* Ao se selecionar uma propriedade da entidade no diagrama, a janela de propriedades será automaticamente preenchida com as informações da propriedade.

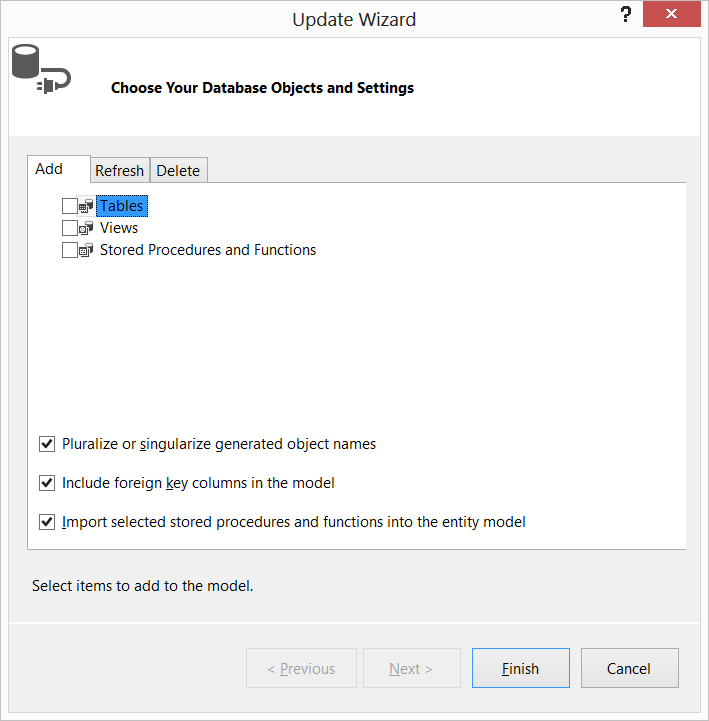


Aqui pode-se alterar o nome da propriedade, indicar se faz parte da chave ou não, entre outras características da propriedade.

* + 1. Refletindo Alterações da Base de Dados no Modelo

Após a primeira importação do esquema da base de dados, ainda é possível atualizar o modelo a partir da base de dados modificada:

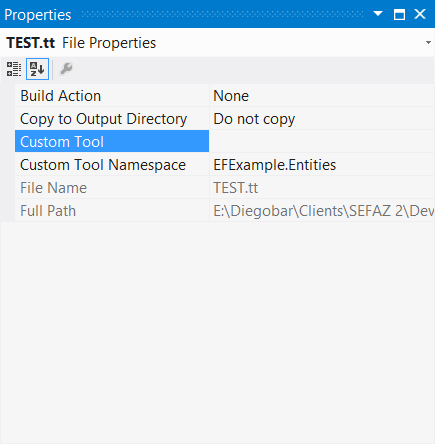
* Clique com o botão direito numa área vazia do modelo e selecione **Update Model from Database…**



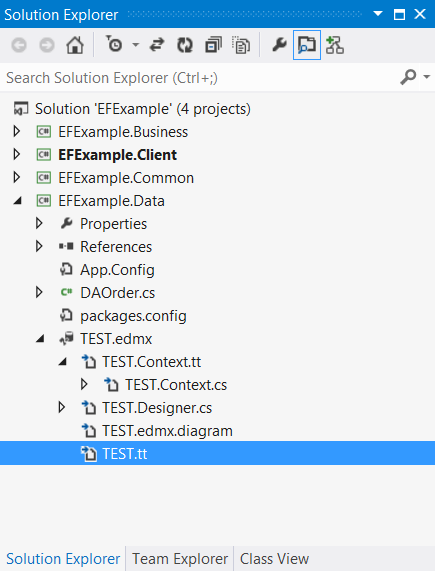
* No **Update Wizard**, na aba **Add** é possível adicionar tabelas, *views* e *stored procedures*, e nas abas **Refresh** e **Delete** é possível visualizar as alterações realizadas nas tabelas já mapeadas.
* Clique **Finish**
  + 1. Gerando as Classes das Entidades em Outro Projeto

No caso de aplicações distribuída em diversos projetos, pode ser necessário ter a classe do contexto num projeto (por exemplo: projeto da camada *Data Access*) e as classes das entidades em outro (por exemplo: projeto da camada *Common*). Para isto pode-se seguir os seguintes passos:

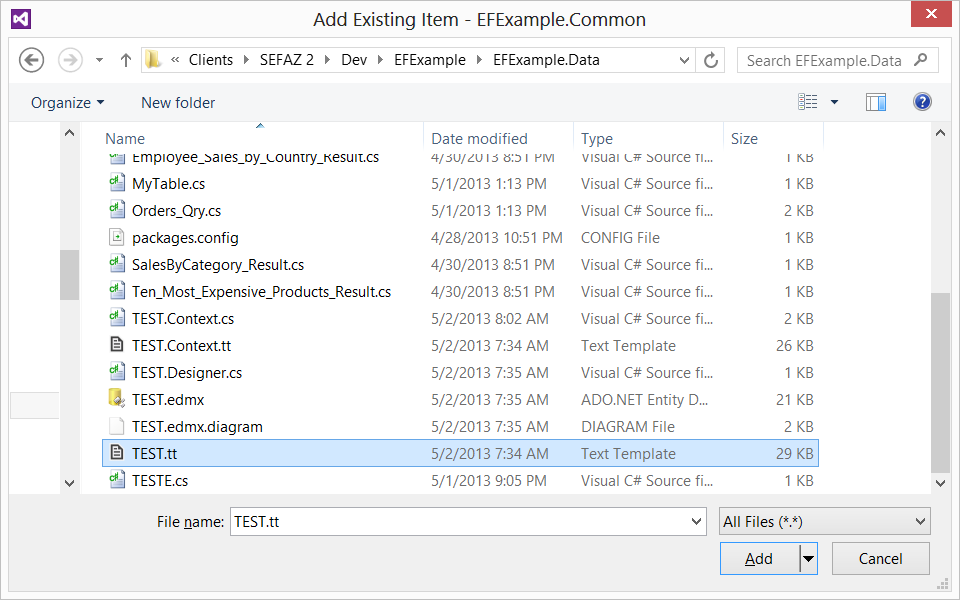
* No projeto **original**.
* Nas propriedades do template T4 da classe das entidades, limpe o campo **Custom Tool**.



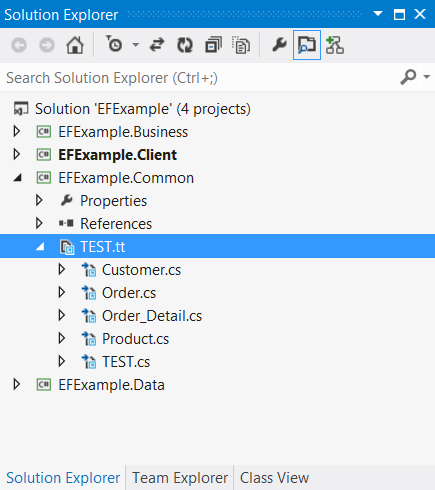
* Deletar as classes das entidades que foram geradas anteriormente.



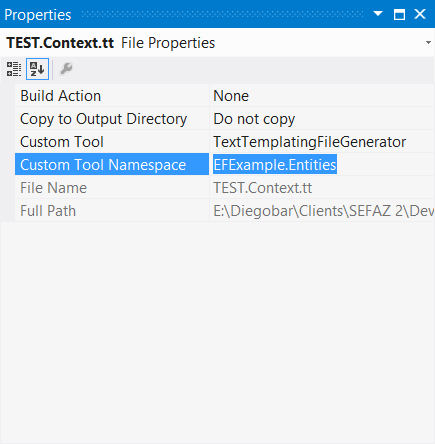
* No projeto **destino.**
* Selecionar **Project** -> **Add Existing Item...**



* Selecionar o template T4 das classes que se encontra no diretório do projeto original e adicionar através da opção **Add As Link** do botão **Add**.



* Caso seja necessário alterar o namespace das classes geradas, basta alterar a propriedade **Custom Tool Namespace** do template T4:



* 1. Uso do Entity Framework em Aplicações Distribuídas

O Entity Framework realiza diversas funções automaticamente através de seu objeto de contexto, o DbContext. Algumas destas operações precisam que os objetos entidades estejam “vinculados” a esse contexto.

Numa arquitetura fisicamente distribuída como a das aplicações corporativas da SEFAZ, onde entidades e DTO’s precisam trafegar entre diferentes máquinas, não é possível manter estes objetos vinculados ao contexto do EF. Por este motivo algumas considerações devem ser observadas:

* Ao serializar os objetos, desabilitar o *lazy-loading*. O *lazy-loading* executa uma consulta à base de dados para cada propriedade de navegação de relacionamento que é acessada. Isto pode causar inesperados acesso à base de dados ao se serializar um objeto.

Para desabilitar o *lazy-loading* basta atribuir o valor **False** para a propriedade **Lazy Loading Enabled** do modelo EDMX.

* Serviços que não mantêm estado são recomendados, consequentemente os serviços devem ser desenhados de forma que um objeto de contexto seja mantido somente durante a duração da solicitação ou da resposta.
* Caso o rastreamento de mudanças das entidades seja necessário, o mecanismo de troca de mensagem deve incluir informação suficiente para que mudanças possam ser aplicadas sem a necessidade de gravar valores em seção ou de refazer a consulta à base de dados. Por exemplo, um serviço que permite a atualização de uma entidade deve solicitar o objeto modificado junto com o objeto original.
* Atualizações concorrentes devem ser tratadas manualmente, pois deve-se usar o mecanismo de concorrência otimista com objetos que passam por serviços. Veja mais na seção 6.7
* Sempre que possível usar classes POCO para suas entidades. Isto aumenta as chances de interoperabilidade dos serviços e diminui os problemas na serialização.
* Ao usar classes POCO, não usar os *proxies*, pois são difíceis de serializar e precisam estar vinculados ao contexto do EF para executar suas funções. Estes *proxies* são objetos criados dinamicamente pelo EF para realizar funções “transparentemente” em cima das entidades, tais como: rastrear alterações e executar *lazy-loading*.
  1. Entidades e DTO’s
     1. Entidades POCO

Quando trabalhamos com o *EF Designer*, o template T4 das entidades já gera uma classe POCO para cada tabela mapeada. Esta classe já possui as *navigation properties* (propriedades que permitem o acesso a outras entidades relacionadas; pode ser uma única referência ou uma coleção de referências a outras entidades) declaradas. Por exemplo:

public partial class Order

{

public Order()

{

this.Order\_Details = new HashSet<Order\_Detail>();

}

public int OrderID { get; set; }

public string CustomerID { get; set; }

public Nullable<int> EmployeeID { get; set; }

public System.DateTime OrderDate { get; set; }

public Nullable<System.DateTime> RequiredDate { get; set; }

public Nullable<System.DateTime> ShippedDate { get; set; }

public Nullable<int> ShipVia { get; set; }

public Nullable<decimal> Freight { get; set; }

public string ShipNameX { get; set; }

public string ShipAddress { get; set; }

public string ShipCity { get; set; }

public string ShipRegion { get; set; }

public string ShipPostalCode { get; set; }

public string ShipCountry { get; set; }

public virtual Customer Customer { get; set; }

public virtual ICollection<Order\_Detail> Order\_Details { get; set; }

}

Listagem 3: Exemplo de entidade

Ao se obter uma entidade, ela deve estar sempre completa, ou seja, com suas propriedades de tipos primários preenchidos com os dados da base, desta forma ela sempre poderá atender as necessidades de persistência.

* + 1. Carregando Navigation Properties

Existem casos nos quais nem todas as *navigation properties* precisam ser carregadas (por padrão elas não são). Isto não evita que a entidade esteja íntegra.

Como citado anteriormente neste documento, para aplicações multicamadas recomenda-se o uso de classes POCO sem seus respectivos *proxies*, pois estes não são serializáveis. Consequentemente, não há como executar *lazy-loading*. Com isso, é importante informar ao EF quais *navigation properties* devem ser lidas em conjunto com a entidade em questão. Este processo é chamado de *eager-loading*.

*Eager-loading* pode ser feito através de queries LINQ ou através do método Include().

using (TESTContext ctx = new TESTContext())

{

List<Order> orders = ctx.Orders

.Include(o => o.Customer)

.ToList();

}

Listagem 4: Exemplo de leitura usando o método Include()

Quando nem todas as *navigation properties* de uma entidade precisam ser carregadas em todos os casos, pode-se criar mais de um método de leitura da entidade, cada um com uma combinação própria das *navigation properties* a serem carregadas. O exemplo a seguir carrega somente a *navigation property* Customer:

public List<Order> GetOrdersWithCustomer()

{

List<Order> orders = null;

using (TESTContext ctx = new TESTContext())

{

orders = ctx.Orders

.Include(o => o.Customer)

.ToList();

}

return orders;

}

Listagem 5: Exemplo de leitura que carrega somente a *navigation property* Customer

A partir do comando acima, o EF gera o seguinte comando SQL:

SELECT

[Extent1].[OrderID] AS [OrderID],

[Extent1].[CustomerID] AS [CustomerID],

[Extent1].[EmployeeID] AS [EmployeeID],

[Extent1].[OrderDate] AS [OrderDate],

[Extent1].[RequiredDate] AS [RequiredDate],

[Extent1].[ShippedDate] AS [ShippedDate],

[Extent1].[ShipVia] AS [ShipVia],

[Extent1].[Freight] AS [Freight],

[Extent1].[ShipName] AS [ShipName],

[Extent1].[ShipAddress] AS [ShipAddress],

[Extent1].[ShipCity] AS [ShipCity],

[Extent1].[ShipRegion] AS [ShipRegion],

[Extent1].[ShipPostalCode] AS [ShipPostalCode],

[Extent1].[ShipCountry] AS [ShipCountry],

[Extent2].[CustomerID] AS [CustomerID1],

[Extent2].[CompanyName] AS [CompanyName],

[Extent2].[ContactName] AS [ContactName],

[Extent2].[ContactTitle] AS [ContactTitle],

[Extent2].[Address] AS [Address],

[Extent2].[City] AS [City],

[Extent2].[Region] AS [Region],

[Extent2].[PostalCode] AS [PostalCode],

[Extent2].[Country] AS [Country],

[Extent2].[Phone] AS [Phone],

[Extent2].[Fax] AS [Fax]

FROM [dbo].[Orders] AS [Extent1]

LEFT OUTER JOIN [dbo].[Customers] AS [Extent2] ON [Extent1].[CustomerID] = [Extent2].[CustomerID]

Listagem 6: Exemplo de SQL gerado pelo EF usando o método Include()

O comando SQL acima pode ser obtido dentro do Visual Studio através do **IntelliTrace** filtrado pela categoria ADO.NET, ou através da inspeção das variáveis locais do tipo **DbQuery** (DbSet herda de DbQuery).

O exemplo a seguir carrega as *navigation properties* Customer e Order\_Details:

public List<Order> GetOrdersFull()

{

List<Order> orders = null;

using (TESTContext ctx = new TESTContext())

{

orders = ctx.Orders

.Include(o => o.Customer)

.Include(o => o.Order\_Details)

.ToList();

}

return orders;

}

Listagem 7: Exemplo de leitura que carrega as *navigation properties* Customer e Order\_Details

A partir do comando acima, o EF gera o seguinte comando SQL. Note que o comando faz apenas um *round-trip* ao banco de dados:

SELECT

[Project1].[OrderID] AS [OrderID],

[Project1].[CustomerID] AS [CustomerID],

[Project1].[EmployeeID] AS [EmployeeID],

[Project1].[OrderDate] AS [OrderDate],

[Project1].[RequiredDate] AS [RequiredDate],

[Project1].[ShippedDate] AS [ShippedDate],

[Project1].[ShipVia] AS [ShipVia],

[Project1].[Freight] AS [Freight],

[Project1].[ShipName] AS [ShipName],

[Project1].[ShipAddress] AS [ShipAddress],

[Project1].[ShipCity] AS [ShipCity],

[Project1].[ShipRegion] AS [ShipRegion],

[Project1].[ShipPostalCode] AS [ShipPostalCode],

[Project1].[ShipCountry] AS [ShipCountry],

[Project1].[CustomerID1] AS [CustomerID1],

[Project1].[CompanyName] AS [CompanyName],

[Project1].[ContactName] AS [ContactName],

[Project1].[ContactTitle] AS [ContactTitle],

[Project1].[Address] AS [Address],

[Project1].[City] AS [City],

[Project1].[Region] AS [Region],

[Project1].[PostalCode] AS [PostalCode],

[Project1].[Country] AS [Country],

[Project1].[Phone] AS [Phone],

[Project1].[Fax] AS [Fax],

[Project1].[C1] AS [C1],

[Project1].[OrderID1] AS [OrderID1],

[Project1].[ProductID] AS [ProductID],

[Project1].[UnitPrice] AS [UnitPrice],

[Project1].[Quantity] AS [Quantity],

[Project1].[Discount] AS [Discount]

FROM ( SELECT

[Extent1].[OrderID] AS [OrderID],

[Extent1].[CustomerID] AS [CustomerID],

[Extent1].[EmployeeID] AS [EmployeeID],

[Extent1].[OrderDate] AS [OrderDate],

[Extent1].[RequiredDate] AS [RequiredDate],

[Extent1].[ShippedDate] AS [ShippedDate],

[Extent1].[ShipVia] AS [ShipVia],

[Extent1].[Freight] AS [Freight],

[Extent1].[ShipName] AS [ShipName],

[Extent1].[ShipAddress] AS [ShipAddress],

[Extent1].[ShipCity] AS [ShipCity],

[Extent1].[ShipRegion] AS [ShipRegion],

[Extent1].[ShipPostalCode] AS [ShipPostalCode],

[Extent1].[ShipCountry] AS [ShipCountry],

[Extent2].[CustomerID] AS [CustomerID1],

[Extent2].[CompanyName] AS [CompanyName],

[Extent2].[ContactName] AS [ContactName],

[Extent2].[ContactTitle] AS [ContactTitle],

[Extent2].[Address] AS [Address],

[Extent2].[City] AS [City],

[Extent2].[Region] AS [Region],

[Extent2].[PostalCode] AS [PostalCode],

[Extent2].[Country] AS [Country],

[Extent2].[Phone] AS [Phone],

[Extent2].[Fax] AS [Fax],

[Extent3].[OrderID] AS [OrderID1],

[Extent3].[ProductID] AS [ProductID],

[Extent3].[UnitPrice] AS [UnitPrice],

[Extent3].[Quantity] AS [Quantity],

[Extent3].[Discount] AS [Discount],

CASE WHEN ([Extent3].[OrderID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C1]

FROM [dbo].[Orders] AS [Extent1]

LEFT OUTER JOIN [dbo].[Customers] AS [Extent2] ON [Extent1].[CustomerID] = [Extent2].[CustomerID]

LEFT OUTER JOIN [dbo].[Order Details] AS [Extent3] ON [Extent1].[OrderID] = [Extent3].[OrderID]

) AS [Project1]

ORDER BY [Project1].[OrderID] ASC, [Project1].[CustomerID1] ASC, [Project1].[C1] ASC

Listagem 8: Exemplo de SQL gerado pelo EF carregando as *navigation properties* Customer e Order\_Details

* + 1. Data Transfer Objects

Existem ainda cenários onde é requerida uma abstração ou composição de uma ou mais entidades. Neste caso recomenda-se o uso de um DTO. Normalmente os DTO’s são usados apenas para leitura e não são usados para persistência e por isso, ao voltarem para a camada *Data Access*, tem que ser mapeados de volta para a(s) entidade(s) a qual ele pertence.

Por exemplo, o seguinte DTO combina informações de duas entidades (Order e Customer):

public class DTOOrder

{

public int OrderID { get; set; }

public string CustomerID { get; set; }

public Nullable<int> EmployeeID { get; set; }

public DateTime OrderDate { get; set; }

public string CustomerCompanyName { get; set; }

public string CustomerContactName { get; set; }

public string CustomerCountry { get; set; }

}

Listagem 9: Exemplo de DTO combinando informações de duas entidades

Para obter a DTO acima, pode-se executar o seguinte comando LINQ:

using (TESTContext ctx = new TESTContext())

{

var query = from o in ctx.Orders

select new DTOOrder

{

OrderID = o.OrderID,

EmployeeID = o.EmployeeID,

CustomerID = o.CustomerID,

OrderDate = o.OrderDate,

CustomerCompanyName = o.Customer.CompanyName,

CustomerContactName = o.Customer.ContactName,

CustomerCountry = o.Customer.Country

};

dtos = query.ToList();

}

Listagem 10: Exemplo de comando LINQ para ler DTO’s

A partir do comando acima, o EF gera o seguinte comando SQL:

SELECT

[Extent1].[OrderID] AS [OrderID],

[Extent1].[EmployeeID] AS [EmployeeID],

[Extent1].[CustomerID] AS [CustomerID],

[Extent1].[OrderDate] AS [OrderDate],

[Extent2].[CompanyName] AS [CompanyName],

[Extent2].[ContactName] AS [ContactName],

[Extent2].[Country] AS [Country]

FROM [dbo].[Orders] AS [Extent1]

LEFT OUTER JOIN [dbo].[Customers] AS [Extent2] ON [Extent1].[CustomerID] = [Extent2].[CustomerID

Listagem 11: Exemplo de SQL gerado pelo EF para carregar DTO’s

* 1. Stored Procedures

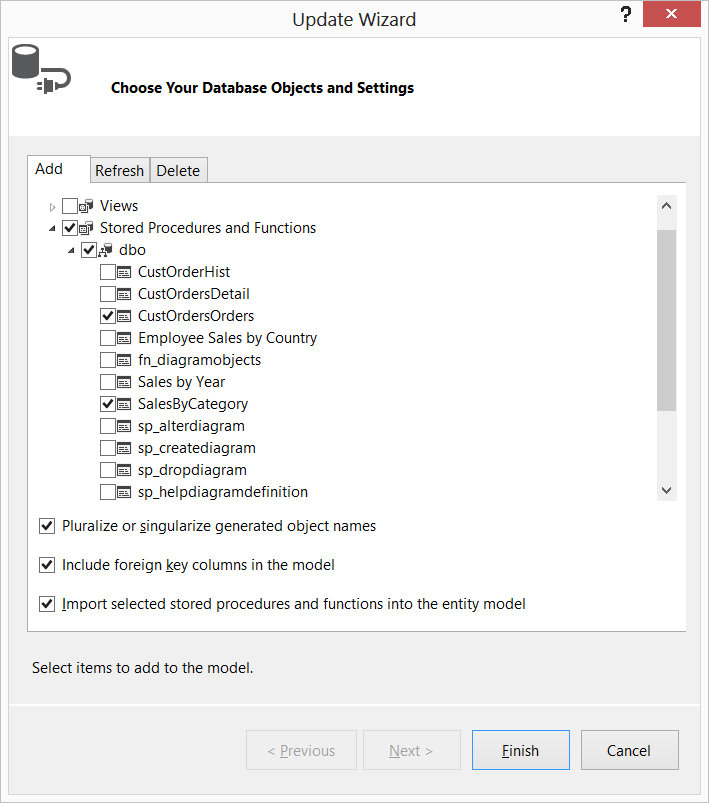
Uma das principais vantagens de se usar um ORM como o Entity Framework é poder abstrair as operações que serão realizadas na base de dados e deixar que ele as resolva. O desenvolvedor não precisa mais se preocupar com as características únicas dos banco de dados de diferentes fornecedores, assim como não precisa ter conhecimento avançado na linguagem SQL.

Os comandos SQL gerados pelo Entity Framework usam técnicas para evitar falhas de segurança e são otimizados para sempre oferecer o melhor desempenho, dada as meta-informações que possui.

* + 1. Mapeando uma Stored Procedure

No entanto, podem existir consultas e/ou operações de atualização complexas na base de dados que são melhor processadas pelo próprio banco de dados através de uma *stored procedure*. Para estas exceções, o *EF Designer* nos permite importar *stored procedures* no modelo:

* Clique com o botão direito numa área vazia do modelo e selecione **Update Model from Database…**



* No **Update Wizard**, na aba **Add** é possível selecionar as *stored procedures* que se deseja adicionar.
* Clique **Finish**

Por padrão, o resultado de cada *stored procedure* ou função importada que retorna mais de uma coluna gerará automaticamente um novo tipo complexo. Por exemplo:

ALTER PROCEDURE SalesByCategory

@CategoryName nvarchar(15), @OrdYear nvarchar(4) = '1998'

AS

IF @OrdYear != '1996' AND @OrdYear != '1997' AND @OrdYear != '1998'

BEGIN

SELECT @OrdYear = '1998'

END

SELECT ProductName,

TotalPurchase=ROUND(SUM(CONVERT(decimal(14,2), OD.Quantity \* (1-OD.Discount) \* OD.UnitPrice)), 0)

FROM [Order Details] OD, Orders O, Products P, Categories C

WHERE OD.OrderID = O.OrderID

AND OD.ProductID = P.ProductID

AND P.CategoryID = C.CategoryID

AND C.CategoryName = @CategoryName

AND SUBSTRING(CONVERT(nvarchar(22), O.OrderDate, 111), 1, 4) = @OrdYear

GROUP BY ProductName

ORDER BY ProductName

Listagem 12: Exemplo de *stored procedure*

A *stored procedure* acima gera o seguinte DTO para os resultados:

public partial class SalesByCategory\_Result

{

public string ProductName { get; set; }

public Nullable<decimal> TotalPurchase { get; set; }

}

Listagem 13: Exemplo de tipo gerado para os resultados de uma *stored procedure*

E pode ser chamado através do seguinte método disponível no objeto de contexto:

public virtual ObjectResult<SalesByCategory\_Result> SalesByCategory(string categoryName, string ordYear)

Listagem 14: Exemplo de método gerado para chamada de uma *stored procedure*

Para que o resultado seja do tipo de uma entidade, as colunas retornadas pela *stored procedure* devem combinar exatamente com as propriedades do tipo entidade retornado.

* 1. Rastreamento de Alterações

Como citado anteriormente neste documento, as entidades devem ser “desvinculadas” do contexto do EF, e também não podem ter *proxies* atrelados a elas. Isto porquê em algum momento elas podem ser serializadas e trafegadas para outra máquina.

Consequentemente, as entidades não podem ser rastreadas automaticamente pelo contexto do EF. Para que o EF identifique quais propriedades de uma entidade tiveram seus valores alterados é necessário passar dois objetos: um com os valores originais e outro com os valores modificados. Isto permite que o EF gere um comando de *update* para as colunas alteradas apenas. Por exemplo:

public void OrderUpdate(Order originalOrder, Order modifiedOrder)

{

using (TESTEContext ctx = new TESTEContext())

{

DbEntityEntry<Order> entry = ctx.Entry<Order>(originalOrder);

entry.State = System.Data.EntityState.Unchanged;

entry.CurrentValues.SetValues(modifiedOrder);

ctx.SaveChanges();

}

}

Listagem 15: Exemplo de atualização com os valores originais

Caso não se deseje manter uma cópia do objeto da entidade com os valores originais, é possível também passar apenas o objeto com os valores modificados. Desta forma o EF gera um comando de *update* para todas as colunas. Por exemplo:

public void OrderUpdate(Order modifiedOrder)

{

using (TESTEContext ctx = new TESTEContext())

{

DbEntityEntry<Order> entry = ctx.Entry<Order>(modifiedOrder);

entry.State = System.Data.EntityState.Modified;

ctx.SaveChanges();

}

}

Listagem 16: Exemplo de atualização sem os valores originais

* 1. Concorrência

Situações de concorrência acontecem quando, na maioria das vezes, quando:

* Processo “A” lê um registro da base de dados e mantêm uma cópia (por exemplo: um objeto em memória);
* Esta cópia tem seus valores modificados;
* Processo “B” atualiza a base de dados com novos valores para o registro.
* No momento que o processo “A” tenta gravar os valores modificados de sua cópia na base de dados, suas modificações não estão mais baseadas na última versão do registro, ocasionando sobrescrita (consequentemente: perda) de dados.

Este tipo de situação pode acontecer em aplicações multiusuários que permitem atualização simultânea no mesmo registro na base de dados.

Por padrão, o Entity Framework salva alterações de uma entidade na base de dados sem verificar situações de concorrência, ou seja, a atualização no registro realizada por último sobrescreve as outras atualizações.

Quando configurado para verificar situações de concorrência, o Entity Framework implementa um modelo otimista. Isto significa que nenhum *lock* em banco é feito entre a leitura dos dados e a sua atualização.

* + 1. Configurando as Propriedades para Concorrência

Para entidades que podem sofrer situações de concorrência, recomenda-se que sejam definidas uma ou mais propriedades para participar do controle da concorrência. Isto é feito através da propriedade **Concurrency Mode**, mudando seu valor para **Fixed**.

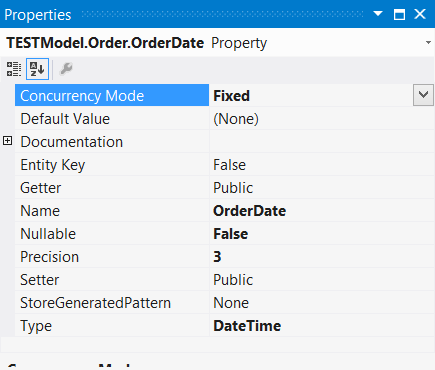


Figura 9: Concurrency Mode

Desta forma estas propriedades farão parte da cláusula *where* do comando de *update* ao se salvar as modificações. Se os valores dessas propriedades que foi lido previamente não for igual ao valores atuais da base de dados, nenhum atualização é feira.

Uma das formas mais eficazes de adicionar controle de concorrência para uma entidade é adicionando uma coluna na base de dados do tipo *timestamp* e marcar no modelo a sua respectiva propriedade como *Concurrency Mode = Fixed*. Uma coluna *timestamp* é atualizada pelo próprio banco de dados toda vez que o registro for atualizado.

* + 1. Tratando Situações de Concorrência

Uma vez as entidades configuradas para verificar a concorrência na sua atualização, fica a cargo do desenvolvedor tratá-la no momento que a concorrência ocorre. O Entity Framework dispara uma exceção do tipo DbUpdateConcurrencyException quando detecta uma situação de concorrência ao atualizar uma entidade.

As seguintes abordagens são utilizadas para resolver conflitos de concorrência:

**Base de dados ganha**

Nesta abordagem a primeira atualização é a que fica. Qualquer atualização subsequente não é realizada. Pode-se conseguir isto usando o método DbEntityEntry.Reload() para sobrescrever os valores modificados do objeto entidade pelos valores correntes da base de dados. Por exemplo:

public void OrderUpdate1(Order modifiedOrder)

{

using (TESTContext ctx = new TESTContext())

{

ctx.Entry(modifiedOrder).State = System.Data.EntityState.Modified;

bool saveSucceeded;

do

{

saveSucceeded = true;

try

{

ctx.SaveChanges();

}

catch (DbUpdateConcurrencyException ex)

{

saveSucceeded = false;

DbEntityEntry entry = ex.Entries.Single();

// Sobrescreve os valores da entidade pelos da base de dados.

entry.Reload();

}

} while (!saveSucceeded);

}

}

Listagem 17: Exemplo de atualização com abordagem Base de dados ganha

**Cliente ganha**

Aqui se faz o oposto, a última atualização é a que permanece. Os valores correntes da base de dados são sobrescritos pelos valores que estão no objeto entidade. Fazemos isto lendo os valores correntes da base de dados e os atribuímos como os valores originais da entidade. Por exemplo:

public void OrderUpdate2(Order modifiedOrder)

{

using (TESTContext ctx = new TESTContext())

{

ctx.Entry(modifiedOrder).State = System.Data.EntityState.Modified;

bool saveSucceeded;

do

{

saveSucceeded = true;

try

{

ctx.SaveChanges();

}

catch (DbUpdateConcurrencyException ex)

{

saveSucceeded = false;

DbEntityEntry entry = ex.Entries.Single();

// Sobrescreve apenas os valores originais pelos da base de dados.

entry.OriginalValues.SetValues(entry.GetDatabaseValues());

}

} while (!saveSucceeded);

}

}

Listagem 18: Exemplo de atualização com abordagem Cliente ganha

**Resolução customizada**

Esta abordagem sugere combinar os valores correntes da base de dados com os valores da entidade, para isso o conflito deve ser resolvido através de alguma lógica ou pela vontade do usuário, antes de realizar a atualização. Por exemplo:

public void OrderUpdate3(Order modifiedOrder)

{

using (TESTContext ctx = new TESTContext())

{

ctx.Entry(modifiedOrder).State = System.Data.EntityState.Modified;

bool saveSucceeded;

do

{

saveSucceeded = true;

try

{

ctx.SaveChanges();

}

catch (DbUpdateConcurrencyException ex)

{

saveSucceeded = false;

DbEntityEntry entry = ex.Entries.Single();

// Recupera os valores corrente da entidade e da base de dados

Order entity = (Order)entry.Entity;

Order databaseValues = (Order)(entry.GetDatabaseValues().ToObject());

Order resolvedValues = null;

// Deixa para o usuário decidir quais valores devem ficar

HaveUserResolveConcurrency(entry, databaseValues, ref resolvedValues);

// Sobrescreve os valores originais com os da base de dados

// e os valores correntes com os resolvidos pelo usuário.

entry.OriginalValues.SetValues(databaseValues);

entry.CurrentValues.SetValues(resolvedValues);

}

} while (!saveSucceeded);

}

}

private void HaveUserResolveConcurrency(DbEntityEntry entry, Order databaseValues, ref Order resolvedValues)

{

// Exibe os valores correntes da entidade e da base de dados ao usuário

// e o deixa resolver os valores a serem gravados.

}

Listagem 19: Exemplo de atualização com abordagem Customizada

* 1. Desempenho

Frameworks de mapeamento objeto-relacional são uma maneira conveniente de fornecer uma abstração para o acesso a dados em um aplicativo orientado a objeto e, como em qualquer abstração, desempenho pode tornar-se uma preocupação.

A seguir estão algumas questões que podem influenciar o desempenho da aplicação que usa o Entity Framework.

* + 1. .NET Framework 4.0 versus 4.5

Muitas das melhorias de desempenho feitas no Entity Framework 5 residirem dentro os componentes principais que acompanham o .NET 4.5.

O .NET Framework 4.5 inclui investimentos significativos em desempenho, beneficiando todos os cenários de aplicativo. Além disso, porque o .NET 4.5 é uma atualização para o .NET 4, mesmo os seus aplicativos que usam o .NET 4 podem desfrutar das muitas melhorias de desempenho aplicadas para os recursos .NET 4 existentes.

* + 1. Auto-Compiled LINQ Queries

Para executar uma consulta *LINQ to Entities*, o Entity Framework caminha sobre a árvore de expressão gerada pelo compilador c# / Visual Basic e a traduz em SQL.

No entanto, compilar a árvore de expressão para SQL envolve algum processamento, especialmente para consultas mais complexas. Em versões anteriores do Entity Framework, para evitar de ter que pagar este preço no desempenho sempre que uma consulta LINQ fosse executada, você tinha que usar a classe *CompiledQuery*.

A versão 5 do Entity Framework oferece suporte a um novo recurso chamado consultas LINQ **Auto-Compiled**. Agora cada consulta LINQ to Entities que você executa, ela é automaticamente compilada e colocada no cache de plano de consulta do Entity Framework. Cada execução subsequente da consulta, o Entity Framework irá encontrá-la em seu cache de consulta e não terá que passar pelo processo de compilação inteiro novamente.

* + 1. Pre-generated Views

*Mapping Views* são representações executáveis das transformações especificadas no mapeamento para cada entidade e associação. Existem dois tipos de *mapping views*:

* Views de consulta: representam a transformação necessária entre o esquema da base de dados para o esquema conceitual.
* Views de atualização: representam a transformação necessária entre o esquema conceitual para o esquema da base de dados.

A geração das *views* pode ser dinâmica quando o modelo é carregado, ou em tempo de desenho, ao usar "*pre-generated views*", as quais são serializadas na forma de comandos *Entity SQL* num arquivo C# ou VB.

* + - 1. Gerando Views através de Templates T4

A forma mais simples para gerar as *views* quando seu modelo está num arquivo EDMX é usando templates T4.

O blog do time do ADO.NET possui um artigo que descreve como usar um template T4 para a geração de *views* ( <http://blogs.msdn.com/b/adonet/archive/2008/06/20/how-to-use-a-t4-template-for-view-generation.aspx> ). Este artigo inclui um template T4 que pode ser baixado e adicionado ao seu projeto. O template foi escrito para a primeira versão do EF. Para usá-lo na versão 5 é necessário modificar os *namespaces* do XML no método GetConceptualMappingAndStorageReaders para usar os *namespaces do* Entity Framework 5:

* XNamespace edmxns = "http://schemas.microsoft.com/ado/2009/11/edmx";
* XNamespace csdlns = "http://schemas.microsoft.com/ado/2009/11/edm";
* XNamespace mslns = "http://schemas.microsoft.com/ado/2009/11/mapping/cs";
* XNamespace ssdlns = "http://schemas.microsoft.com/ado/2009/11/edm/ssdl";
  + 1. Consultas Não-rastreáveis

O rastreamento das entidades lidas pelo do objeto de contexto pode ser desabilitado. Isto pode melhorar o desempenho das consultas realizadas pelo EF quando esta funcionalidade não é necessária. Para isto pode-se usar o método DbQuery.AsNoTracking(). Por exemplo:

using (TESTContext ctx = new TESTContext())

{

List<Order> orders = ctx.Orders

.Where(o => o.CustomerID == "QUEDE")

.AsNoTracking()

.ToList();

}

Listagem 20: Exemplo do uso do método AsNoTracking()

1. Referências

|  |  |
| --- | --- |
| Assunto | Url |
| Ten advantages of an ORM | <http://blogs.msdn.com/b/gblock/archive/2006/10/26/ten-advantages-of-an-orm.aspx> |
| Entity Framework 5.0 | <http://msdn.microsoft.com/en-us/data/ee712907> |
| Why use the Entity Framework? | <http://blogs.msdn.com/b/dsimmons/archive/2008/05/17/why-use-the-entity-framework.aspx> |
| Entity Framework Development Workflows | <http://msdn.microsoft.com/en-us/data/jj590134> |
| Performance Considerations for Entity Framework 5 | <http://msdn.microsoft.com/en-us/data/hh949853> |
| Entity Framework Providers | <http://msdn.microsoft.com/en-us/data/dd363565.aspx> |
| Database First | <http://msdn.microsoft.com/en-us/data/jj206878> |
| Anti-Patterns To Avoid In N-Tier Applications | <http://msdn.microsoft.com/en-us/magazine/dd882522.aspx> |
| Working with N-Tier applications | <http://msdn.microsoft.com/en-us/data/jj613668> |
| T4 Templates and the Entity Framework | <http://msdn.microsoft.com/en-us/data/gg558520.aspx> |
| An Overview of Performance Improvements in .NET 4.5 | <http://msdn.microsoft.com/en-us/magazine/hh882452.aspx> |
| How to use a T4 template for View Generation | http://blogs.msdn.com/b/adonet/archive/2008/06/20/how-to-use-a-t4-template-for-view-generation.aspx |

Tabela 4: Referências

1. Apêndice A – Modelos de Programação do EF

O EF 5.0 oferece três abordagens de desenvolvimento:

**Code First**

Utilizando *Code First* é possível gerar as classes a partir de uma base de dados já existente ou gerar uma nova base de dados a partir do código-fonte. Uma das diferenças é que esta modalidade dispensa o modelo na forma de xml (extensão .edmx), pois a estrutura estará definida diretamente nas classes do modelo, ou seja em código-fonte.

**Model First**

Esta modalidade está mais direcionada para aplicativos novos que ainda não possuem uma base de dados criada. Nesta abordagem, ao definir visualmente o modelo, serão definidas as classes e por fim o SQL que dará origem a base de dados.

O *Entity Data Model* é um modelo de dados entidade-relacionamento cujo conceito central são as entidades e os relacionamentos

**Database First**

Está mais direcionado para aplicativos que já possuem uma base de dados definida. O *Database First* permite a engenharia reversa de um modelo a partir de um banco de dados existente. O modelo é armazenado em um arquivo EDMX (extensão .edmx) e pode ser visualizado e editado no *Entity Framework Designer*. As classes que interagem com sua aplicação são geradas automaticamente a partir do arquivo EDMX.

Para maiores detalhes, veja em <http://msdn.microsoft.com/en-us/data/jj590134> .

1. Apêndice B – Code First

No guia foi apresentada a abordagem Database First, a qual cria um modelo de domínio a partir de uma base de dados existente e usa o EF Designer para “desenhar” as entidades de negócio e seus relacionamentos.

Neste apêndice mostraremos uma outra abordagem, chamada Code First, que define as classes de entidades de negócio e todo o mapeamento no código. A ideia principal desta abordagem é ter o desenvolvimento centrado no modelo de domínio (ou entidades de negócio e seus relacionamentos), e o repositório de dados ser totalmente transparente.

* 1. Criando o Modelo

O primeiro passo é criar as entidades de negócio. No nosso exemplo criaremos duas entidades simples relacionadas:

public class Order

{

public Order()

{

this.Order\_Details = new HashSet<Order\_Detail>();

}

public int OrderID { get; set; }

public string CustomerID { get; set; }

public System.DateTime OrderDate { get; set; }

public virtual ICollection<Order\_Detail> Order\_Details { get; set; }

}

Listagem 21: Entidade de negócio Order

public class Order\_Detail

{

public int OrderID { get; set; }

public int ProductID { get; set; }

public decimal UnitPrice { get; set; }

public short Quantity { get; set; }

public float Discount { get; set; }

public virtual Order Order { get; set; }

}

Listagem 22: Entidade de Negócio Order\_Detail

O segundo passo é criar a classe que representa a sessão com a base de dados. Para isso criar-se uma classe derivada de DbContext e acrescenta-se uma propriedade DbSet<BE> para cada entidade:

public class MyContext : DbContext

{

public DbSet<Order> Orders { get; set; }

public DbSet<Order\_Detail> OrderDetails { get; set; }

public MyContext()

: base()

{ }

public MyContext(string nameOrConnectionString)

: base(nameOrConnectionString)

{ }

}

Listagem 23: Classe de contexto

* 1. Configurando o Mapeamento

É muito comum termos diferenças entre as entidades de negócio e as tabelas do repositório. Além disso, alguns dados adicionais devem ser informados para que o banco de dados possa garantir a integridade dos dados da aplicação.

Nas classes das entidades exemplo acima, precisamos indicar quais serão as propriedades que formarão as chaves primárias. Existem algumas formas de se conseguir isto, uma delas é usando **Data Annotations**. As entidades ficariam assim:

public class Order

{

public Order()

{

this.Order\_Details = new HashSet<Order\_Detail>();

}

[Key]

public int OrderID { get; set; }

public string CustomerID { get; set; }

public System.DateTime OrderDate { get; set; }

public virtual ICollection<Order\_Detail> Order\_Details { get; set; }

}

Listagem 24: Entidade Order com atributo *[Key]*

public class Order\_Detail

{

[Key]

public int OrderID { get; set; }

[Key]

public int ProductID { get; set; }

public decimal UnitPrice { get; set; }

public short Quantity { get; set; }

public float Discount { get; set; }

public virtual Order Order { get; set; }

}

Listagem 25: Entidade Order\_Detail com atributo *[Key]*

Esta técnica decora as propriedades da entidade de negócio com atributos que adicionam informação para ser usada no mapeamento com a base de dados.

Outra técnica usada é através de **Fluent API**. Na classe do contexto que deriva de *DbContext*, sobrescreva o método *OnModelCreating* e use o parâmetro *modelBuilder* para configurar as entidades, as propriedades e os relacionamentos. Por exemplo:

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Order>()

.HasKey(o => o.OrderID);

modelBuilder.Entity<Order\_Detail>()

.HasKey(d => new { d.OrderID, d.ProductID });

}

Listagem 26: Exemplo de método *OnModelCreating*

O exemplo acima tem o mesmo efeito que os atributos *[Key]* mostrados anteriormente.

Ainda usando *Fluent API*, pode-se definir classes de configuração para cada entidade, por exemplo:

public class OrderMap : EntityTypeConfiguration<Order>

{

public OrderMap()

{

this.HasKey(o => o.OrderID);

this.Property(o => o.CustomerID)

.HasMaxLength(5);

}

}

Listagem 27: Exemplo de classe de configuração para a entidade Order

public class OrderDetailMap : EntityTypeConfiguration<Order\_Detail>

{

public OrderDetailMap()

{

this.HasKey(o => new { o.OrderID, o.ProductID });

}

}

Listagem 28: Exemplo de classe de configuração para a entidade Order\_Detail

Deve-se derivar da classe *EntityTypeConfiguration<BE>* e implementar seu construtor com as configurações necessárias. Depois basta adicionar uma instância desta classe na coleção *modelBuilder.Configurations* no método *OnModelCreating*.

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Configurations.Add(new OrderMap());

modelBuilder.Configurations.Add(new OrderDetailMap());

}

Listagem 29: Método *OnMedelCreating*

* 1. Alterando o Modelo

Quando fazemos mudanças nas entidades de negócio (modelo de domínio), também precisamos alterar a base de dados. Para isto existe uma funcionalidade chamada **Code First Migrations**.

*Migrations* permite que tenhamos no nosso projeto um conjunto ordenado de classes, cada uma com uma série de atualizações no esquema da base de dados. Desta forma é possível manter o histórico das mudanças no esquema da base de dados.

1) O primeiro passo é habilitar o *Code First* *Migrations*:

* No **Package Manager Console**, execute o comando **Enable-Migrations**.
  + Uma pasta chamada **Migrations** é criada no seu projeto. Dentro dessa pasta, duas classes são inicialmente criadas:
    - Configuration.cs – Oferece a possibilidade de gerar uma carga inicial nas tabelas, etc.
    - <timestamp>\_InitialCreate.cs – É a primeira migração do projeto. Representa o esquema da base de dados naquele momento. Esta classe deriva de *DbMigration* e implementa basicamente dois métodos: *Up()* e *Down()*, responsáveis pelo upgrade e downgrade, respectivamente.

2) Em seguida, fazemos as alterações necessárias no modelo e/ou nas suas configurações.

* No nosso exemplo adicionamos uma restrição de tamanho para a coluna Order.CustomerId:

this.Property(o => o.CustomerID).HasMaxLength(5);

3) Geramos uma nova classe *Migration* para registrar as mudanças necessárias a serem aplicadas à base de dados:

* No **Package Manager Console**, execute o comando **Add-Migration MaxLengthCustomerId**. O parâmetro “MaxLengthCustomerId” é o nome que demos à migração.
  + A classe <timestamp>\_MaxLengthCustomerId.cs é criada na pasta Migrations:

public partial class MaxLengthCustomerId : DbMigration

{

public override void Up()

{

AlterColumn("dbo.Orders", "CustomerID", c => c.String(maxLength: 5));

}

public override void Down()

{

AlterColumn("dbo.Orders", "CustomerID", c => c.String());

}

}

Listagem 30: Exemplo de classe *Migration* gerada

4) Por fim, atualizamos o esquema da base de dados:

* No **Package Manager Console**, execute o comando **Update-Database**.

Este comando executará todas as classes *Migration* ainda não executadas, na sua ordem cronológica.

1. Apêndice C – Data Providers para o Entity Framework

O modelo de *data provider* do ADO.NET fornece uma interface gerenciada comum no .NET para se conectar e interagir com um repositório de dados. O Entity Framework baseia-se em cima do modelo de *data provider* do ADO.NET para permitir a utilização do EF com qualquer fonte de dados para o qual exista um *provider* suportado disponível.

O .NET Framework inclui *providers* ADO.NET para acesso direto ao Microsoft SQL Server e para o acesso indireto a outros bancos de dados através de drivers ODBC e OLE DB. Para acesso direto a outros bancos de dados, muitos *providers* de terceiros estão disponíveis. Todos os provedores listados incluem suporte do Entity Framework.

|  |  |
| --- | --- |
| Empresa | Bancos de Dados Suportados |
|  | * MySQL |
|  | * Oracle |
|  | * Oracle * DB2 * Sybase * Microsoft SQL Server. |
|  | * Oracle * MySQL * SQL Lite * PostgreSQL * Microsoft SQL Server |
|  | * Virtuoso data (SQL, XML, and RDF) * Oracle (versions 7.x to 11.x) * Microsoft SQL Server (6.x to 2005) * IBM DB2 * Sybase (4.2 to 12.x+) * IBM Informix (5.x to 11.x) * Ingres (6.4 to 9.x) * Progress (7.x to 10.x) * MySQL * PostgreSQL * Firebird |
|  | * DB2 * Informix * U2 |
|  | * Sybase iAnywhere's |
|  | * Sybase ASE |
|  | * SQLite |
|  | * Synergy DBMS |
|  | * Firebird |
|  | * PostgreSQL |

Tabela 5: Data Providers para o Entity Framework

Veja mais detalhes em <http://msdn.microsoft.com/en-us/data/dd363565.aspx>