Microsoft

Marián Kubica

Základy technológie Entity Framework

OD študentov PRE študentov

Fakulta hospodárskej informatiky EU v Bratislave

Microsoft Visual Studio 2010

Autor: Ing. Marián Kubica
Odborný garant: Ing. Ján Hanák, PhD., MVP

Základy technológie Entity Framework
Praktické cvičenie zo série „Od študentov pre študentov“

Rok vydania: 2012

Charakteristika praktických cvičení zo série „Od študentov pre študentov“

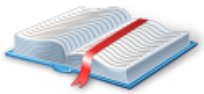
Sme presvedčení o tom, že keď inteligentní mladí ľudia ovládnu najmodernejšie počítačové technológie súčasnosti, ich tvorivý potenciál je vskutku nekonečný. Primárnym cieľom iniciatívy, ktorá stojí za sériou praktických cvičení „Od študentov pre študentov“, je maximalizácia hodnôt ľudských kapitálov študentov ako hlavných členov akademických komunít. Praktické cvičenia zo série „Od študentov pre študentov“ umožňujú študentom využiť ich existujúce teoretické znalosti, pričom efektívnym spôsobom predvádzajú, ako možno tieto znalosti s výhodou uplatniť pri vývoji atraktívnych počítačových aplikácií v rôznych programovacích jazykoch (C, C++, C++/CLI, C#, Visual Basic, F#). Budeme nesmierne šťastní, keď sa našim praktickým cvičeniam podarí u študentov prebudiť a naplno rozvinúť záujem o programovanie a vývoj počítačového softvéru. Veríme, že čím viac sofistikovaných IT odborníkov vychováme, tým viac budeme môcť spoločnými silami urobiť všetko pre to, aby sa z tohto sveta stalo lepšie miesto pre život.

Základy technológie Entity Framework

Cieľové publikum: **študenti** so znalosťami databázového programovania a jazyka C#.
Vedomostná náročnosť: ■■■□□.
Časová náročnosť: **1** hodina a **20** minút.
Programovacie jazyky: **C#**.
Vývojové prostredia: **Microsoft Visual Studio 2010 Ultimate**.
Operačné systémy: **Windows 7**, Windows Vista, Windows XP.
Technológie: bázová knižnica tried (BCL), ADO.NET Entity Framework.

V tomto praktickom cvičení sa cieľové publikum oboznámi so základmi platformy ADO.NET Entity Framework. Čitatelia sa dozvedia, ako sa uskutočňuje návrh a implementácia EF-modelu v prostredí tejto platformy. Rovnako preštudujú problematiku jazykov Entity SQL a LINQ pri formulovaní dopytovacích výrazov. Stranou nezostane ani tematika kompilovaných dopytov či uložených procedúr. Praktické cvičenie je vhodné pre záujemcov o platformu Entity Framework, ktorí ovládajú základné databázové koncepty na platforme Microsoft .NET Framework.

Tip: Čitateľom, ktorí majú skúsenosti s jazykom C++, no dosiaľ sa ešte nestretli s jazykom C#, odporúčame, aby si najskôr preštudovali užitočné praktické cvičenie „**C++ => C#: Od jazyka C++ k jazyku C#**“, ktoré sa nachádza v galérii študentských praktických cvičení zo série „Od študentov pre študentov“.



Obsah praktického cvičenia

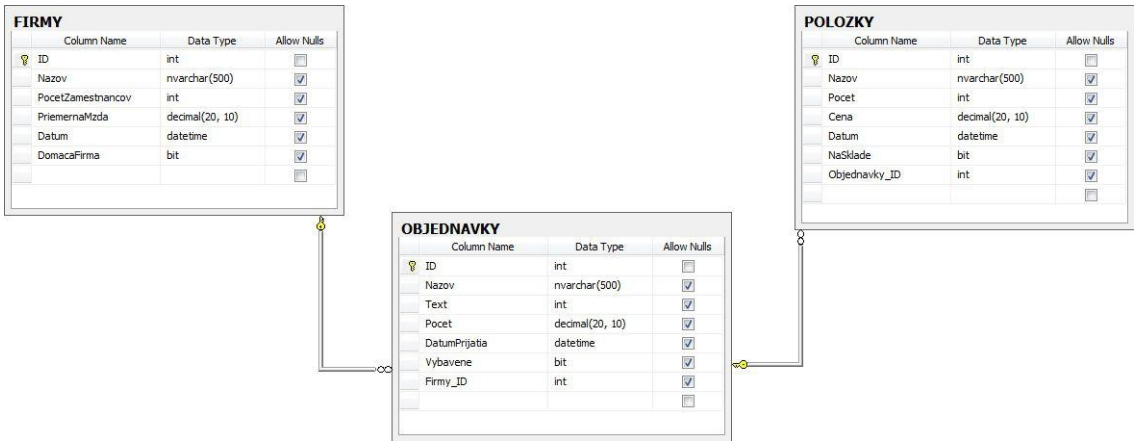
1 Predstavenie technológie Entity Framework 3
2 Návrh EF-modelu 10
3 Vytvorenie EF-modelu 10
4 Prispôsobenie EF-modelu 13
4.1 Parciálne triedy 13
4.2 Parciálne metódy 14
5 Dopytovací jazyk Entity SQL 14
6 Spôsoby dopytovania 14
7 Kompilované dopyty 16
8 Uložené procedúry v prostredí Entity Framework 16
9 EF-dopyt a fázy jeho tvorby 18
10 Výhody EF oproti iným technológiám ADO.NET (DataSet, SqlDataReader) 19
Použitá literatúra 20

1 Predstavenie technológie Entity Framework

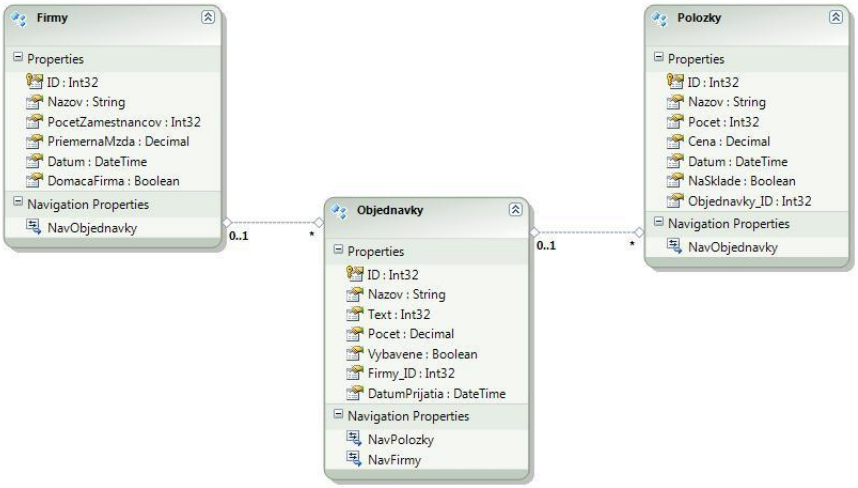
Entity Framework (EF) je technológia objektovo relačného mapovania (ORM) umožňujúca pracovať s databázovými objektmi ako s ktorýmikoľvek inými objektmi objektového programovania v prostredí príslušného programovacieho jazyka. Keď do vyvíjaného projektu softvérovej aplikácie pridáme šablónu **ADO.NET Entity Data Model**, určíme voľbu **Generate from database** a vykonáme ostatné nastavenia, vytvorí sa súbor s príponou „edmx", v ktorom máme k dispozícii konceptuálny model zodpovedajúci databázovému modelu na SQL serveri. Rovnako sú automaticky vygenerované asociácie medzi jednotlivými dátovými entitami, ktoré sú identické s definovanými databázovými reláciami.

Každá entita vytvorená v modeli slúži ako zdroj pre entitu triedy a všetky atribúty entity v modeli sa prejavia ako vlastnosti týchto tried. Špeciálnym atribútom sú tzv. navigačné vlastnosti, ktoré obsahujú referencie na asociované objekty.

Výhodou technológie Entity Framework je skutočnosť, že entity sú transformované na databázové objekty prostredníctvom mapovacej vrstvy. To v prípade zmeny databázového prostredia znamená len pretransformovať túto vrstvu a nie je nutné vykonávať zložité zmeny v aplikácii.



Obr. 1: Dátový model MSSQL Data model



Obr. 2: Konceptuálny model Entity Framework Data model

Platforma Entity Framework je rozdelená do troch základných vrstiev (logická vrstva, konceptuálna vrstva a mapovacia vrstva), pričom každá vrstva je definovaná prostredníctvom definičného jazyka. Na nasledujúcom príklade predstavíme všetky základné vrstvy platformy Entity Framework:

- **Logická vrstva.** Táto vrstva definuje relačné dáta, je definovaná v súbore .edmx pomocou definičného jazyka SSDL a opisuje štruktúru databázových tabuliek a vzťahy medzi nimi (dátové typy, primárne kľúče, cudzie kľúče, atď.).

```
<!-- SSDL content -->
<edmx:StorageModels>
  <Schema Namespace="EFDataModel.Store" Alias="Self" Provider="System.Data.SqlClient"
ProviderManifestToken="2008"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/02/edm/ssdl">
    <EntityContainer Name="EFDataModelStoreContainer">
      <EntitySet Name="FIRMY" EntityType="EFDataModel.Store.FIRMY" store:Type="Tables"
Schema="dbo" />
      <EntitySet Name="OBJEDNAVKY" EntityType="EFDataModel.Store.OBJEDNAVKY"
store:Type="Tables" Schema="dbo" />
      <EntitySet Name="POLOZKY" EntityType="EFDataModel.Store.POLOZKY" store:Type="Tables"
Schema="dbo" />
      <AssociationSet Name="FK_OBJEDNAVKY_FIRMY"
Association="EFDataModel.Store.FK_OBJEDNAVKY_FIRMY">
        <End Role="FIRMY" EntitySet="FIRMY" />
        <End Role="OBJEDNAVKY" EntitySet="OBJEDNAVKY" />
      </AssociationSet>
      <AssociationSet Name="FK_POLOZKY_OBJEDNAVKY"
Association="EFDataModel.Store.FK_POLOZKY_OBJEDNAVKY">
        <End Role="OBJEDNAVKY" EntitySet="OBJEDNAVKY" />
        <End Role="POLOZKY" EntitySet="POLOZKY" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="FIRMY">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Name="ID" Type="int" Nullable="false" StoreGeneratedPattern="Identity" />
      <Property Name="Nazov" Type="nvarchar" MaxLength="500" />
      <Property Name="PocetZamestnancov" Type="int" />
      <Property Name="PriemernaMzda" Type="decimal" Precision="20" Scale="10" />
      <Property Name="Datum" Type="datetime" />
      <Property Name="DomacaFirma" Type="bit" />
    </EntityType>
    <EntityType Name="OBJEDNAVKY">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Name="ID" Type="int" Nullable="false" StoreGeneratedPattern="Identity" />
      <Property Name="Nazov" Type="nvarchar" MaxLength="500" />
      <Property Name="Text" Type="int" />
      <Property Name="Pocet" Type="decimal" Precision="20" Scale="10" />
      <Property Name="DatumPrijatia" Type="datetime" />
      <Property Name="Vybavene" Type="bit" />
      <Property Name="Firmy_ID" Type="int" />
    </EntityType>
```

```
    <EntityType Name="POLOZKY">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Name="ID" Type="int" Nullable="false" StoreGeneratedPattern="Identity" />
      <Property Name="Nazov" Type="nvarchar" MaxLength="500" />
      <Property Name="Pocet" Type="int" />
      <Property Name="Cena" Type="decimal" Precision="20" Scale="10" />
      <Property Name="Datum" Type="datetime" />
      <Property Name="NaSklade" Type="bit" />
      <Property Name="Objednavky_ID" Type="int" />
    </EntityType>
    <Association Name="FK_OBJEDNAVKY_FIRMY">
      <End Role="FIRMY" Type="EFDataModel.Store.FIRMY" Multiplicity="0..1" />
      <End Role="OBJEDNAVKY" Type="EFDataModel.Store.OBJEDNAVKY" Multiplicity="*" />
      <ReferentialConstraint>
        <Principal Role="FIRMY">
<PropertyRef Name="ID" />
        </Principal>
        <Dependent Role="OBJEDNAVKY">
          <PropertyRef Name="Firmy_ID" />
        </Dependent>
      </ReferentialConstraint>
    </Association>
    <Association Name="FK_POLOZKY_OBJEDNAVKY">
      <End Role="OBJEDNAVKY" Type="EFDataModel.Store.OBJEDNAVKY" Multiplicity="0..1" />
      <End Role="POLOZKY" Type="EFDataModel.Store.POLOZKY" Multiplicity="*" />
      <ReferentialConstraint>
        <Principal Role="OBJEDNAVKY">
          <PropertyRef Name="ID" />
        </Principal>
        <Dependent Role="POLOZKY">
          <PropertyRef Name="Objednavky_ID" />
        </Dependent>
      </ReferentialConstraint>
    </Association>
    <Function Name="pObjednavky_Delete" Aggregate="false" BuiltIn="false"
NiladicFunction="false" IsComposable="false"
ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
      <Parameter Name="ID" Type="int" Mode="In" />
    </Function>
    <Function Name="pObjednavky_Insert" Aggregate="false" BuiltIn="false"
NiladicFunction="false" IsComposable="false"
ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
      <Parameter Name="Nazov" Type="nvarchar" Mode="In" />
      <Parameter Name="Text" Type="int" Mode="In" />
      <Parameter Name="Pocet" Type="decimal" Mode="In" />
      <Parameter Name="DatumPrijatia" Type="datetime" Mode="In" />
      <Parameter Name="Vybavene" Type="bit" Mode="In" />
      <Parameter Name="Firmy_ID" Type="int" Mode="In" />
    </Function>
    <Function Name="pObjednavky_Update" Aggregate="false" BuiltIn="false"
NiladicFunction="false" IsComposable="false"
ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
      <Parameter Name="ID" Type="int" Mode="In" />
      <Parameter Name="Nazov" Type="nvarchar" Mode="In" />
      <Parameter Name="Text" Type="int" Mode="In" />
```

```
      <Parameter Name="Pocet" Type="decimal" Mode="In" />
      <Parameter Name="DatumPrijatia" Type="datetime" Mode="In" />
      <Parameter Name="Vybavene" Type="bit" Mode="In" />
      <Parameter Name="Firmy_ID" Type="int" Mode="In" />
    </Function>
  </Schema>
</edmx:StorageModels>
```

Zdrojový kód 1: Definičný jazyk SSDL

- **Konceptuálna vrstva.** V tejto vrstve sa nachádzajú definície jednotlivých tried (entít). Konceptuálna vrstva je definovaná v súbore .edmx pomocou definičného jazyka CSDL, opisuje jednotlivé entity a deklaruje ich typy.

```
<!-- CSDL content -->
<edmx:ConceptualModels>
  <Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm"
xmlns:cg="http://schemas.microsoft.com/ado/2006/04/codegeneration"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
Namespace="EFDataModel" Alias="Self"
xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm/annotation">
    <EntityContainer Name="EFDataModelContainer" annotation:LazyLoadingEnabled="true">
      <EntitySet Name="ObjednavkySada" EntityType="EFDataModel.Objednavky" />
      <EntitySet Name="PolozkySada" EntityType="EFDataModel.Polozky" />
      <AssociationSet Name="FK_POLOZKY_OBJEDNAVKY"
Association="EFDataModel.FK_POLOZKY_OBJEDNAVKY">
        <End Role="Objednavky" EntitySet="ObjednavkySada" />
        <End Role="Polozky" EntitySet="PolozkySada" />
      </AssociationSet>
      <EntitySet Name="FirmySada" EntityType="EFDataModel.Firmy" />
      <AssociationSet Name="FK_OBJEDNAVKY_FIRMY"
Association="EFDataModel.FK_OBJEDNAVKY_FIRMY">
        <End Role="FIRMY" EntitySet="FirmySada" />
        <End Role="Objednavky" EntitySet="ObjednavkySada" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="Objednavky">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Type="Int32" Name="ID" Nullable="false"
annotation:StoreGeneratedPattern="Identity" />
      <Property Type="String" Name="Nazov" MaxLength="500" FixedLength="false"
Unicode="true" />
      <Property Type="Int32" Name="Text" />
      <Property Type="Decimal" Name="Pocet" Precision="20" Scale="10" />
      <Property Type="Boolean" Name="Vybavene" />
      <Property Type="Int32" Name="Firmy_ID" />
      <NavigationProperty Name="NavPolozky"
Relationship="EFDataModel.FK_POLOZKY_OBJEDNAVKY" FromRole="Objednavky" ToRole="Polozky" />
      <Property Type="DateTime" Name="DatumPrijatia" />
      <NavigationProperty Name="NavFirmy" Relationship="EFDataModel.FK_OBJEDNAVKY_FIRMY"
FromRole="Objednavky" ToRole="FIRMY" />
    </EntityType>
    <EntityType Name="Polozky">
      <Key>
        <PropertyRef Name="ID" />
```

```
      </Key>
      <Property Type="Int32" Name="ID" Nullable="false"
annotation:StoreGeneratedPattern="Identity" />
      <Property Type="String" Name="Nazov" MaxLength="500" FixedLength="false"
Unicode="true" />
      <Property Type="Int32" Name="Pocet" />
      <Property Type="Decimal" Name="Cena" Precision="20" Scale="10" />
      <Property Type="DateTime" Name="Datum" />
      <Property Type="Boolean" Name="NaSklade" />
      <Property Type="Int32" Name="Objednavky_ID" />
      <NavigationProperty Name="NavObjednavky"
Relationship="EFDataModel.FK_POLOZKY_OBJEDNAVKY" FromRole="Polozky" ToRole="Objednavky" />
    </EntityType>
    <Association Name="FK_POLOZKY_OBJEDNAVKY">
    <End Type="EFDataModel.Objednavky" Role="Objednavky" Multiplicity="0..1" />
      <End Type="EFDataModel.Polozky" Role="Polozky" Multiplicity="*" />
      <ReferentialConstraint>
        <Principal Role="Objednavky">
          <PropertyRef Name="ID" />
        </Principal>
        <Dependent Role="Polozky">
          <PropertyRef Name="Objednavky_ID" />
        </Dependent>
      </ReferentialConstraint>
    </Association>
    <EntityType Name="Firmy">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Type="Int32" Name="ID" Nullable="false"
annotation:StoreGeneratedPattern="Identity" />
      <Property Type="String" Name="Nazov" MaxLength="500" FixedLength="false"
Unicode="true" />
     <Property Type="Int32" Name="PocetZamestnancov" />
     <Property Type="Decimal" Name="PriemernaMzda" Precision="20" Scale="10" />
      <Property Type="DateTime" Name="Datum" />
      <Property Type="Boolean" Name="DomacaFirma" />
      <NavigationProperty Name="NavObjednavky"
Relationship="EFDataModel.FK_OBJEDNAVKY_FIRMY" FromRole="FIRMY" ToRole="Objednavky" />
    </EntityType>
    <Association Name="FK_OBJEDNAVKY_FIRMY">
      <End Type="EFDataModel.Firmy" Role="FIRMY" Multiplicity="0..1" />
      <End Type="EFDataModel.Objednavky" Role="Objednavky" Multiplicity="*" />
      <ReferentialConstraint>
        <Principal Role="FIRMY">
          <PropertyRef Name="ID" />
        </Principal>
        <Dependent Role="Objednavky">
          <PropertyRef Name="Firmy_ID" />
        </Dependent>
      </ReferentialConstraint>
    </Association>
  </Schema>
</edmx:ConceptualModels>
```

Zdrojový kód 2: Definičný jazyk CSDL

- **Mapovacia vrstva.** V tejto vrstve sa nachádzajú informácie o vzťahoch medzi jednotlivými triedami, databázovými objektmi a reláciami. Mapovacia vrstva je definovaná v súbore .edmx pomocou definičného jazyka MSL a mapuje entity definované prostredníctvom jazykov CSDL na SSDL.

```
<!-- C-S mapping content -->
<edmx:Mappings>
  <Mapping xmlns="http://schemas.microsoft.com/ado/2008/09/mapping/cs" Space="C-S">
    <Alias Key="Model" Value="EFDataModel" />
    <Alias Key="Target" Value="EFDataModel.Store" />
    <EntityContainerMapping CdmEntityContainer="EFDataModelContainer"
StorageEntityContainer="EFDataModelStoreContainer">
      <EntitySetMapping Name="ObjednavkySada">
        <EntityTypeMapping TypeName="EFDataModel.Objednavky">
          <MappingFragment StoreEntitySet="OBJEDNAVKY">
            <ScalarProperty Name="DatumPrijatia" ColumnName="DatumPrijatia" />
            <ScalarProperty Name="Firmy_ID" ColumnName="Firmy_ID" />
            <ScalarProperty Name="Vybavene" ColumnName="Vybavene" />
            <ScalarProperty Name="Pocet" ColumnName="Pocet" />
            <ScalarProperty Name="Text" ColumnName="Text" />
            <ScalarProperty Name="Nazov" ColumnName="Nazov" />
            <ScalarProperty Name="ID" ColumnName="ID" />
          </MappingFragment>
        </EntityTypeMapping>
        <EntityTypeMapping TypeName="EFDataModel.Objednavky">
          <ModificationFunctionMapping>
            <InsertFunction FunctionName="EFDataModel.Store.pObjednavky_Insert">
              <ScalarProperty Name="Firmy_ID" ParameterName="Firmy_ID" />
              <ScalarProperty Name="Vybavene" ParameterName="Vybavene" />
              <ScalarProperty Name="DatumPrijatia" ParameterName="DatumPrijatia" />
              <ScalarProperty Name="Pocet" ParameterName="Pocet" />
              <ScalarProperty Name="Text" ParameterName="Text" />
              <ScalarProperty Name="Nazov" ParameterName="Nazov" />
              <ResultBinding Name="ID" ColumnName="ID" />
            </InsertFunction>
            <UpdateFunction FunctionName="EFDataModel.Store.pObjednavky_Update">
              <ScalarProperty Name="Firmy_ID" ParameterName="Firmy_ID" Version="Current" />
              <ScalarProperty Name="Vybavene" ParameterName="Vybavene" Version="Current" />
              <ScalarProperty Name="DatumPrijatia" ParameterName="DatumPrijatia"
Version="Current" />
              <ScalarProperty Name="Pocet" ParameterName="Pocet" Version="Current" />
              <ScalarProperty Name="Text" ParameterName="Text" Version="Current" />
              <ScalarProperty Name="Nazov" ParameterName="Nazov" Version="Current" />
              <ScalarProperty Name="ID" ParameterName="ID" Version="Current" />
            </UpdateFunction>
            <DeleteFunction FunctionName="EFDataModel.Store.pObjednavky_Delete">
              <ScalarProperty Name="ID" ParameterName="ID" />
            </DeleteFunction>
          </ModificationFunctionMapping>
        </EntityTypeMapping>
      </EntitySetMapping>
      <EntitySetMapping Name="PolozkySada">
        <EntityTypeMapping TypeName="EFDataModel.Polozky">
          <MappingFragment StoreEntitySet="POLOZKY">
            <ScalarProperty Name="Objednavky_ID" ColumnName="Objednavky_ID" />
            <ScalarProperty Name="NaSklade" ColumnName="NaSklade" />
```

```
            <ScalarProperty Name="Datum" ColumnName="Datum" />
            <ScalarProperty Name="Cena" ColumnName="Cena" />
            <ScalarProperty Name="Pocet" ColumnName="Pocet" />
            <ScalarProperty Name="Nazov" ColumnName="Nazov" />
            <ScalarProperty Name="ID" ColumnName="ID" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
      <EntitySetMapping Name="FirmySada">
        <EntityTypeMapping TypeName="EFDataModel.Firmy">
          <MappingFragment StoreEntitySet="FIRMY">
          <ScalarProperty Name="DomacaFirma" ColumnName="DomacaFirma" />
            <ScalarProperty Name="Datum" ColumnName="Datum" />
            <ScalarProperty Name="PriemernaMzda" ColumnName="PriemernaMzda" />
            <ScalarProperty Name="PocetZamestnancov" ColumnName="PocetZamestnancov" />
            <ScalarProperty Name="Nazov" ColumnName="Nazov" />
            <ScalarProperty Name="ID" ColumnName="ID" />
          </MappingFragment>
        </EntityTypeMapping>
      </EntitySetMapping>
    </EntityContainerMapping>
  </Mapping>
</edmx:Mappings>
```

Zdrojový kód 3: Definičný jazyk MSL

Ako je spomenuté v definícii konceptuálnej vrstvy, entita, ako základný prvok technológie Entity Framework, je definovaná pomocou jazyka CSDL a zdedená z triedy **EntityObject**, ktorá zabezpečuje entite informácie o jej zmene (**EntityState**).

Entity sa nachádzajú v súpravách (**EntitySet**) a v tzv. kontexte (**ObjectContext**), ktorý okrem iného zabezpečuje konektivitu na databázu a tiež metódy na prácu s entitami (ako je napr. ich načítavanie z databázy, ukladanie do databázy či vymazávanie z databázy).

Prepojenie dátového modelu Entity Framework a databázy je možné realizovať v dvoch variantoch:

1. Načítanie databázových objektov do modelu Entity Data Model.
2. Vytvorenie samotného modelu Entity Data Model a z neho následné vygenerovanie databázovej štruktúry priamo na databázovom serveri.

Pre väčšinu programátorov bude zrejme zaujímavejšia prvá možnosť, vzhľadom na to, že dátový model už vo väčšine prípadov existuje vytvorený na databáze.

Entity v modeli môžu byť vytvorené viacerými spôsobmi. Najbežnejšou alternatívou je pridanie a transformácia tabuliek z databázy pomocou funkcie **Update Model from Database**. Táto funkcia umožňuje pridať nielen tabuľky, ale aj iné databázové objekty, ako sú napr. pohľady a uložené procedúry. Druhou možnosťou je vytvoriť prázdny model, v ktorom budeme sami vytvárať entitno-relačný (ER) model. To je vhodné napríklad vtedy, keď ako dátový vstup nepoužívame databázu, ale údaje máme uložené v XML súboroch.

2 Návrh EF-modelu

V tejto kapitole sa budeme zaoberať rozdielnymi prístupmi k návrhu EF-modelu:

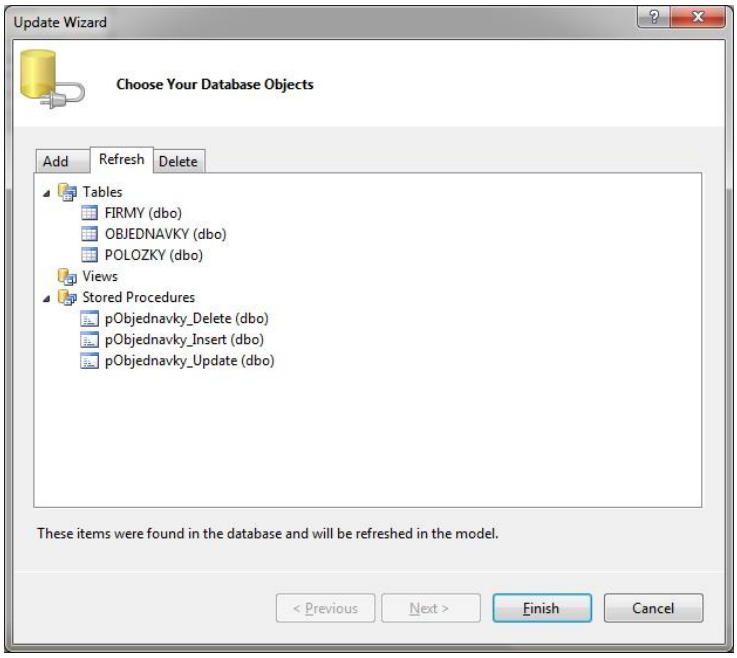
- **Prístup „Najskôr databáza“.** Na platforme Entity Framework možno vytvárať modely viacerými spôsobmi. Pri prvom z nich je najskôr použitá databáza (buď už existujúca, alebo vytvorená ako zbrusu nová) a na jej základe dochádza k vygenerovaniu dátového modelu Entity Data Model. Všetky nasledujúce zmeny v konceptuálnom modeli, ako aj pri mapovaní entít, sa realizujú v prostredí návrhára Entity Data Model Designer. Ak je nutná zmena na strane úložiska údajov, tak sa musia tieto zmeny uskutočniť najprv na strane databázy a až potom je potrebné vykonať aktualizáciu v modeli Entity Data Model. Pri tomto návrhu je podpora zo strany prostredia Visual Studio 2010 realizovaná len pre Microsoft SQL Server, ale zo strany iných hlavných výrobcov databáz (medzi inými Oracle, MySQL, PostgreSQL či Informix) existuje podpora pre ich databázy aj pre toto vývojové prostredie.

- **Prístup „Najskôr model“.** Tento spôsob návrhu EF-modelu bol uvedený až v prostredí produktu Visual Studio 2010. Ako prvý je vytvorený konceptuálny model, následne dochádza k vygenerovaniu databázy a napokon aj k realizácii mapovania z konceptuálneho modelu. Potom je vygenerovaný skript databázy. V prípade zmeny modelu v návrhárovi Entity Data Model Designer sa najskôr vygeneruje databázový skript pre DROP tabuliek a vzápätí aj ďalší skript pre CREATE databázových objektov. Natívne Visual Studio 2010 podporuje tento typ návrhu len pre databázu Microsoft SQL Server, ale od výrobcov databáz Oracle, MySQL a PostgreSQL je tiež podpora pre tento typ návrhu.

- **Prístup „Najskôr kód“.** Ide o najnovší prístup k návrhu EF-modelu, v rámci ktorého programátor definuje model pomocou tried a nastavení a z nich je potom vygenerovaný predmetný EF-model.

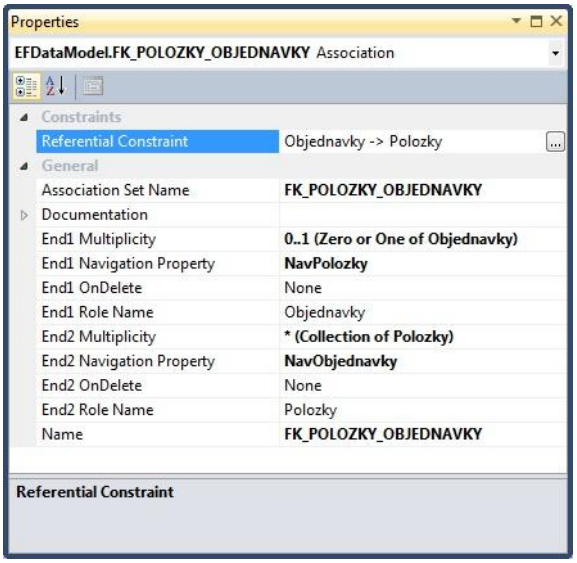
3 Vytvorenie EF-modelu

Najčastejším spôsobom zhotovenia modelu EDM (Entity Data Model) je jeho vytvorenie z existujúcej databázy. Keď už máme vytvorený dátový model na strane databázy, môžeme ho importovať do EDM pomocou funkcie **Update Model from Database**. Po úspešnom importe sú automaticky vygenerované entity a v prípade, že sú na strane databázy definované aj závislosti, tak tieto sú vygenerované aj v EDM ako asociácie medzi prítomnými entitami.



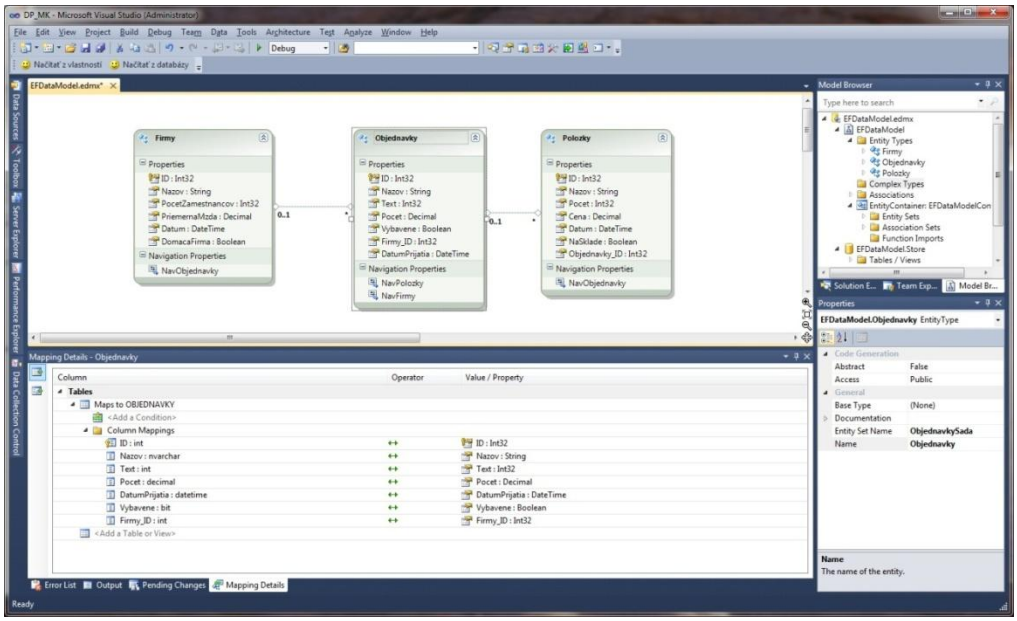
Obr. 3: Pridanie a aktualizácia databázových objektov v modeli EDM

Po vytvorení modelu sa dá pomocou nástroja **Mappings Detail** v záložke **Tables** zmeniť mapovanie jednotlivých stĺpcov databázovej tabuľky na entite, resp. možno tiež editovať mapovaciu vrstvu.



Obr. 4: Nastavenie asociácie v modeli EDM

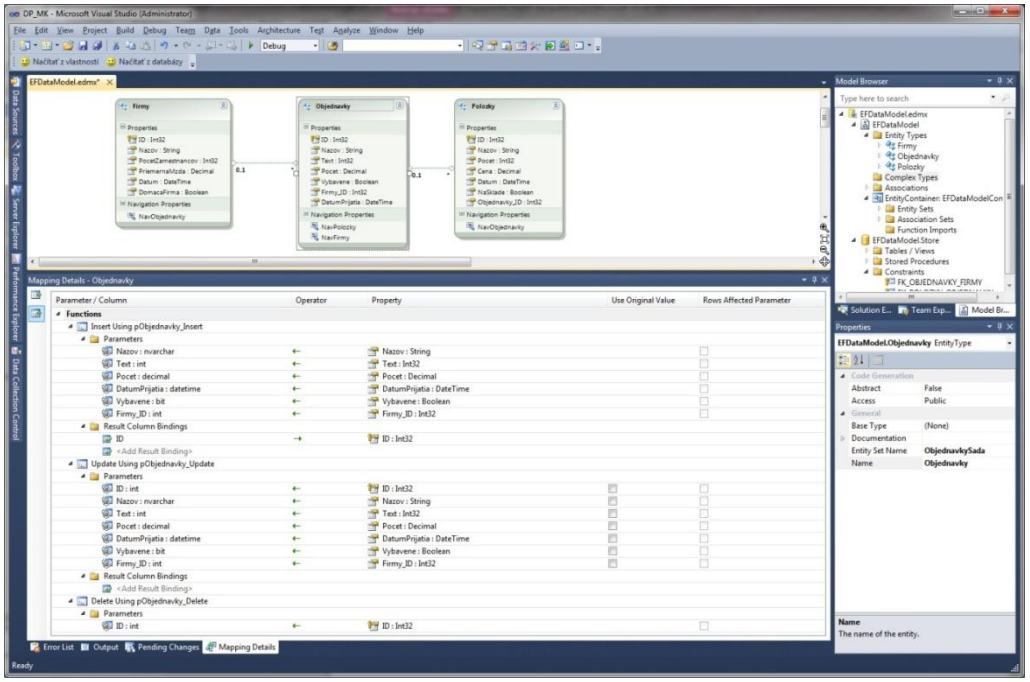
Po pridaní entít do modelu EDM môžeme s nimi vykonávať rôzne činnosti, ku ktorým patrí napríklad presúvanie vlastností, ich zmena či možnosť nastavovať vlastnosti asociácií medzi entitami. Aj keď sú asociácie prevzaté z databázového prostredia, možno ich pridávať, editovať, mazať a upravovať ich kardinalitu a parcialitu (resp. multiplicitu), a to presne podľa potrieb vyvíjanej softvérovej aplikácie.



Obr. 5: Model EDM - mapovanie entity

Na záložke **Functions** môžeme entite namapovať uložené procedúry pre operácie **Insert**, **Update** a **Delete**. Hoci mapovanie a používanie uložených procedúr má v modeli EDM svoje osobitosti, v tomto prípade ide o špecifickú funkčnosť, ktorá je viazaná len na vloženie novej inštancie entity do databázy, jej aktualizáciu, alebo vymazanie pomocou príslušných uložených procedúr. Na tejto záložke možno mapovať aj viacero entít na jednu tabuľku, resp. naopak, viacero tabuliek na jednu entitu.

Ako si môžeme všimnúť na obr. 5, každá entita obsahuje vlastnosti (ktoré korešpondujú so stĺpcami príslušnej tabuľky) a rovnako aj asociačné vzťahy. V spodnej časti je vizuálne zobrazené mapovanie entity a tabuľky definované v jazyku MSL. Každá entita v modeli EDM musí mať definovaný kľúč **EntityKey**.



Obr. 6: EDM - mapovanie uložených procedúr pre operácie **Insert**, **Update** a **Delete**

Pri práci s entitou táto prechádza rôznymi stavmi, ktoré sú charakterizované jej vlastnosťou **EntityState**. Táto vlastnosť hovorí o tom, v akom stave sa entita práve nachádza a na jej stave je potom volaná uložená procedúra mapovaná na operácie **Insert**, **Update** a **Delete**. Teraz opíšeme jednotlivé stavy entity:

- **Detached** – v tomto stave sa entita nachádza, keď je vytvorená, ale jej stav nie je sledovaný prostredníctvom **ObjectStateManager**. Je to väčšinou ihneď po vytvorení entity, ešte pred jej pridaním do kontextu, alebo po zavolaní metódy **Detach** je entita vyňatá zo sledovania.

- **Unchanged** – entita je v tomto stave vtedy, keď od posledného pripojenia do kontextu alebo od posledného volania metódy **SaveChanges** nebola modifikovaná.

- **Added** – po vytvorení entity a jej pripojení do kontextu sa dostane do stavu **Added**. V tomto stave sa nachádza až do volania metódy **SaveChanges**. Po volaní tejto metódy sa entita dostane do stavu **Unchanged**. V tomto stave hodnoty v **Object State Entry** nie sú originálne. Pri volaní metódy **SaveChanges** a v prípade, že máme v modeli EDM definovanú uloženú procedúru pre operáciu **Insert**, je volaná táto procedúra.

- **Deleted** – po volaní metódy **DeleteteObject** s parametrom entity na kontexte sa entita dostane do stavu **Deleted**. Po uložení zmien prejde entita vzápätí do stavu **Detached**. Pri volaní metódy **SaveChanges** a v prípade, že máme v modeli EDM definovanú uloženú procedúru pre operáciu **Delete**, je volaná táto procedúra.

- **Modified** – po zmene ktoréhokoľvek skalárneho atribútu, pokiaľ nie je zavolaná metóda **SaveChanges**, je entita v stave **Modified**. Po uložení zmien sa entita dostane do stavu **Unchanged**. Pri volaní metódy **SaveChanges** a v prípade, že máme v modeli EDM definovanú uloženú procedúru pre operáciu **Update**, je volaná táto procedúra.

4 Prispôsobenie EF-modelu

4.1 Parciálne triedy

Všetky objekty zdedené z **ObjectContext** sú implementované ako parciálne triedy[1], takže je možné im doprogramovať vlastnú funkcionalitu. V prípade, že potrebujeme implementovať vlastné nastavenia triedy, nemusíme prepisovať pôvodnú triedu, ale stačí, keď vytvoríme triedu s rovnakým názvom v separátnom zdrojovom súbore a do jej tela zapíšeme požadovaný zdrojový kód.

[1] Parciálna trieda je trieda, ktorej deklarácia je rozdelená do viacerých zdrojových súborov. Takáto kompozícia je výhodná najmä pri racionálnej fragmentácii automaticky generovaného zdrojového kódu (ako výsledku práce prekladača) a používateľom dodaného zdrojového kódu (ako výsledku práce programátora).

4.2 Parciálne metódy

Rovnako ako parciálne triedy sa dajú implementovať aj parciálne metódy, ktoré sú v princípe rovnaké ako parciálne triedy. Implementácia parciálnej metódy spočíva vo vytvorení metódy s rovnakým názvom a vlastnou implementáciou v rôznych kontextoch.

```
public partial class EFDataModelContainer : ObjectContext
{
    partial void OnContextCreated();
}
```

Zdrojový kód 4: Parciálna trieda s parciálnou metódou

5 Dopytovací jazyk Entity SQL

Entity SQL je dopytovací jazyk, ktorý umožňuje vykonávať dopytovanie vo vzťahu ku konceptuálnemu modelu platformy Entity Framework. Dopytovací jazyk Entity SQL vychádza principiálne z deklaratívneho jazyka SQL. Dopytovanie je realizované nad entitami. Z toho vyplýva, že nie je podstatné, nad akou databázou je model vytvorený, pretože o preklad do samotného natívneho jazyka SQL databázy sa stará poskytovateľ Entity Framework Provider. Entity SQL vracia typovo silné objekty, ktoré sú inštanciami entitných typov.

Dopytovací jazyk Entity SQL sa využíva predovšetkým v prostredí iných .NET-kompatibilných programovacích jazykov ako je C# a Visual Basic (teda najmä v C++/CLI a F#), keďže tieto jazyky nemajú implementovanú technológiu LINQ, a jazyk Entity SQL sa tak stáva jediným spôsobom dopytovania v týchto prostrediach.

6 Spôsoby dopytovania

V technológii Entity Framework existuje viacero možností dopytovania údajov. Jedným z klasických spôsobov je napísanie SQL dopytu pomocou metód triedy **ObjectQuery<T>**, ktorá využíva schopnosti už spomenutého jazyka Entity SQL. Druhou možnosťou je tvorba a následné spracovanie dopytov skonštruovaných v jazyku LINQ. V oboch prípadoch je výsledkom tzv. materializovaný objekt. Je to objekt obsahujúci vopred pripravenú a načítanú súpravu údajov, s ktorými ďalej pracujeme. Čiže dopytovanie v aplikácii, ktorá už pracovala s istou údajovou súpravou, neprebieha ďalej nad samotnou databázou, ale nad touto načítanou súpravou. To nám umožňuje oveľa pružnejšie pracovať s rozsiahlymi tabuľkami (resp. zoznamami), pretože tieto sa už nachádzajú v pamäťovom priestore nášho počítača. (Samozrejme, ďalším plusom je čiastočné uvoľnenie sieťového spojenia medzi klientom a databázovým serverom, ktoré by bolo inak zaťažené opakovaným načítavaním údajov.) Na druhej strane si však musíme dávať pozor na externé zmeny v databáze a tieto v situáciách, keď je to žiaduce, podchytiť opätovným obnovením.

Pri využití dopytovania prostredníctvom triedy **ObjectQuery<T>** môžeme využiť možnosť priameho zápisu dopytu v jazyku Entity SQL ako textového argumentu konštruktora, čím získame výhodu

skladania tohto reťazca. Druhou možnosťou zápisu je využitie metód triedy **ObjectQuery<T>**, ktoré nahrádzajú príkazy jazyka Entity SQL.

Tab. 1: Prevodná tabuľka medzi príkazmi jazyka Entity SQL a metódami triedy **ObjectQuery<T>**

Entity SQL príkaz	**ObjectQuery<T>** metóda
DISTINCT	Distinct
EXCEPT	Except
GROUP BY	GroupBy
INTERSECT	Intersect
OFTYPE	OfType
ORDER BY	OrderBy
SELECT	Select
SELECT VALUE	SelectValue
SKIP	Skip
TOP and LIMIT	Top
UNION	Union
UNION ALL	UnionAll
WHERE	Where

```
Using (EFDataModelContainer dc = new EFDataModelContainer())
{
   ObjectQuery<Okres> oqESQL = new ObjectQuery<Okres>("SELECT VALUE okresy FROM
                  EFDataModelContainer.OkresSada AS okresy", dc);
   ObjectQuery<DbDataRecord> oqMetoda = oqESQL.Select("it.Okres");
}
```

Zdrojový kód 5: Použitie **ObjectQuery<T>** s Entity SQL

Druhým spôsobom dopytovania je využitie technológie LINQ. Aj tu sú k dispozícii dva druhy dopytovania:

1. Dopytovanie pomocou zabudovaných operátorov jazyka LINQ.
2. Dopytovanie pomocou ekvivalentných rozširujúcich metód jazyka LINQ.

```
Using (EFDataModelContainer dc = new EFDataModelContainer())
{
     IEnumerable<Objednavky> lm = dc.ObjednavkySada
              .Where(c => c.Firmy_ID == 2)
              .OrderBy(d => d.Nazov)
              .Select(d => d);
}
```

Zdrojový kód 6: Dopyt realizovaný pomocou zreťazených metód jazyka LINQ

Jednotný dopytovací jazyk LINQ významne uľahčuje získavanie údajov z rôznych dátových zdrojov vzhľadom na široké možnosti využitia výrazových stromov, ktoré implikuje vo forme lambda-výrazov.

7 Kompilované dopyty

V situácii, keď potrebujeme realizovať značný počet štrukturálne jednoduchých dopytov nad platformou Entity Framework, je mimoriadne výhodné použiť kompilované dopyty, ktoré výrazne zvyšujú rýchlosť svojho spracovania. Kompilovaný dopyt odstraňuje zdržanie spôsobené kompilovaním dopytu pred každým spustením tým, že ho skompiluje raz a uchová do vyrovnávacej pamäte pri prvom spustení, a potom už používa len túto skompilovanú podobu. Z uvedeného vyplýva, že pokiaľ nie je dopyt prvýkrát zavolaný, nedeje sa vôbec nič. Dôležitým prvkom v tomto prípade je, že ak vytvoríme dopyt založený na kompilovanom dopyte, resp. ak zmeníme dopyt a pošleme na server niečo, čo zmení tento dopyt, stráca sa výhoda kompilácie. Čiže vždy musíme poslať na server ten istý dopyt. Jedinou možnosťou variability dopytu je použitie argumentov, ktoré sú odovzdané funkcii, alebo poslanie kompilovaného dopytu na databázu, ktorý je stále rovnaký, pričom je potrebné vykonávať ďalšie zmenné dopyty nad EF-modelom v pamäti pomocou konštrukcií LINQ to Objects.

```
Func<EFDataModelContainer, long, IQueryable<Objednavky>>
   objednavky = System.Data.Objects.CompiledQuery.Compile<EFDataModelContainer, long,
    IQueryable<Objednavky>>(
    (ctx, id_firmy) => from d in ctx.ObjednavkySada
                       where d.Firmy_ID == id_firmy
                       select d);
   using (EFDataModelContainer dc = new EFDataModelContainer())
   {
      var s = objednavky.Invoke(dc, 2);
      List<Objednavky> o = s.ToList();
   }
```

Zdrojový kód 7: Kompilovaný dopyt

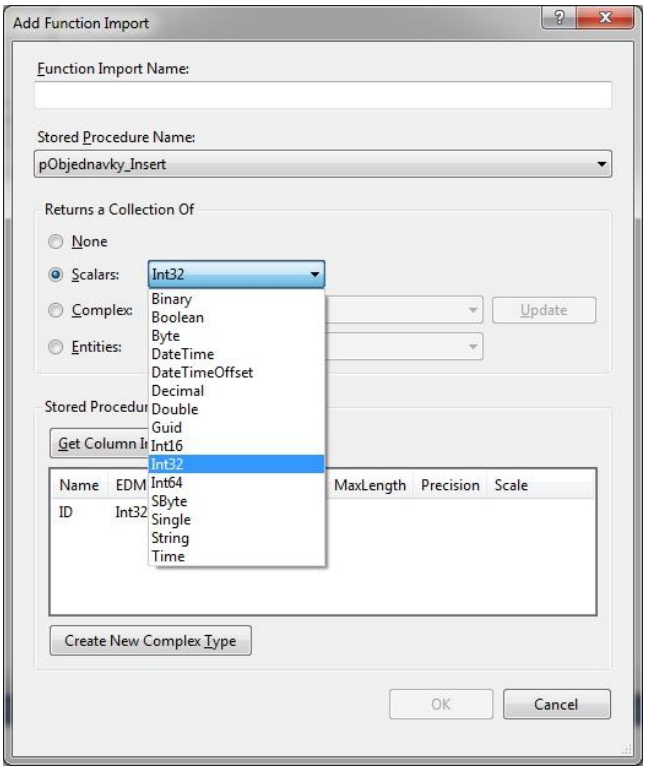
8 Uložené procedúry v prostredí Entity Framework

Model EDM mapuje na entity uložené procedúry ako **read**, **insert**, **update** a **delete**. Vďaka návrhárovi je teraz jednoduché namapovať na entitu uložené procedúry. Uložené procedúry a funkcie sú realizované v jazyku SSDL ako funkcie importované do konceptuálneho modelu, zatiaľ čo databázové pohľady sú aktualizované pomocou mapovania funkcií. Aj keď je možné namapovať uloženú procedúru na entitu podľa predchádzajúcich riadkov, väčšinu uložených procedúr nie je možné namapovať na entitu, ale je možné ich namapovať na tzv. skalárne typy alebo komplexné typy. V tomto prípade možno volať uloženú procedúru priamo zo zdrojového kódu ako metódu triedy **ObjectContext**.

Databázové objekty sa dajú namapovať funkciou **Add Function Import** nielen na entity, ale je možné si vytvoriť aj ďalšie objekty ako:

- **Skalárny typ.** Keď uložená procedúra vracia jeden konkrétny typ, napr. pri vracaní počtu riadkov, alebo sumy určitých riadkov, je možné (a aj oveľa vhodnejšie) ako na entitu, ju mapovať v prostredí modelu EDM na skalárny dátový typ.

- **Komplexný typ**. Za okolností, keď uložená procedúra vracia štruktúru údajov, ktoré sa nedajú mapovať na entitu, je možné vytvoriť vlastnú štruktúru, na ktorú je zase možné túto uloženú procedúru mapovať. Komplexný typ obsahuje vlastnosti rôzneho druhu, ale neobsahuje vlastnosť **EntityKey**, a preto ho nie je možné manažovať pomocou triedy **ObjectContext**.



Obr. 7: Ponuka dialógového okna **Add Function Import**

Výsledkom tohto mapovania je funkcia v jazyku CSDL, ktorá môže byť tiež vytvorená ako metóda triedy **ObjectContext**. Majme procedúru, ktorá vracia nejaké údaje:

```
from s in dc.pObjednavky(25) where s.Nazov.Contains("Nazov") select s
```

Zdrojový kód 8: Dopyt jazyka LINQ s využitím uloženej procedúry

Samotná uložená procedúra je spracovaná na strane servera, ale ostatná funkčnosť dopytu je vykonávaná v pamäti na strane klienta. Pri tomto príklade (Zdrojový kód 8) teda uložená procedúra vráti z databázy tabuľku **Objednávky** obmedzenú parametrom a následne je výber obmedzený podľa kritéria v klauzule **where**. Z toho vyplýva, že je mimoriadne žiaduce, a to najmä pri rozsiahlych tabuľkách, starostlivo dbať na efektívne používanie uložených procedúr. V tomto smere sa ako dobrý nápad osvedčilo zabaliť uloženú procedúru do databázového pohľadu a ten namapovať na entitu v modeli EDM. V takomto prípade by sa samotná uložená procedúra vykonala na serveri a dopyt spracovaný nad pohľadom by načítal už len samotné požadované údaje.

Uložená procedúra je schopná tiež načítať údaje z databázovej tabuľky do inej štruktúry, než akou je entita. Na tento účel slúži metóda **ExecuteStoreQuery<T>**, ktorá vracia výsledok typu **ObjectResult<T>**, kde **T** je nami definovaný typ, na ktorý chceme uloženú procedúru namapovať.

```
string query = "select o.Nazov as NazovObjednavky, o.Pocet as ObjednavkaPocet, o.Text as
    ObjednavkaText, o.DatumPrijatia as ObjDatumPrijatia, p.Cena as PolozkaCena, p.Nazov as
    PolozkaNazov, p.Pocet as PolozkaPocet, p.Datum as PolozkaDatum, f.Nazov as FirmaNazov,
    f.PocetZamestnancov as FirmaPocetZamestnancov, f.PriemernaMzda as FirmaPriemernaMzda
    from OBJEDNAVKY as o LEFT OUTER JOIN POLOZKY as p on p.Objednavky_ID = o.ID LEFT OUTER
    JOIN FIRMY as f on o.Firmy_ID = f.ID ORDER BY f.Nazov;";

using (EFDataModelContainer dc = new EFDataModelContainer())
{
   List<Objednavky> result = dc.ExecuteStoreQuery<Objednavky>(query).ToList();
}
```

Zdrojový kód 9: Dopyt s využitím metódy **ExecuteStoreQuery<T>**

9 EF-dopyt a fázy jeho tvorby

Vytváranie dopytu v prostredí platformy Entity Framework sa skladá z určitých fáz, ktoré keď pochopíme a optimalizujeme, dosiahneme výraznú úsporu času pri načítavaní údajov a ich následnom spracovaní.

Tab. 2: Fázy tvorby EF-dopytu

Operácia	Trvanie	Potreba	Opis
Načítanie metadát	Stredné	Pri spustení aplikácie	Metadáta sú načítané pri spustení aplikácie. Sú dostupné pre každú inštanciu triedy **ObjectContext**.
Otvorenie databázového spojenia	Stredné	Podľa potreby	Vzhľadom na to, že otvorenie spojenia je operácia náročná na systémové prostriedky, platforma Entity Framework otvára spojenie podľa aktuálnej potreby.
Generovanie pohľadov	Vysoké	Pri spustení aplikácie	Pred vykonaním dopytu, alebo uložením údajov do databázy musí Entity Framework zakaždým vygenerovať pohľad pre prístup do databázy. Toto generovanie zaberá veľmi dlhý čas, preto možno vopred generovať pohľady už počas návrhovej etapy a ihneď ich pridať do projektu.
Príprava dopytu	Stredné	Vždy pri novom dopyte	Príprava dopytu spočíva v generovaní príkazového stromu a mapovaní metadát. Z dôvodu, že príkazy jazyka Entity SQL sú kešované, ich ďalšie vykonanie je podstatne kratšie. Rovnako možno použiť kompilované dopyty.
Vykonanie dopytu	Nízke	Raz pre každý dopyt	Vykonanie dopytu sa realizuje na strane dátového poskytovateľa platformy ADO.NET. Väčšina dátových zdrojov podporuje plánovanie dopytov.
Načítanie a validácia typov	Nízke	Raz pre každú inštanciu **ObjectContext**	Načítanie a validácia typov je realizovaná z typov, ktoré sú definované v konceptuálnom modeli.
Sledovanie zmien	Nízke	Raz pre každý objekt, ktorý vracia dopyt	Ak je nastavený **NoTracking**, tento faktor nemá žiaden vplyv. Pri ostatných možnostiach je sledovanie zabezpečované správcom **ObjectStateManager**.
Materializovanie objektov	Stredné	Raz pre každý objekt, ktorý vracia dopyt	Pre údaje vrátené čítačom **DbDataReader** je vytvorený objekt, ktorému sú nastavené tieto údaje ako vlastnosti. Táto fáza nemá vplyv na výkon vtedy, keď je enumerácia **MergeOption** nastavená na hodnotu **AppendOnly** alebo **PreserveChanges**.

10 Výhody EF oproti iným technológiám ADO.NET (DataSet, SqlDataReader)

Nevýhodou klasických prístupov k dátam v prostredí .NET Framework 4.0, ku ktorým patrili dátové čítače (**DataReaders**) alebo dátové súpravy (**DataSets**), je veľká náročnosť na napísaný zdrojový kód a veľký počet vyskytujúcich sa chýb pri pretypovaní stĺpcov, keď bolo potrebné najskôr zistiť, akého dátového typu je daný stĺpec v tabuľke, a potom nájsť alternatívny typ v použitom programovacom jazyku. Hlavne pri použití databáz tretích strán (Oracle, MySQL, PostgreSQL, atď.) to pôsobilo často nemalé komplikácie.

V prípade technológie Entity Framework je tento problém vyriešený už samotným systémom a programátor sa preto nemusí zaťažovať hľadaním alternatívnych dátových typov. Systém nám poskytne už vytvorený ORM model, v ktorom sú entity reprezentované ako triedy programovacieho jazyka a každý atribút entity je reprezentovaný ako vlastnosť, resp. metóda tejto triedy. Je samozrejme možná manuálna úprava týchto transformovaných atribútov podľa vlastných potrieb. Z pohľadu programátora sa javí pohľad na databázu ako pohľad na štruktúrovaný objektový model, skladajúci sa z tried a ich atribútov, ku ktorým pristupuje ako ku klasickým triedam a ich inštanciám.

Ďalšia devíza platformy Entity Framework spočíva v tom, že vygenerované objekty sú pod úplnou kontrolou správcu pamäte (Garbage Collector), a tak je už odstránená chyba, keď aplikácia alokovala pamäť, ktorá postupnou prácou s aplikáciou rástla až do vyčerpania dostupnej pamäťovej kapacity.

Použitá literatúra

- David Obando, Eric Dettinger and others. 2012. *Performance Considerations for Entity Framework 5*. [Online] apríl 2012. http://msdn.microsoft.com/en-us/data/hh949853.
- Dawson, Brian. 2008. *ADO.NET Entity Framework Performance Comparison*. [Online] Microsoft, 27. marec 2008. http://blogs.msdn.com/b/adonet/archive/2008/03/27/ado-net-entity-framework-performance-comparison.aspx.
- Hanák, Ján. 2009. *Praktické paralelné programovanie v jazykoch C# 4.0 a C++*. Brno : Artax a.s., 2009. s. 129. ISBN 978-80-87017-06-7.
- Jenings, Roger. 2009. *Professional ADO.NET 3.5 with LINQ and the Entity Framework*. Indianapolis : Wiley Publishing, Inc., 2009. s. 634. ISBN 978-0-470-18261-1.
- Lerman, Julia. 2010. *Programming Entity Framework*. 2nd Edition. Sebastopol : O'Reilly, 2010. s. 1. ISBN 978-0-596-80726-9.
- Paolo Pialorsi, Marco Russo. 2009. *Microsoft LINQ - Kompletní průvodce programátora*. Brno : Computer Press, a.s., 2009. s. 615. ISBN 978-80-251-2735-3.

Ing. Marián Kubica
je absolventom študijného programu Manažérske rozhodovanie a informačné technológie (MRIT) na Fakulte hospodárskej informatiky Ekonomickej univerzity v Bratislave.

OD študentov
PRE študentov

FAKULTA HOSPODÁRSKEJ INFORMATIKY EU V BRATISLAVE

Microsoft