

Vytvoření jednoduché aplikace a zobrazující data a její přesun do cloudu Windows Azure

Toto praktické cvičení vám na jednoduchém příkladu ukáže, jak si můžete vytvořit vlastní jednoduchou aplikaci zobrazující data a poté ji přenést přenos do cloud prostředí Windows Azure. Data budou uložena v relační databázi a zpřístupněná pro konzumaci na zařízení ve formátu OData.

Cvičení lze provést ve třech variantách:

- Aplikace zobrazující data na webové stránce – provedte části 1 až 5
- Aplikace nabízející data, která jsou zobrazena na Windows Phone zařízení – provedte části 1 až 6
- Aplikace nabízející data, která jsou zobrazena na desktopové aplikaci napsané pomocí WPF – provedte části 1 až 5 a část 7

Doba potřebná k provedení celého cvičení je v ideální případě cca 3 hodiny. Pokud si chcete ušetřit práci, kód řešení po ukončení Kapitoly 3 a finální kód po ukončení celého cvičení jsou přiloženy.

Více informací o migraci aplikací na Windows Azure můžete získat z [našeho kurzu](#).

Obsah

Vytvoření jednoduché aplikace a jej zobrazující data z cloudu.....	1
1. Potřebné vybavení.....	2
2. Vytvoření databáze.....	2
3. Vytvoření aplikační vrstvy	10
4. Přenos databáze do cloudu	22
5. Přenos aplikační vrstvy do cloudu	32
6. Vytvoření Windows Phone aplikace.....	37
7. Vytvoření desktopové aplikace	44

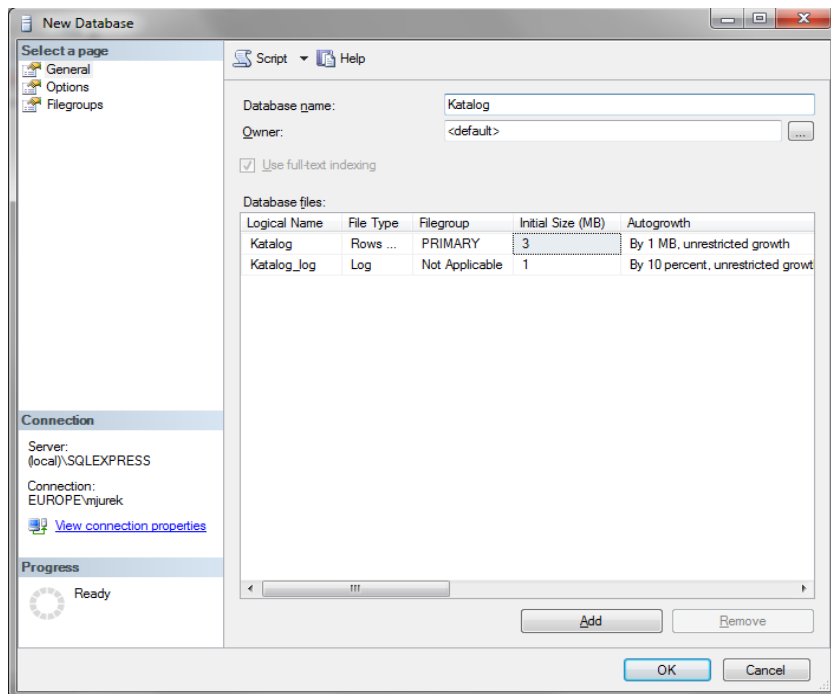
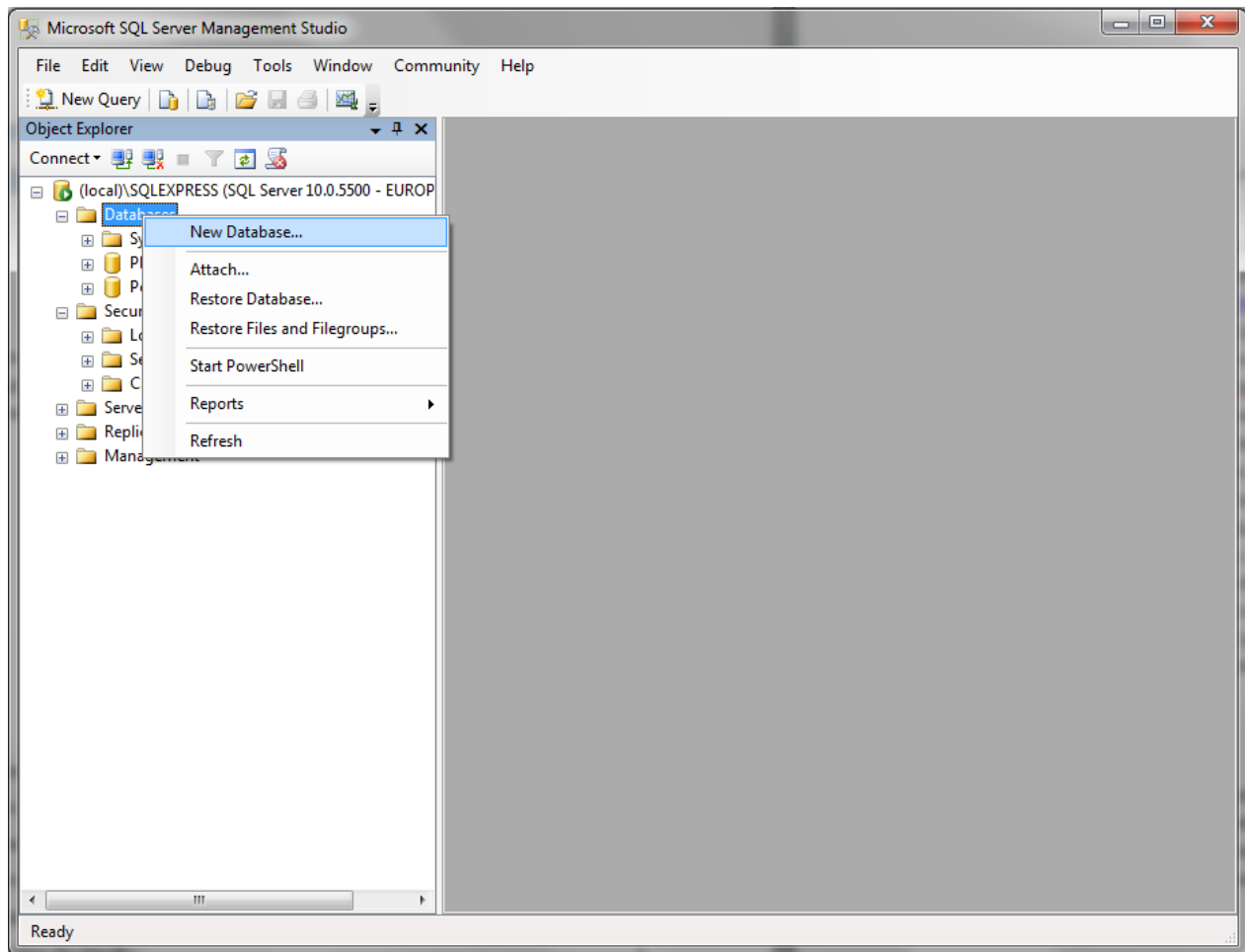
1. Potřebné vybavení

Veškeré potřebné nástroje a vybavení pro vytvoření tohoto příkladu lze stáhnout zdarma. Konkrétně budete potřebovat:

- Visual Studio 2010 – buď libovolnou komerční edici (trial verze se dají stáhnout [zde](#)) anebo lze použít bez platné edice anebo omezené bezplatné edice [Visual Web Developer 2010 Express](#) a [Visual Studio 2010 Express for Windows Phone](#). Veškeré obrazovky v tomto návodu byly vytvořeny pomocí Visual Studia Ultimate, vzhled vašeho vývojového prostředí se může mírně lišit
- Windows Azure SDK a Windows Azure Tools for Microsoft Visual Studio – obojí si nainstalujete naráz pomocí Web Platform Installeru, odkaz je [zde](#).
- SQL Server 2005 nebo vyšší, libovolnou verzi – bezplatnou Express, plnou, zkušební – je to celkem jedno. Pokud nevíte nebo nemáte, doporučujeme verzi [SQL Express 2008 R2 Express](#)
- SQL Server Management Studio 2008 R2 SP1 – na rozdíl od verze databáze, která není až tak důležitá, verze Management Studia důležitá je. Pouze [tato verze](#) umí generovat skripty, které jsou zaručeně kompatibilní s SQL Azure
- Zřízený účet na Windows Azure. Nově je možné využít tzv. Free Trial, kdy máte po 90 dní garantováno bezplatné užívání zdrojů Windows Azure a ani po uplynutí 90 dnů vás nepřekvapí žádné nečekané poplatky. Stále ovšem budete potřebovat bankovní kartu s povolenými internetovými platbami – slouží pouze jako autentizační prostředek a není na ni nic účtováno. Více informací o zhruba 10minutové registraci [zde](#), pokud máte MSDN předplatné, přečtěte si též [toto](#).

2. Vytvoření databáze

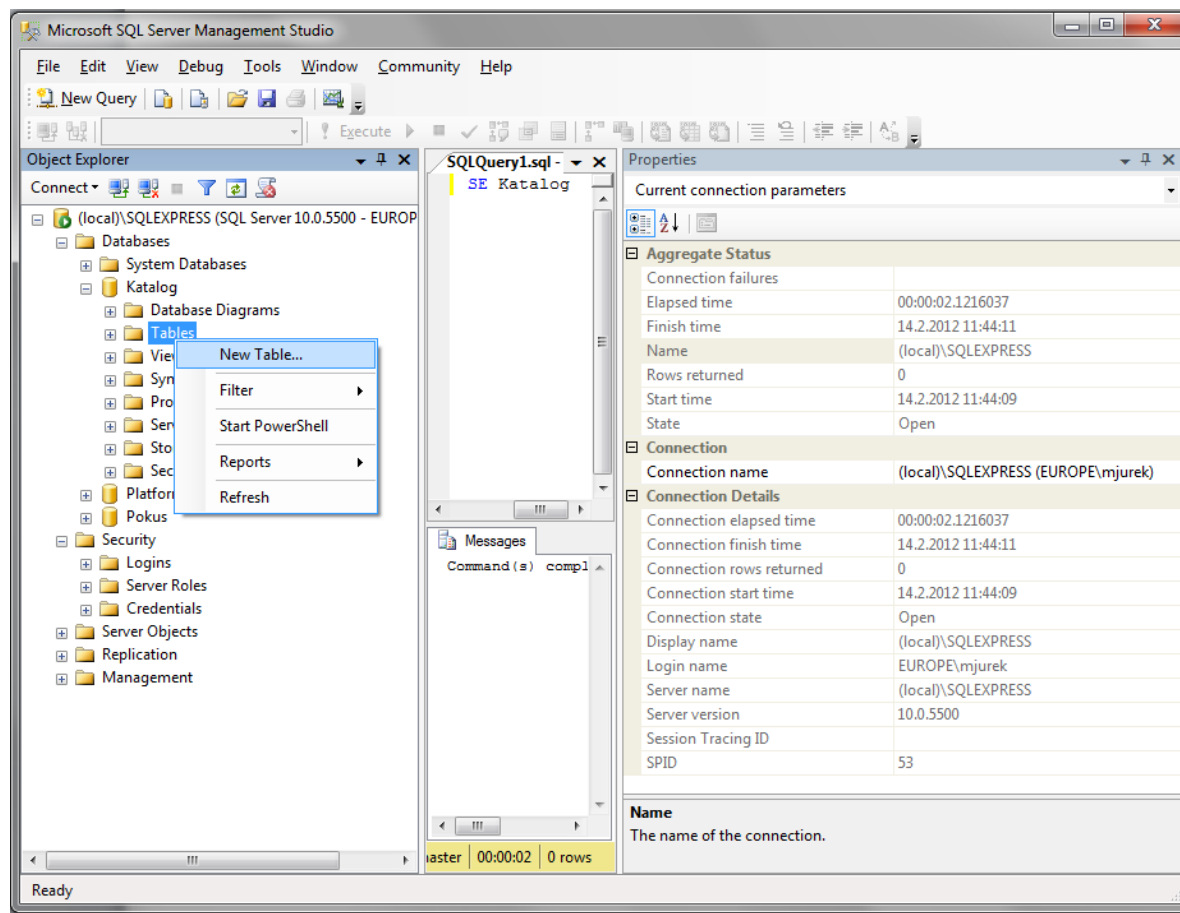
Aplikaci začneme vytvářet „odspodu nahoru“, začneme tedy databází. Naše databáze bude velmi jednoduchá a bude mít dvě tabulky – Kategorie a Produkty (samozřejmě si můžete vymyslet jiný podobný příklad). Otevřte si SQL Management Studio, připojte se ke svojí lokální databázové instanci (např. .\SQLEXPRESS) a vytvořte novou prázdnou databázi Katalog:



Kdo dává přednost skriptu, může spustit

```
CREATE DATABASE Katalog  
GO
```

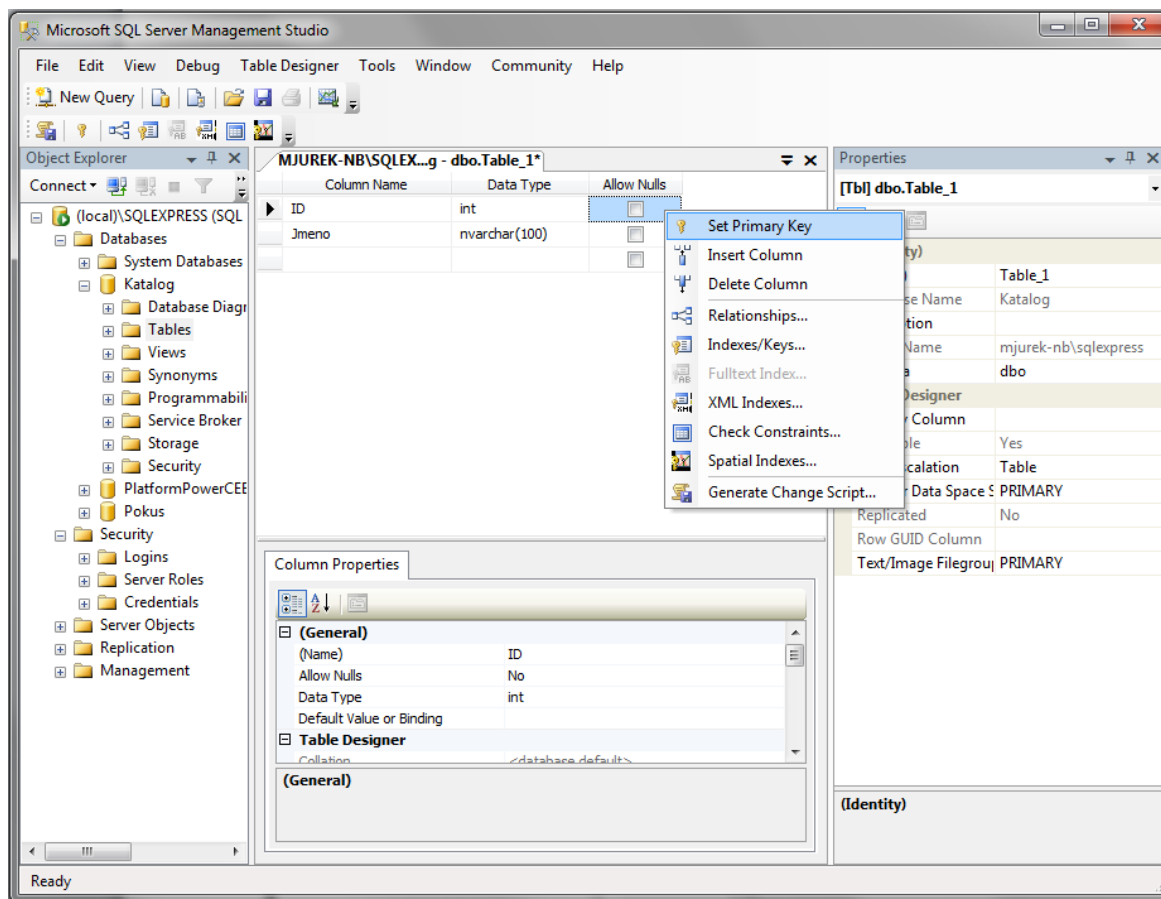
V databázi Katalog nyní vytvoříme novou tabulku Kategorie:



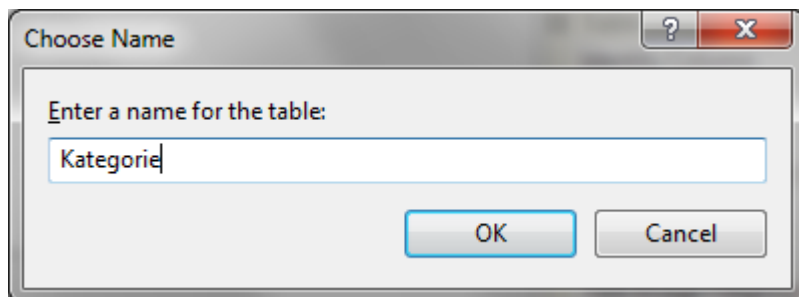
V ní vytvoříme dva sloupce:

- ID, int
- Jmeno, nvarchar(100)

Oba sloupce nepovolují hodnotu NULL a sloupec ID slouží jako primární klíč:



Nyní stiskneme Save a tabulku uložíme:

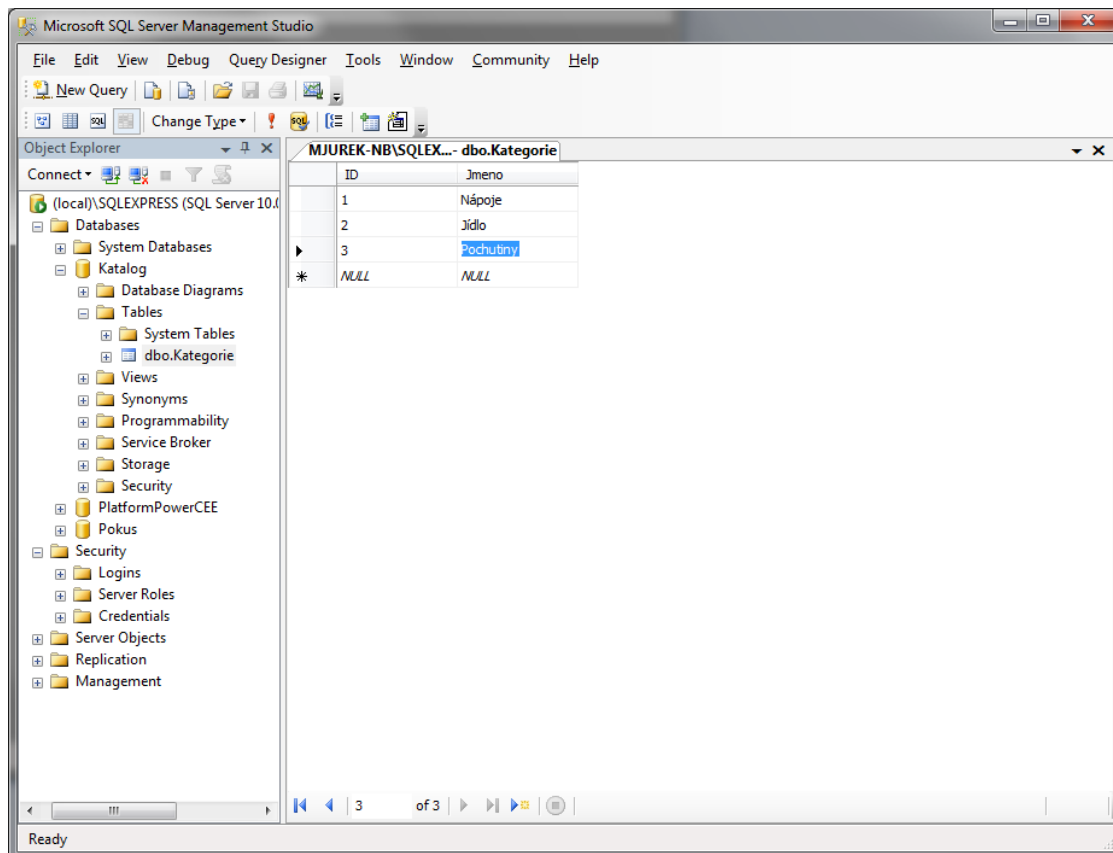
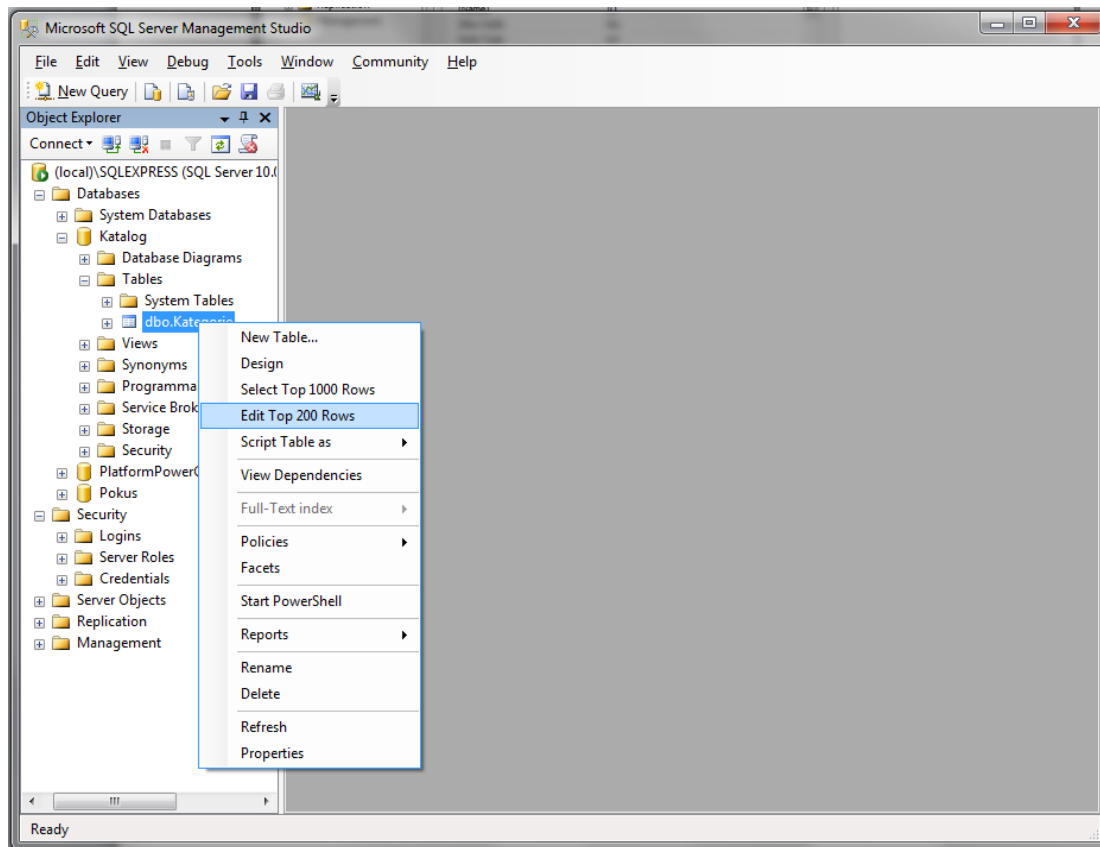


Kdo dává přednost skriptu, může místo výše uvedených akcí spustit následující skript:

```
CREATE TABLE dbo.Kategorie
(
    ID int NOT NULL,
    Jmeno nvarchar(100) NOT NULL,
    CONSTRAINT PK_Kategorie PRIMARY KEY CLUSTERED (ID)
)
```

GO

Nyní vložíme tři vzorové kategorie. Buď můžeme využít přímo editaci v Management Studiu:



Případně opět můžete použít následující skript:

```
INSERT dbo.Kategorie (ID, Jmeno) VALUES (1, N'Nápoje')
INSERT dbo.Kategorie (ID, Jmeno) VALUES (2, N'Jídlo')
INSERT dbo.Kategorie (ID, Jmeno) VALUES (3, N'Pochutiny')
GO
```

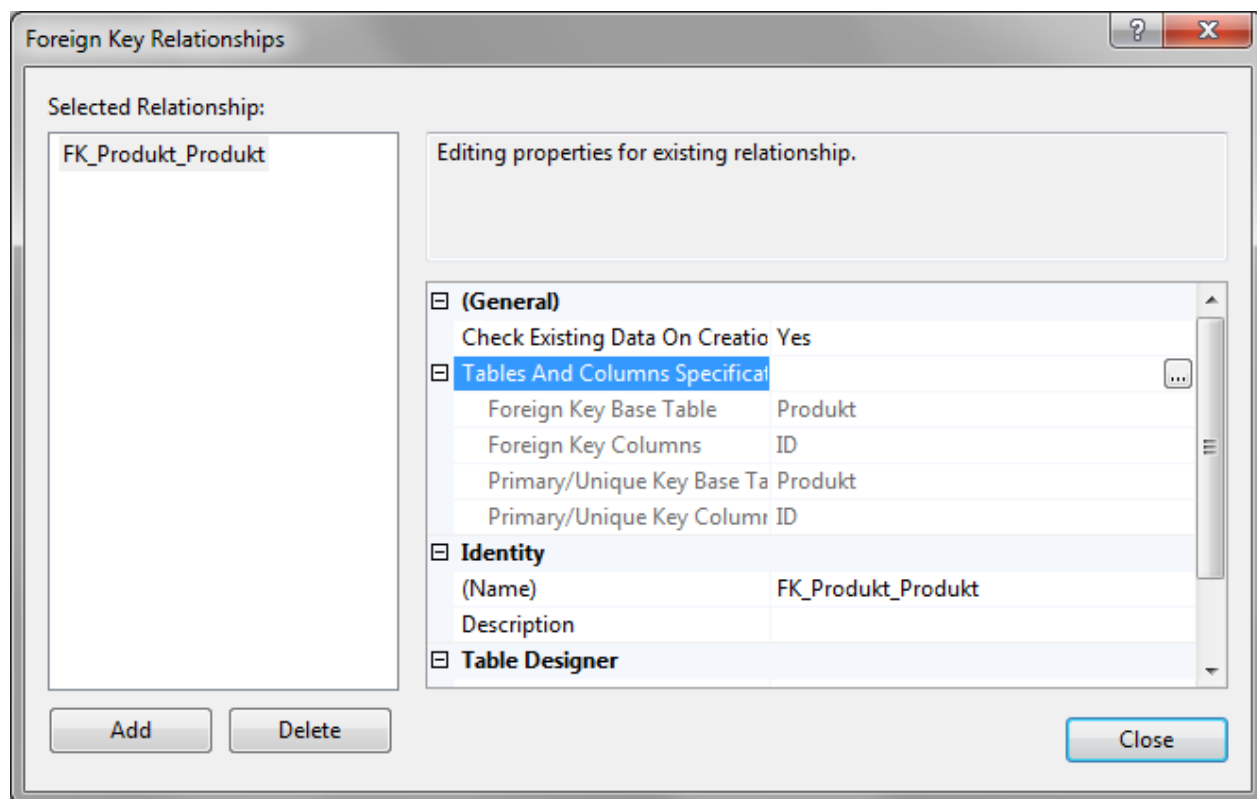
Analogickým způsobem jistě zvládnete vytvořit tabulku Produkt s těmito sloupečky:

- ID, int
- ID_Kategorie, int (odkaz do tabulky kategorií)
- Nazev, nvarchar(100)
- Mnozstvi, nvarchar(100)
- Cena, money

ID je opět primární klíč a žádný sloupeček nepovoluje hodnoty NULL. Kdo dává přednost spuštění skriptu, může využít následující:

```
CREATE TABLE dbo.Produkt
(
    ID int NOT NULL,
    ID_Kategorie int NOT NULL,
    Nazev nvarchar(100) NOT NULL,
    Mnozstvi nvarchar(100) NOT NULL,
    Cena money NOT NULL,
    CONSTRAINT PK_Produkt PRIMARY KEY CLUSTERED (ID)
)
GO
```

Mezi tabulkami je ještě nutné vytvořit relaci, neboť potřebujeme, aby sloupeček ID_Kategorie z tabulky Produkt odkazoval na řádky z tabulky Kategorie identifikované sloupečkem ID. Pokud pro vytvoření tabulky Produkt používáte vizuální designer, přidáte relaci stiskem tlačítka Relationships a použitím tlačítka Add:



Stiskněte tlačítko se 3 tečkami na konci řádku Table and Columns Specifications a vyplňte dialog následovně:

Tables and Columns

Relationship name: FK_Produkt_Kategorie

Primary key table: Kategorie

Foreign key table: Produkt

ID

ID_Kategorie

OK Cancel

Poté již jenom vše potvrďte a na závěr dejte Save pro uložení změn v definici tabulky. Kdo upřednostňuje skript, spustí:

```
ALTER TABLE dbo.Produkt ADD CONSTRAINT FK_Produkt_Kategorie
FOREIGN KEY (ID_Kategorie) REFERENCES dbo.Kategorie(ID)
GO
```

Zbývá ještě naplnit tuto tabulku několika vzorovými řádky dat, což snadno zvládnete sami, anebo použijte následující skript:

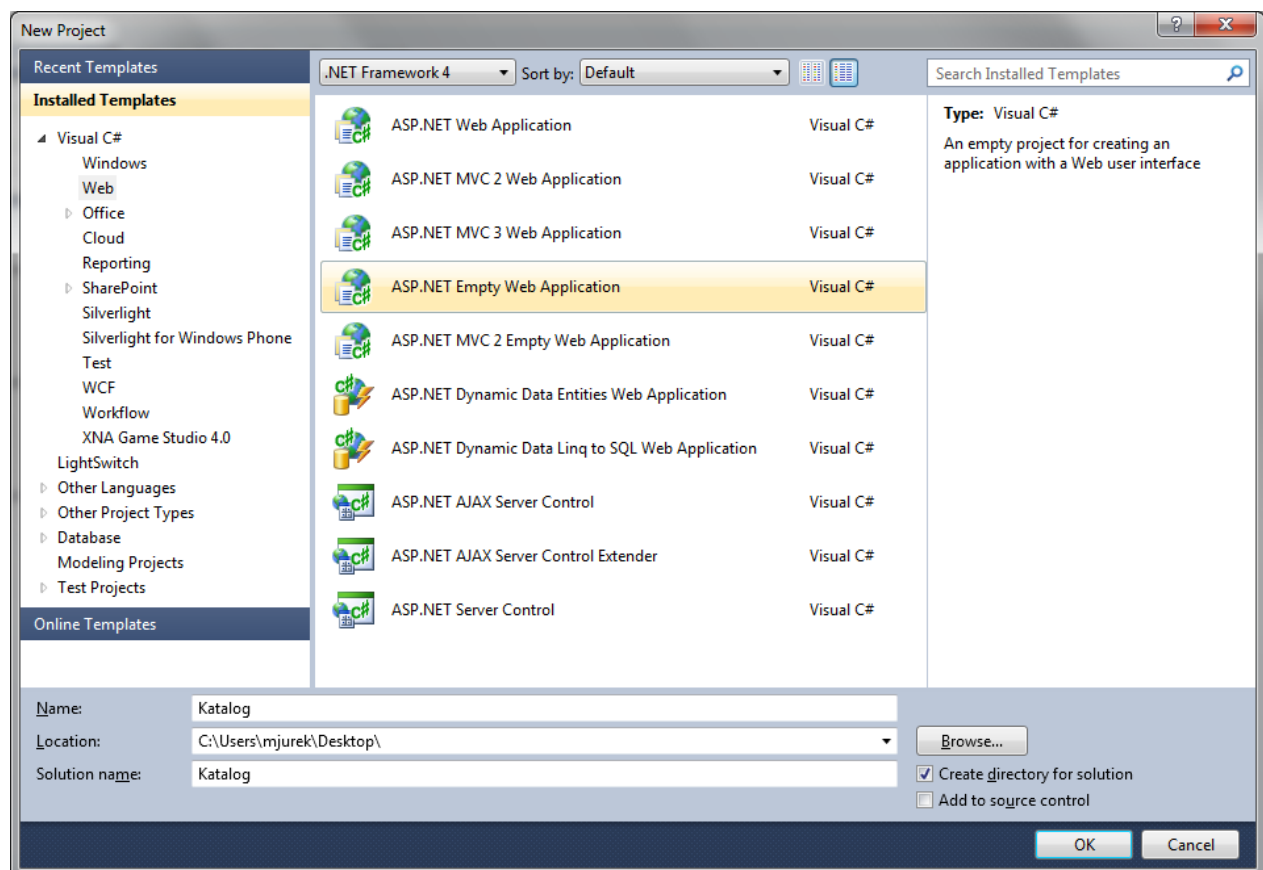
```
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (1, 1,
N'Pivo desítka', N'0,5l', 23)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (2, 1,
N'Pivo dvanáctka', N'0,5l', 29)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (3, 1,
N'Malinovka', N'0,3l', 18.50)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (4, 2,
N'Utopenec', N'2 ks', 29)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (5, 2,
N'Sekaná', N'100g', 23)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (6, 2,
N'Matesy', N'100g', 27)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (7, 3,
N'Chipsy', N'0,5l', 20)
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (8, 3,
N'Slané tyèinky', N'0,5l', 6.50)
```

```
INSERT INTO dbo.Produkt (ID, ID_Kategorie, Nazev, Mnozstvi, Cena) VALUES (9, 3,
N'Preclík', N'1 ks', 8)
GO
```

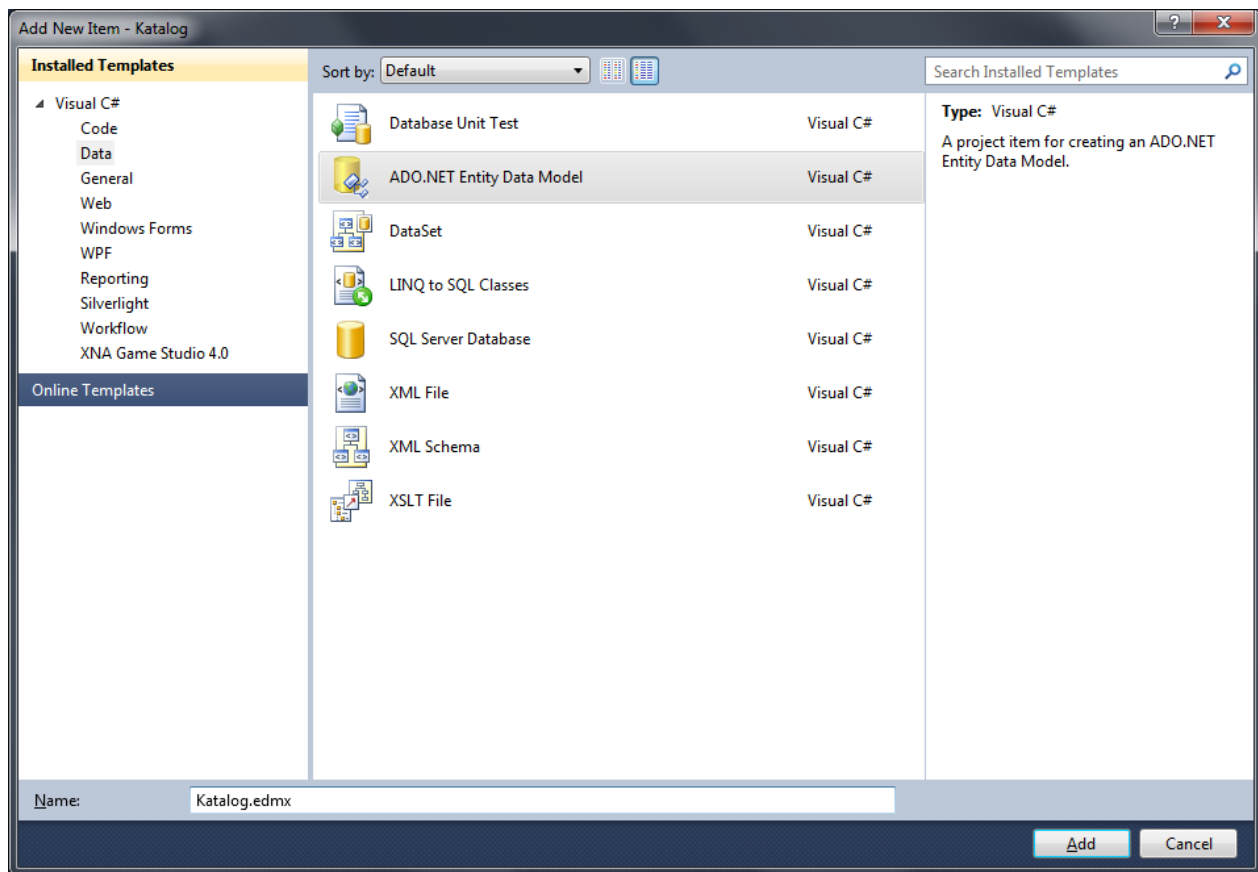
Tím máme databázi hotovou a můžeme přistoupit k vytvoření aplikační vrstvy zpřístupňující data.

3. Vytvoření aplikační vrstvy

Aplikační vrstva bude jednoduchá webová aplikace, která zpřístupňuje data pomocí HTTP/REST rozhraní s podporou protokolu OData pro dotazování. Nejjednodušším způsobem jejího vytvoření je použití Entity Framework pro objektový model dat a použití WCF Data Services pro jeho zpřístupnění přes výše uvedené protokoly. Celý proces je velmi jednoduchý. Otevřte Visual Studio 2010 (případně Web Express) a založte nový projekt typu AP.NET Empty Web Application, aplikaci dejte jméno např. Katalog:



Nyní přidejte do projektu položku typu ADO.NET Entity Data Model s názvem Katalog.edmx:



Na první stránce průvodce zvolte „Generate from database“, na další zvolte New Connection a zadejte připojovací informace k vaší databázi:

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
.\SQLEXPRESS Refresh

Log on to the server

☒ Use Windows Authentication

☐ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name:
Katalog

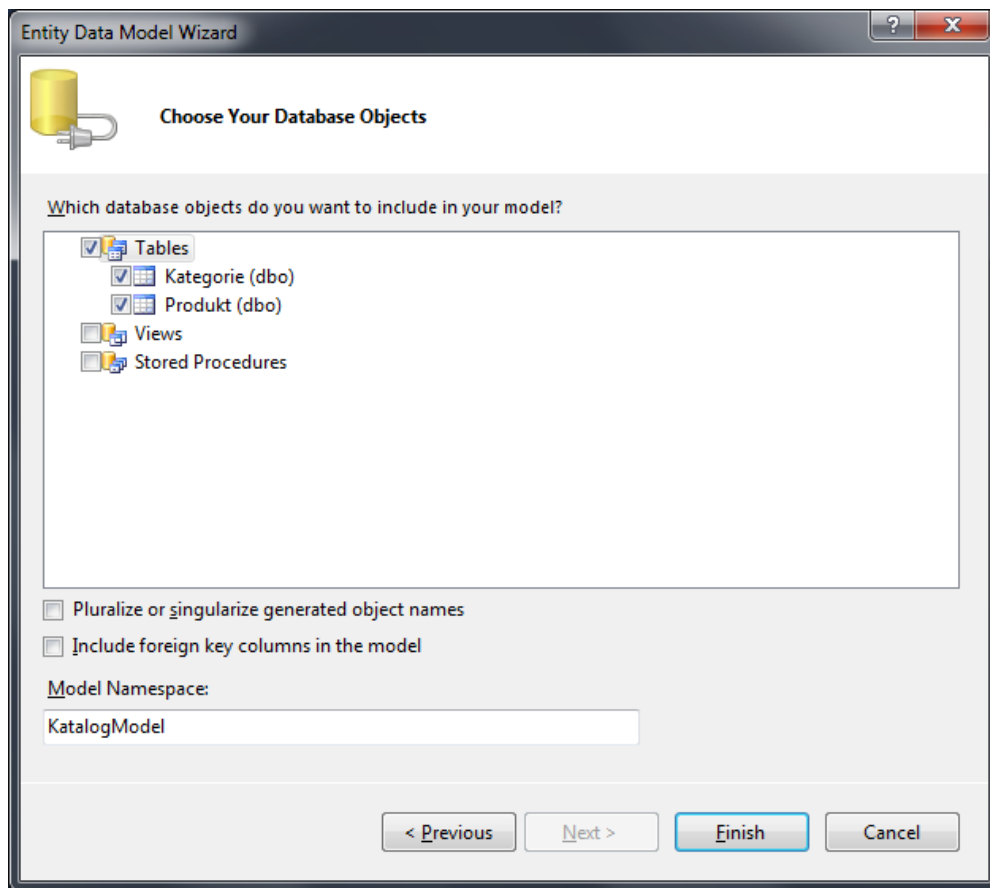
☐ Attach a database file:
 Browse...

Logical name:

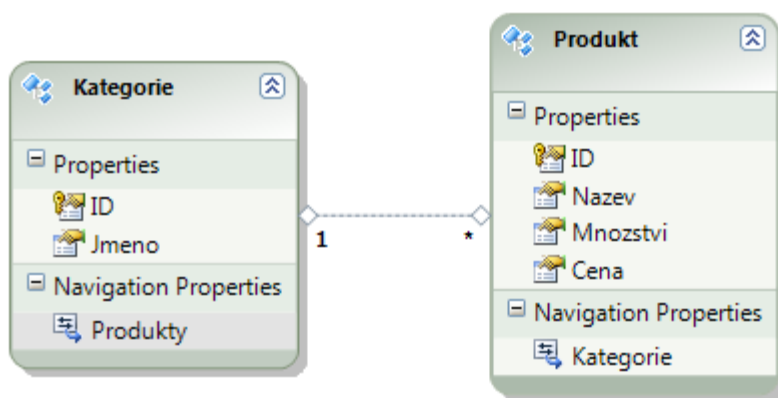
Advanced...

Test Connection OK Cancel

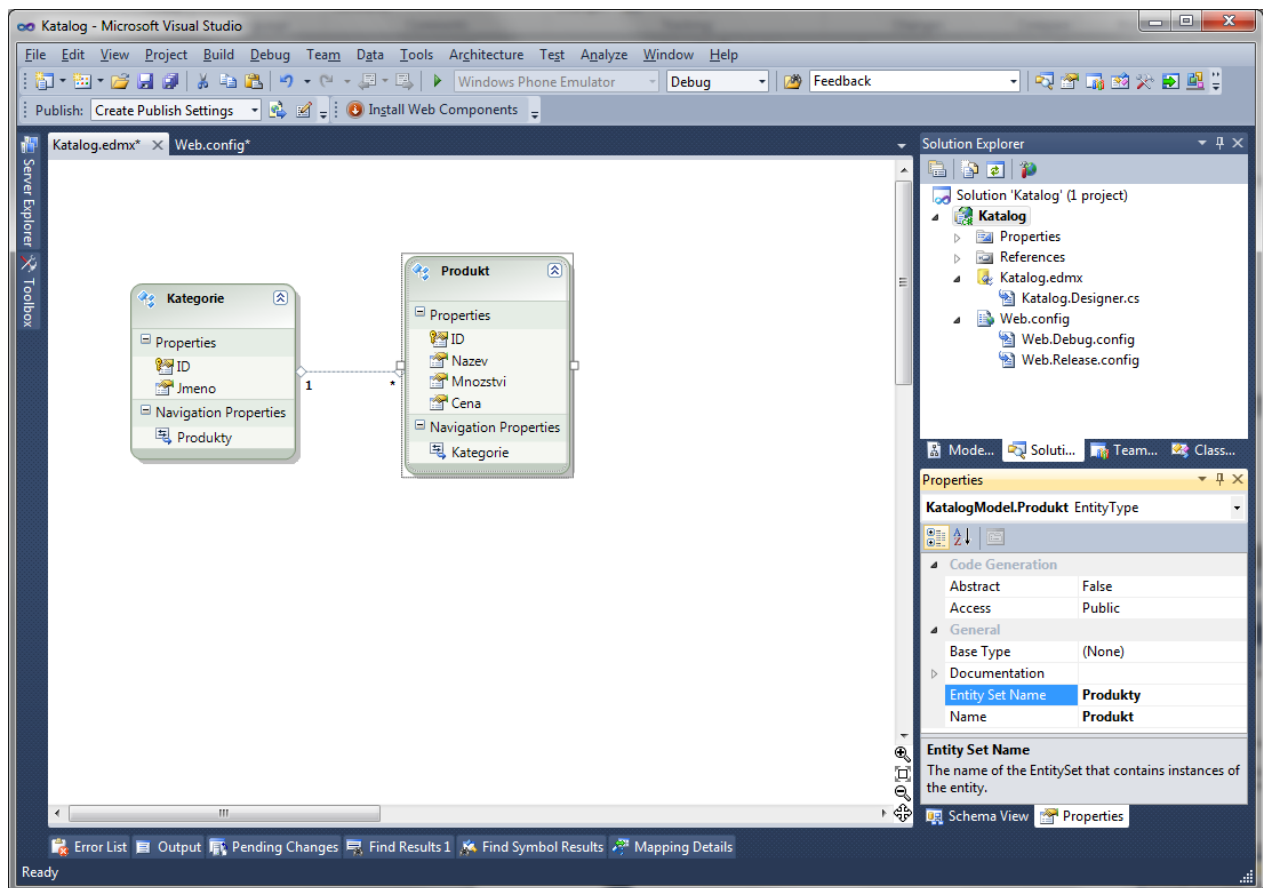
Případně vyzkoušejte správnost zadaných údajů pomocí volby Test Connection. Po návratu do průvodce akceptujte výchozí hodnoty, na další stránce nastavte generování objektového modelu následovně:



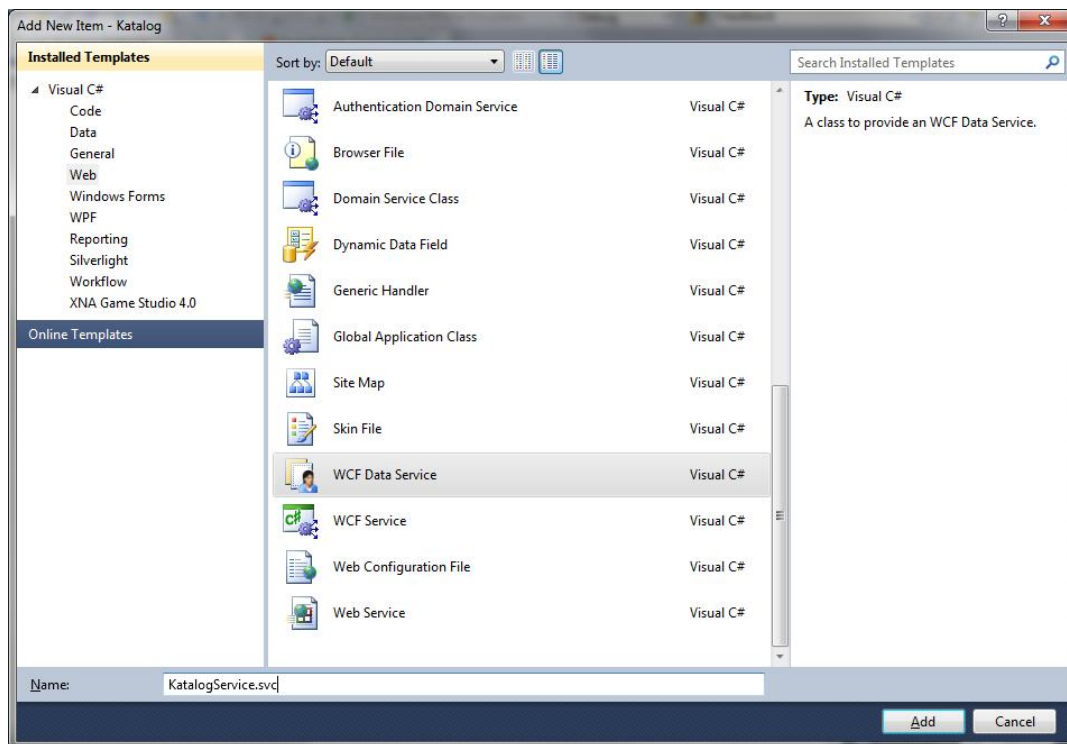
Otevře se vám grafický návrh objektového modelu entit, v něm můžete upravit vlastnost Kategorie jménem Produkt na Produkty (množné číslo):



A dále ve vlastnostech entity Produkt nastavte Entity Set Name rovněž na množné číslo Produkty:



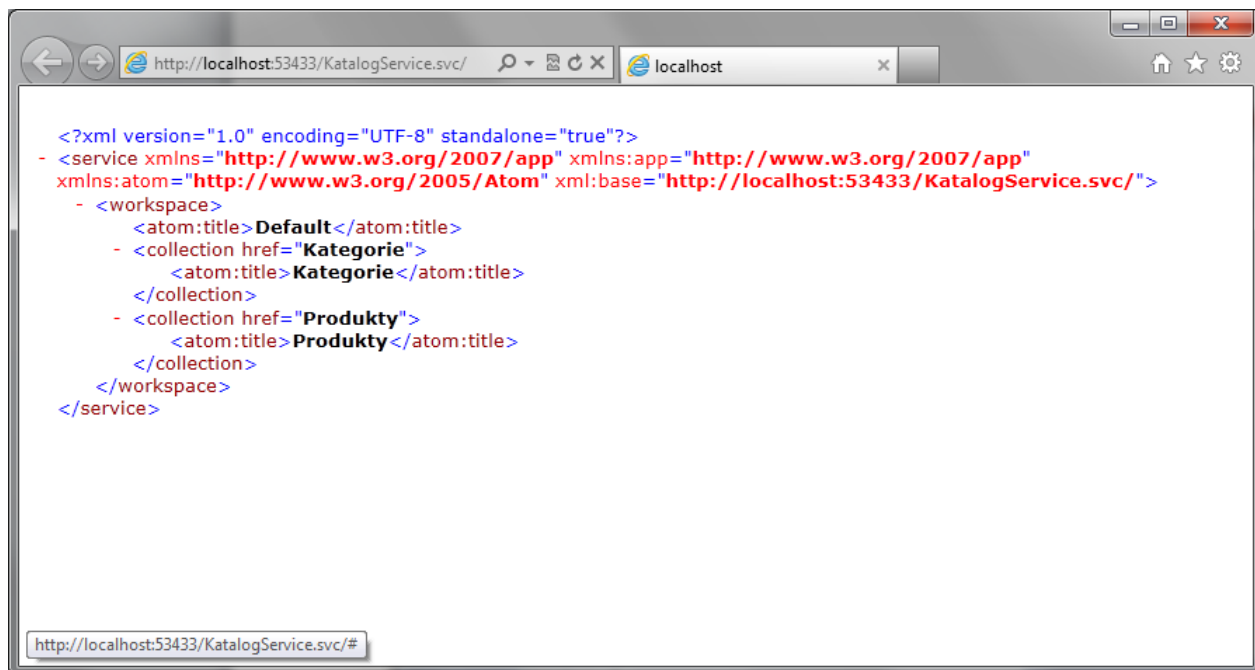
Uložte všechny změny a zkuste zkompilevat projekt (Build/Build Solution). Nyní přidáme třídu zpřístupňující tento model jako REST službu. Přidejte do projektu novou položku typu WCF Data Service pojmenovanou KatalogService.svc :



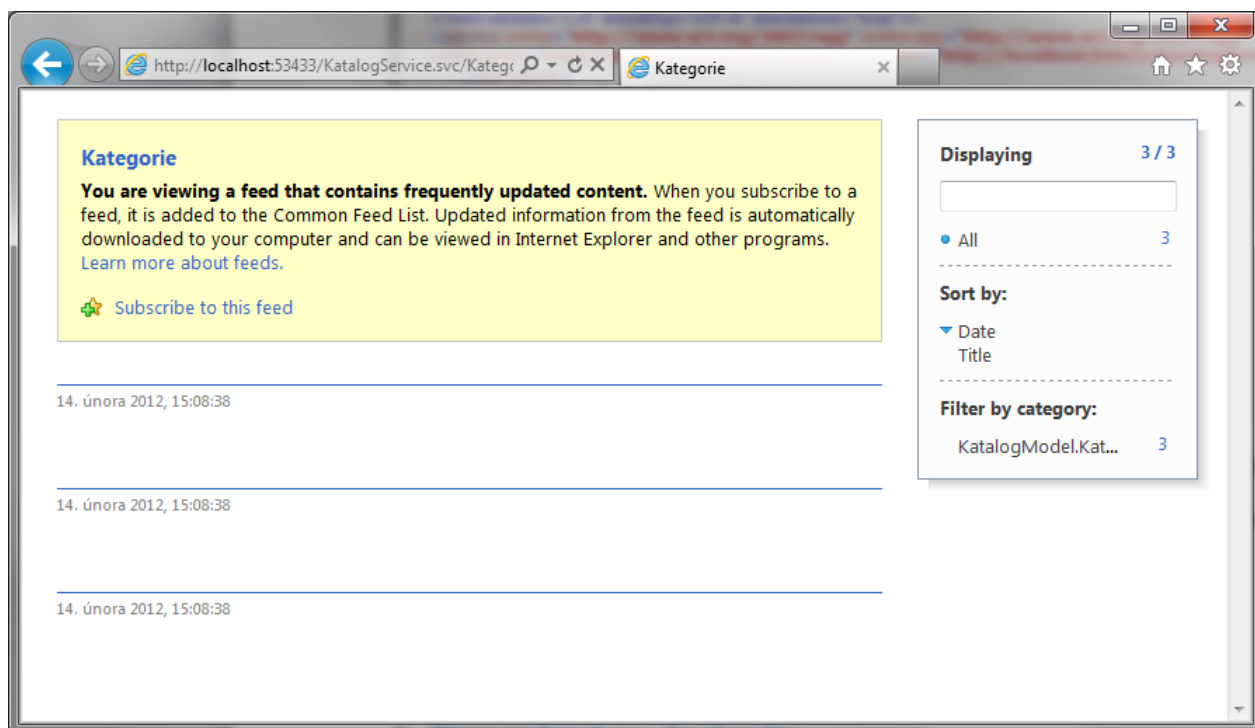
Ve vygenerovaném kódu je třeba doplnit zpřístupňovaný objektový kontext (KatalogEntities) a dále dát práva ke čtení veškerých dat:

```
public class KatalogService : DataService<KatalogEntities>
{
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
        config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
    }
}
```

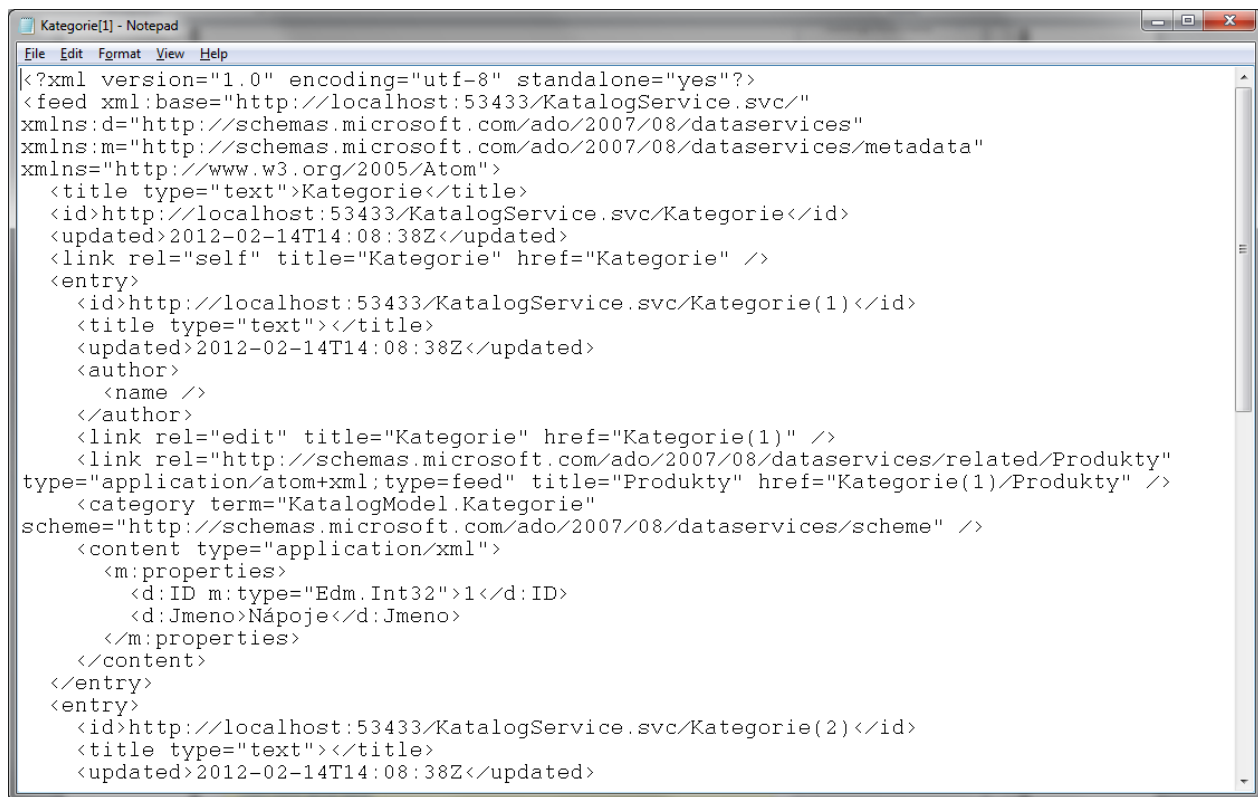
Nyní můžeme službu vyzkoušet, stačí použít Debug/Start Debugging a měl by se vám zobrazit prohlížeč s popisem objektového modelu (vaše číslo portu může být jiné):



Pokud URL doplníte na konci o /Kategorie, měl by se vám objevit feed se 3 položkami pro kategorie, něco podobného jako:

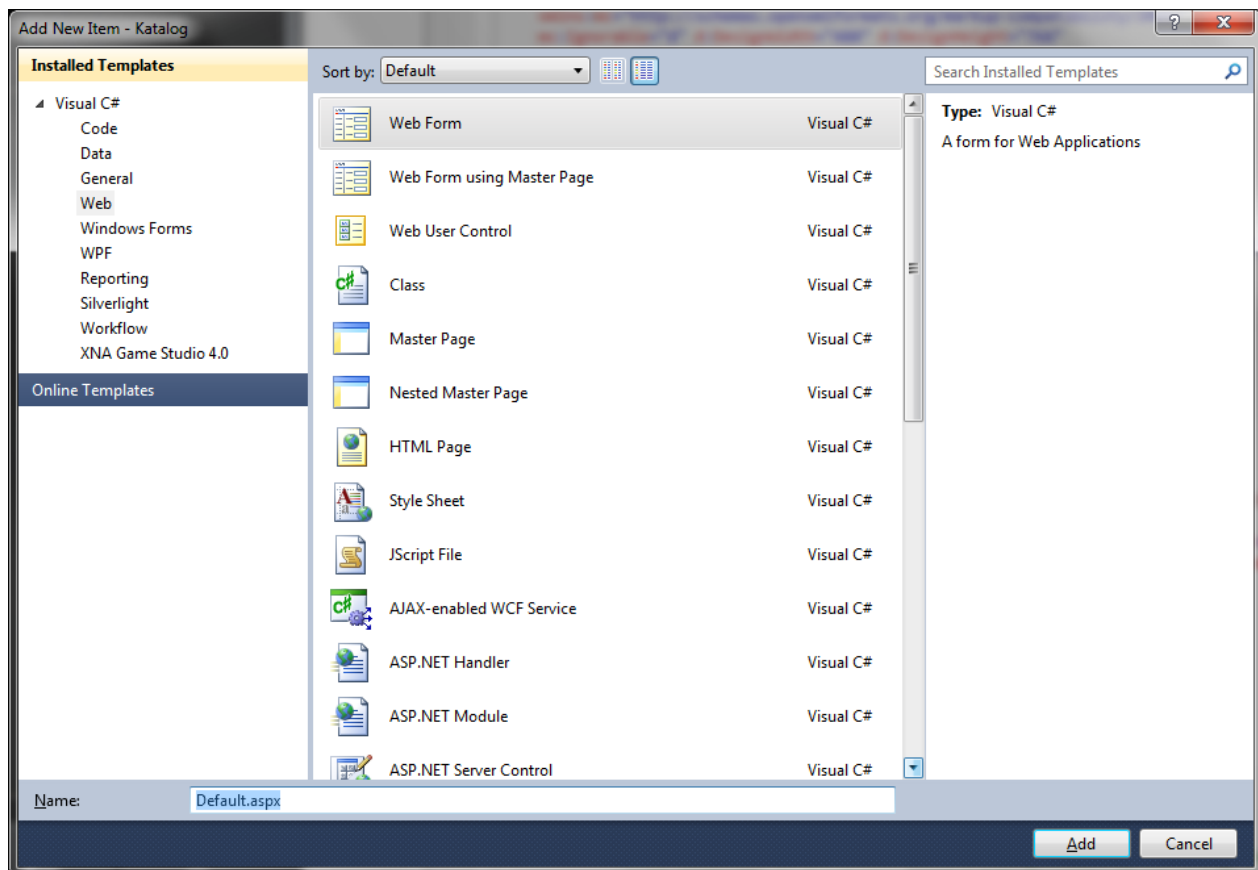


Případně si můžete zobrazit zdroj stránky:

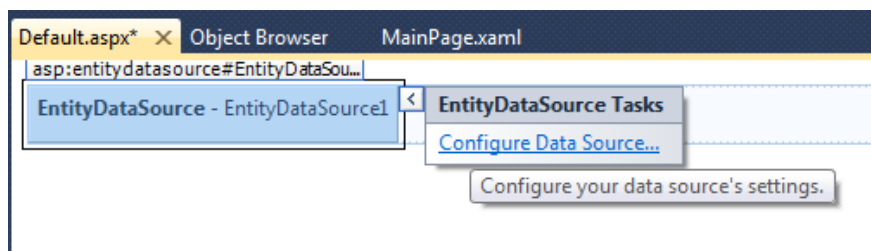


```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="http://localhost:53433/KatalogService.svc/"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Kategorie</title>
  <id>http://localhost:53433/KatalogService.svc/Kategorie</id>
  <updated>2012-02-14T14:08:38Z</updated>
  <link rel="self" title="Kategorie" href="Kategorie" />
  <entry>
    <id>http://localhost:53433/KatalogService.svc/Kategorie(1)</id>
    <title type="text"></title>
    <updated>2012-02-14T14:08:38Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Kategorie" href="Kategorie(1)" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Produkty"
type="application/atom+xml;type=feed" title="Produkty" href="Kategorie(1)/Produkty" />
    <category term="KatalogModel.Kategorie"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:ID m:type="Edm.Int32">1</d:ID>
        <d:Jmeno>Nápoje</d:Jmeno>
      </m:properties>
    </content>
  </entry>
  <entry>
    <id>http://localhost:53433/KatalogService.svc/Kategorie(2)</id>
    <title type="text"></title>
    <updated>2012-02-14T14:08:38Z</updated>
```

Nyní ještě můžeme vytvořit webovou stránku pro zobrazení dat. Přidáme do projektu novou položku typu Web Form pojmenovanou Default.aspx:




Přepněte do Design zobrazení a přetáhněte z toolboxu ovládací prvek EntityDataSource, poté klikněte na Configure Data Source:



V seznamu připojení byste měli mít k dispozici KatalogEntities:

Configure Data Source - EntityDataSource1

 **ConfigureObjectContext**

ConnectionString:

☒ Named Connection

KatalogEntities

☐ Connection String


DefaultContainerName:

KatalogEntities

< Previous Next > Finish Cancel

Na další stránce pak vyberte EntitySetName Kategorie:

Configure Data Source - EntityDataSource1

 **Configure Data Selection**

EntitySetName:

Kategorie

EntityTypeFilter:

(None)

Select:

☒ Select All (Entity Value)

☐ ID

☐ Jmeno

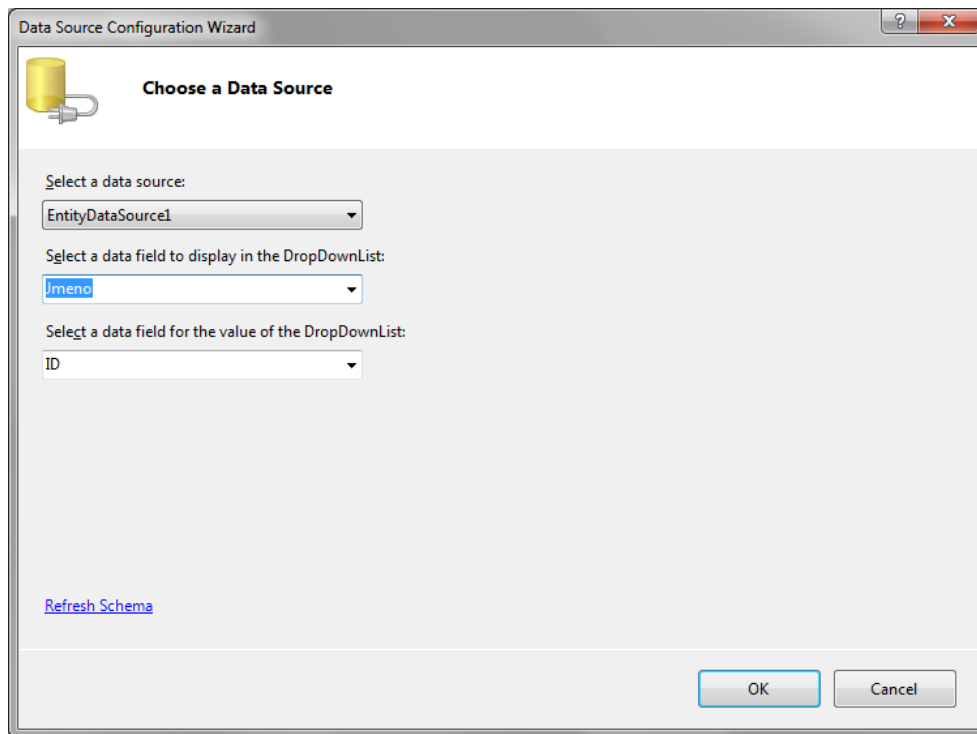
☐ Enable automatic inserts

☐ Enable automatic updates

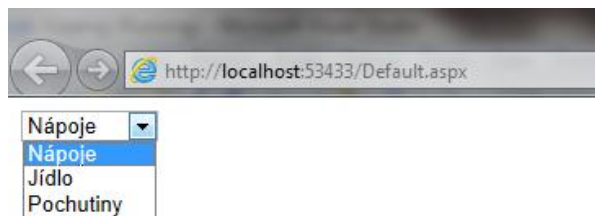
☐ Enable automatic deletes

< Previous Next > Finish Cancel

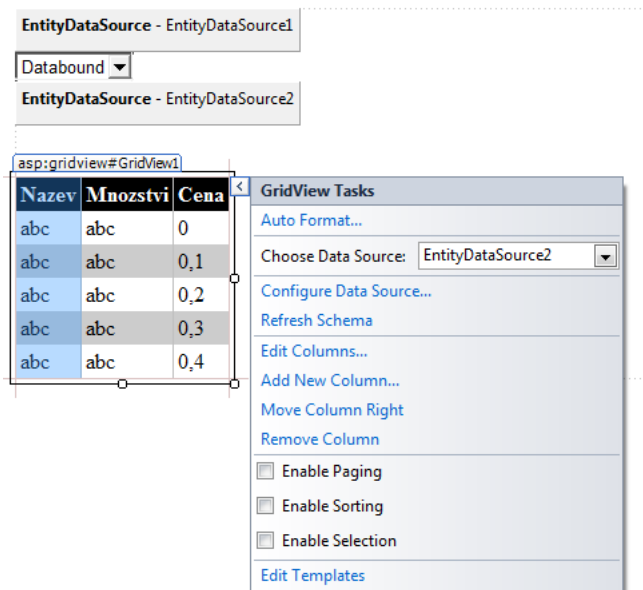
Dále z toolboxu vložte na stránku DropDownList a zvolte Choose Data Source z nabízených možností. Dialog, který se zobrazí, vyplňte následovně:



Pro kontrolu si můžeme stránku zobrazit, rozbalovací seznam by měl být vyplněn správnými kategoriemi:



Dále přidáme na stránku ještě jeden EntityDataSource, tentokrát pro zobrazení produktů. Postupujte stejně jako pro první EntityDataSource, pouze jako jméno EntitySetName vyberte Produkty. Poté vložte na webový formulář prvek GridView, který můžete využitím funkce AutoFormat trochu polidštit, a poté vyberte v Choose Data Source možnost EntityDataSource2. Pokud chcete, můžete též odstranit nepotřebné sloupce ID a Kategorie.ID jejich výběrem a funkcí Remove Column. Výledek vaší práce by měl vypadat zhruba takto:



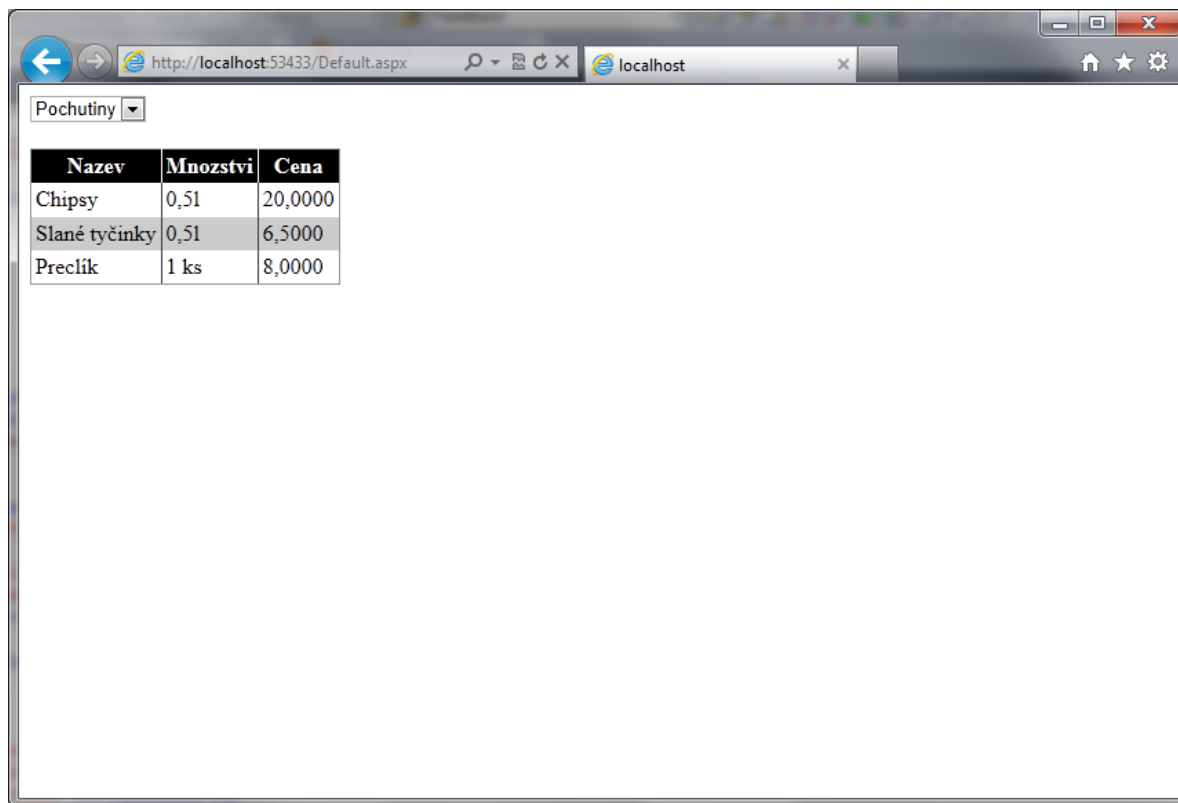
Posledním krokem je propojení zobrazovaných produktů se zvolenou kategorií – zatím se zobrazují vždy všechny produkty. Za tím účelem je nutné nastavit u prvku DropDownList1 vlastnost AutoPostBack na True, aby se při změně vybrané kategorie stránka znovu načetla:

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
    DataSourceID="EntityDataSource1" DataTextField="Jmeno" DataValueField="ID">
</asp:DropDownList>
```

a u zdroje dat EntityDataSource2 nastavit podmínku Where parametrem převzatým z hodnoty ovládacího prvku DropDownList1:

```
<asp:EntityDataSource ID="EntityDataSource2" runat="server"
    ConnectionString="name=KatalogEntities" DefaultContainerName="KatalogEntities"
    EntitySetName="Produkty" Where="it.Kategorie.ID=@IdKategorie">
    <WhereParameters>
        <asp:ControlParameter Name="IdKategorie" Type="Int32"
ControlID="DropDownList1"/>
    </WhereParameters>
</asp:EntityDataSource>
```

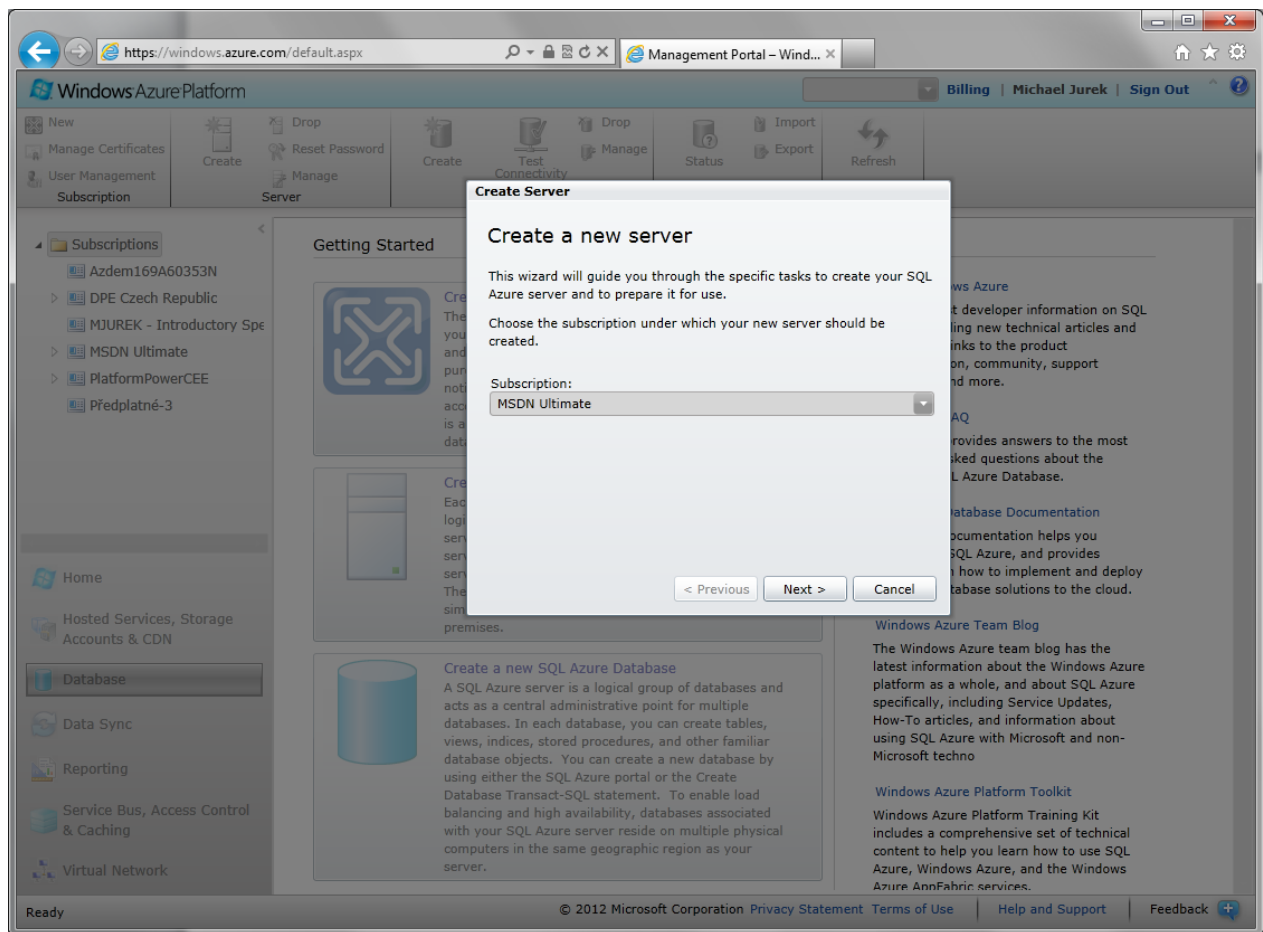
Po spuštění bychom měli mít hotovou funkční stránku:



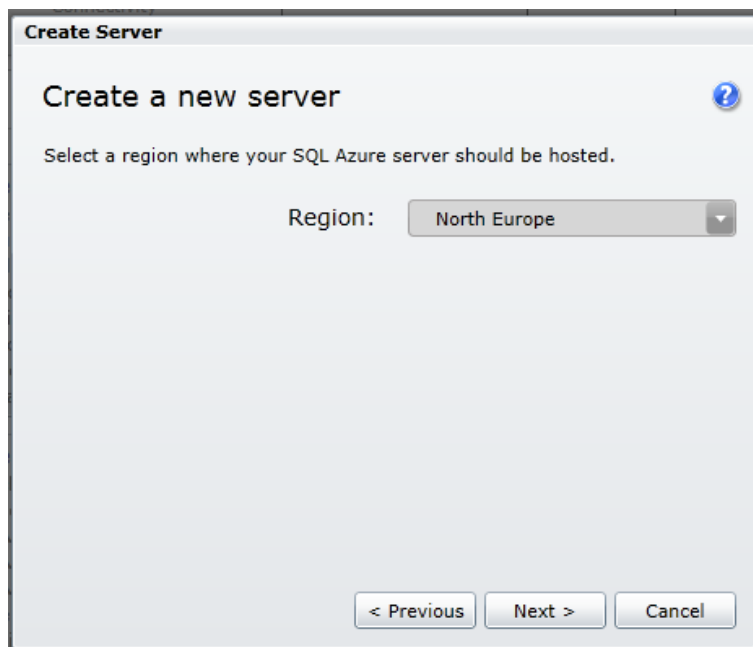
Aplikační vrstva nám tedy úspěšně běží na lokálním počítači a nic již nebrání pokusit se ji přenést do cloudu. Začneme opět odspodu, tedy databází.

4. Přenos databáze do cloudu

Předpokládám, že máte zařízený účet na Windows Azure tak jak je popsáno v první kapitole. Jděte na stránku <http://windows.azure.com> a přihlaste se svým LiveID. Přejděte na stránku Database a zvolte Create a new SQL Azure Server:



Zvolte svoji Azure subskripci, na další stránce vyberte některé z evropských datových center:



Na další stránce zadejte jméno a heslo administrátora:

Create Server

Create a new server

Specify the login and password of the server-level principal of your SQL Azure server.

Administrator Login: michael

Password:

Confirm password:

< Previous Next > Cancel

Na poslední stránce nastavte firewall tak, aby byl SQL Server dostupný ze všech IP adres. To rozhodně není doporučený bezpečnostní postup, ale pro účely zkoušení je to tolerovatelné:

Create Server

Create a new server

Specify one or more firewall rules that enable access to your SQL Azure server. If you do not, you will not be able to connect to or manage the databases on this server.

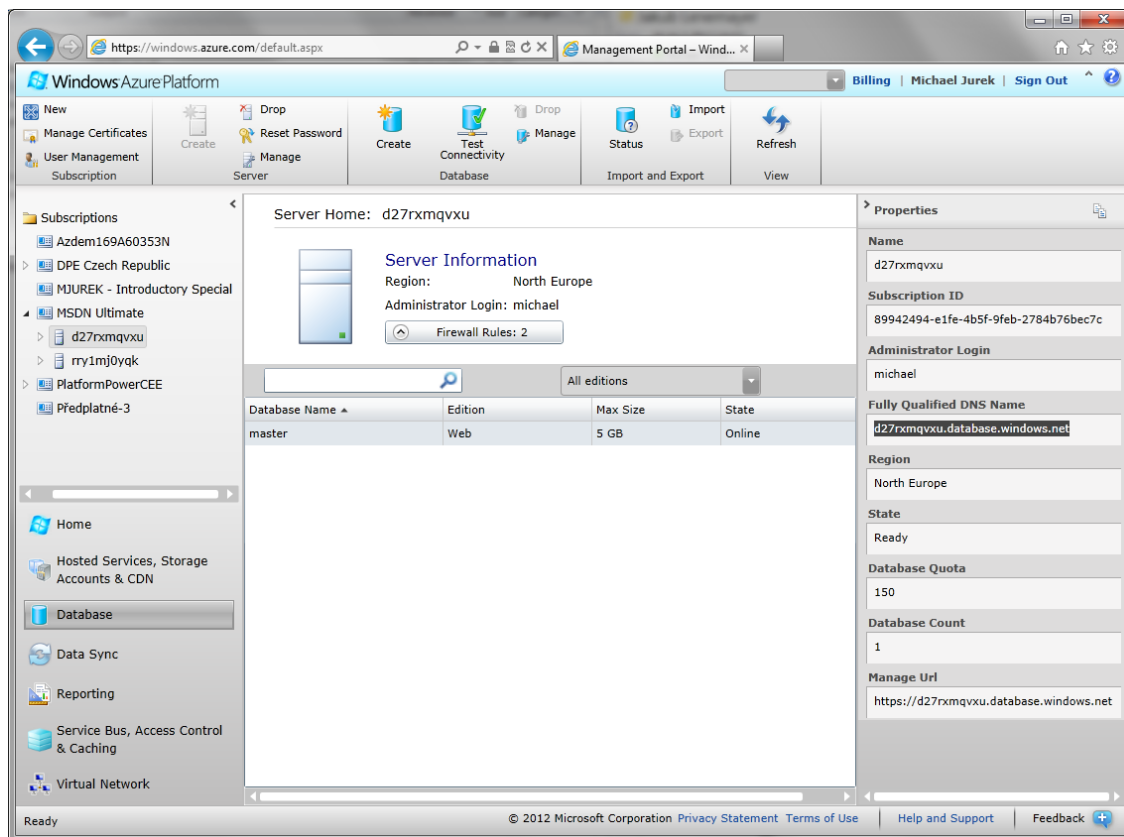
Rule Name ▲	IP Range Start	IP Range End
MicrosoftServices	0.0.0.0	0.0.0.0
Vsechno	0.0.0.0	255.255.255.255

Add Update Delete

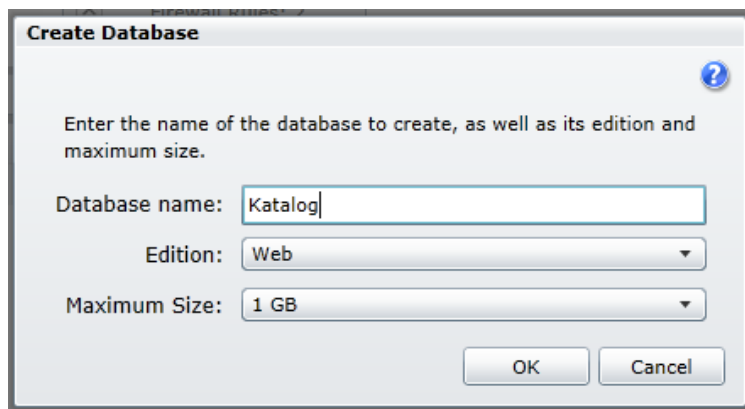
☒ Allow other Windows Azure services to access this server

< Previous Finish Cancel

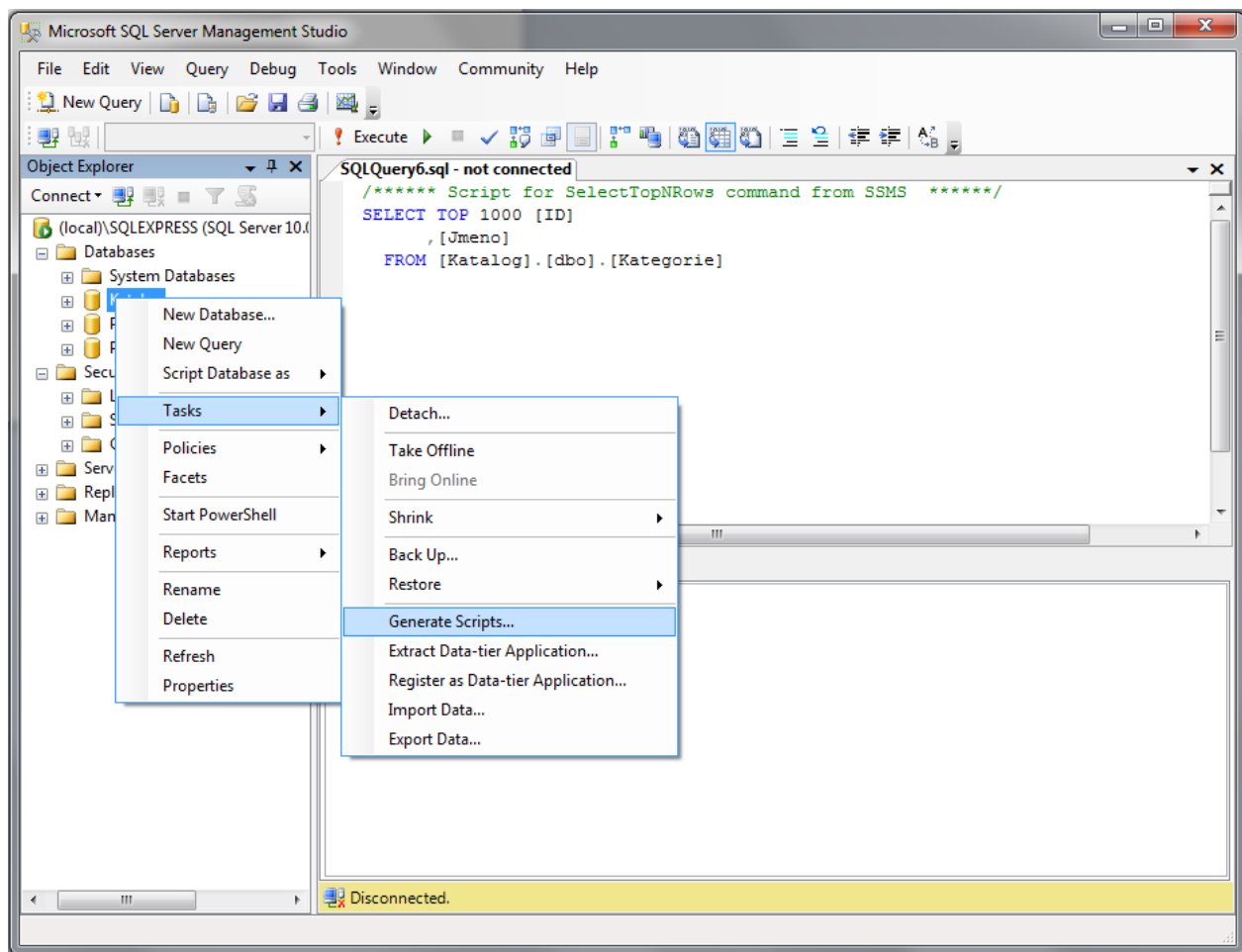
Po potvrzení se vrátíte na hlavní stránku portálu, odkud si nezapomeňte poznamenat DNS jméno nově vytvořeného SQL Serveru:



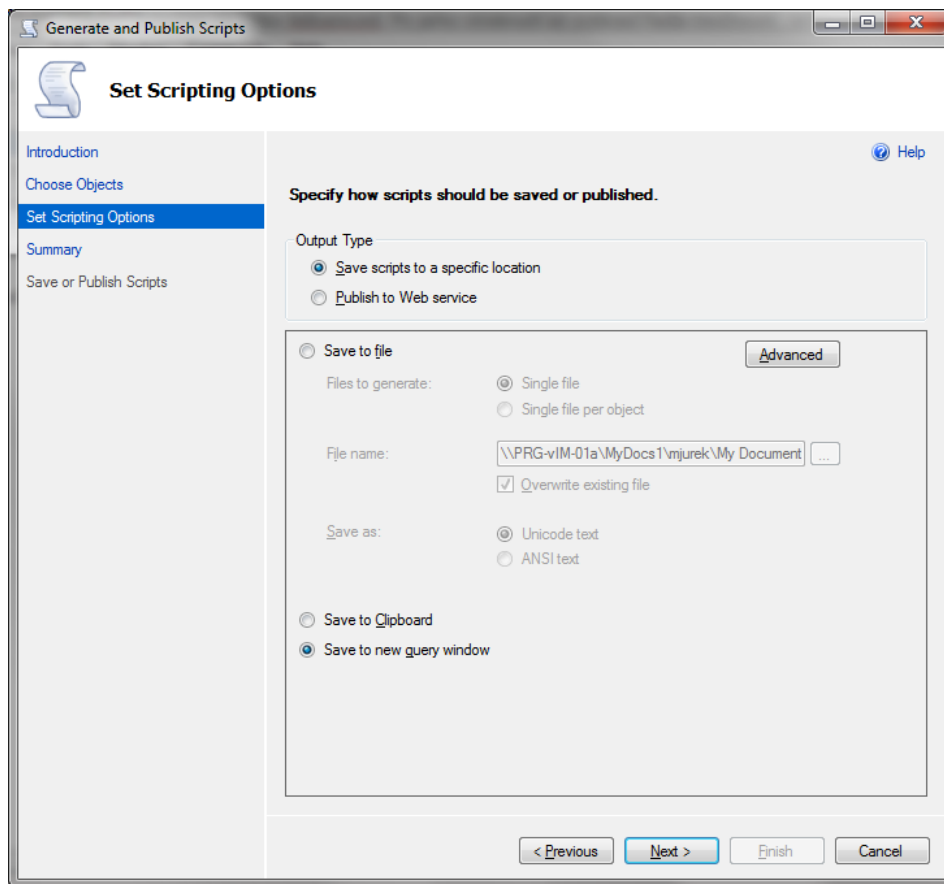
Nyní založíme novou databázi Katalog pomocí Database/Create, plně bude postačovat nejmenší velikost 1 GB:



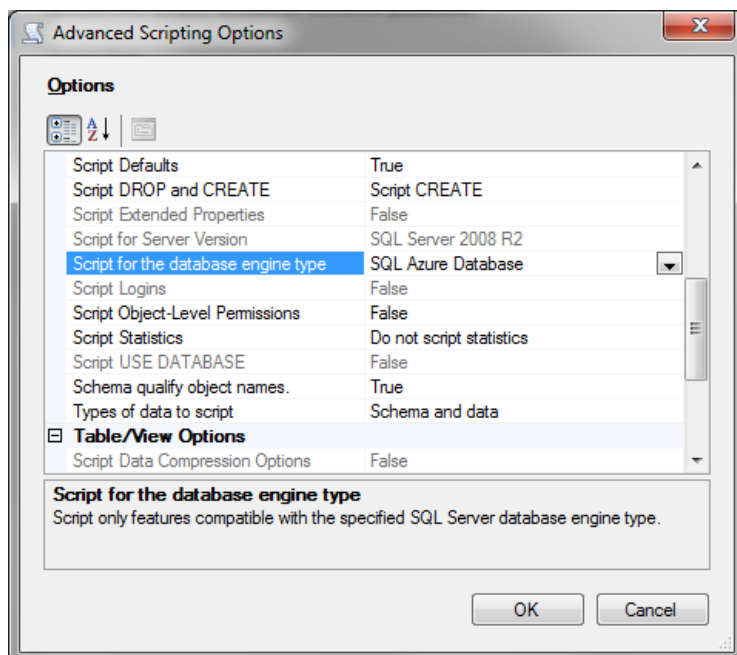
Dalším úkolem je přenos dat z klasické databáze do prostředí SQL Azure. Vzhledem k malé velikosti dat je nejjednodušším způsobem vygenerování skriptů a jejich spuštění v cílové databázi. Pomocí SQL Server Management Studia je to jednoduché:



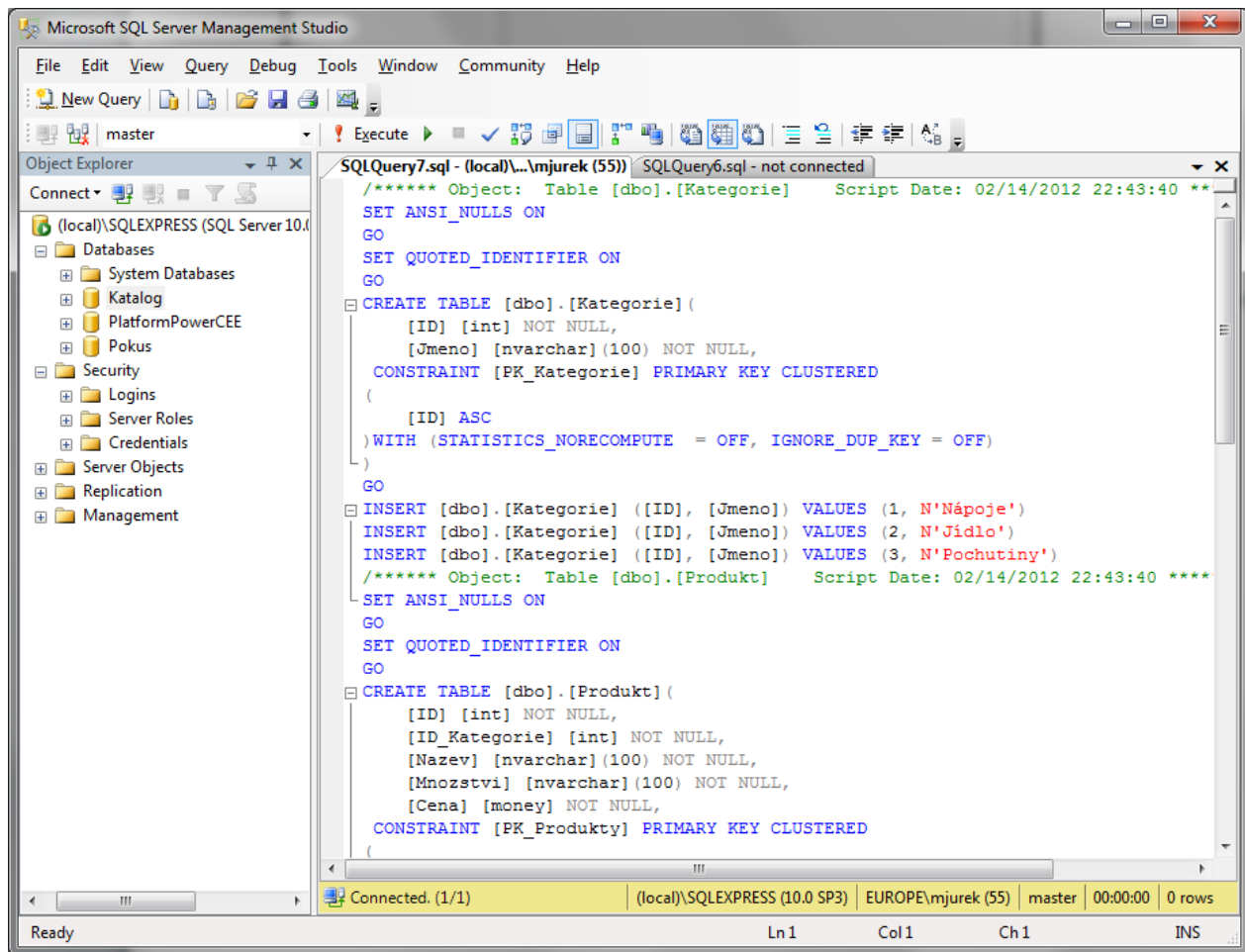
První dvě stránky průvodce můžete rychle proběhnout, zajímavější je až třetí strana:



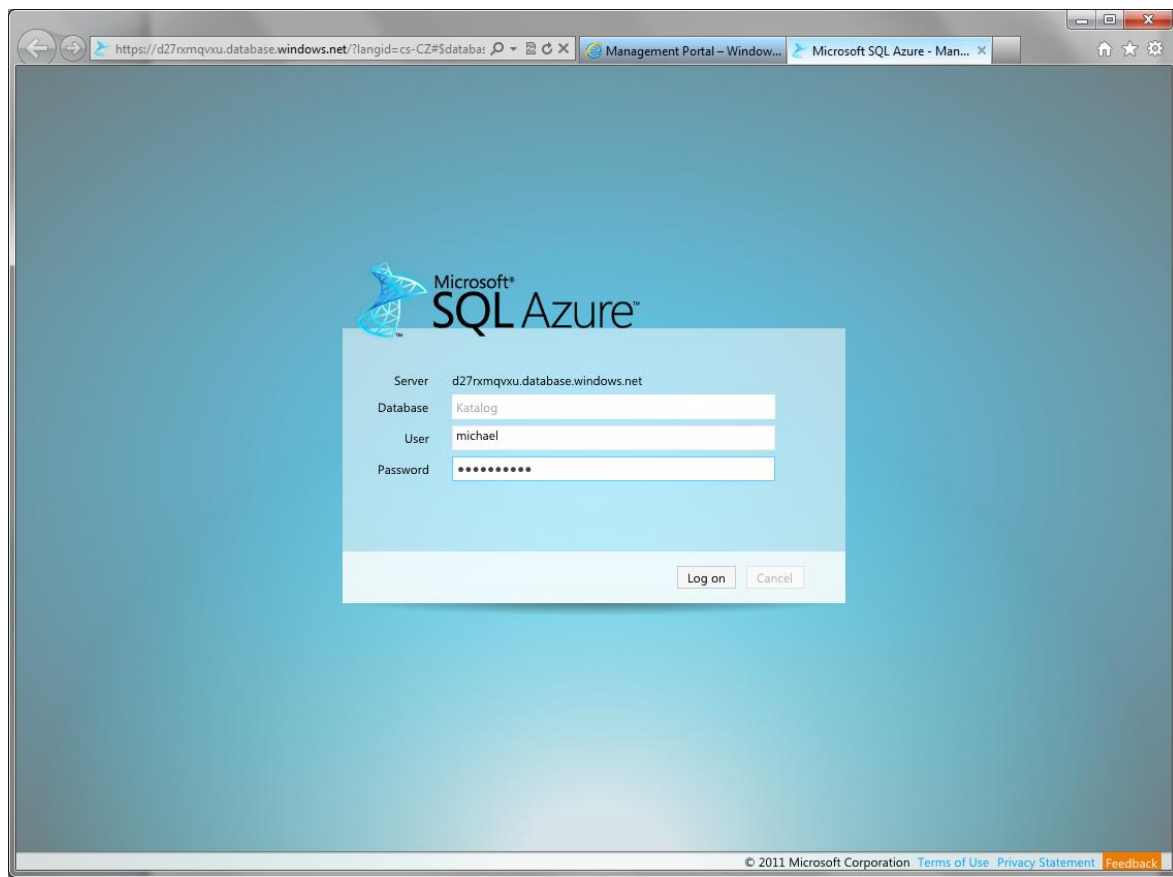
Kromě toho, že si můžete vybrat, kam bude skript vygenerován (osobně volím nové okno v SSMS), je zde též velmi důležité tlačítko **Advanced**. Po jeho stisknutí se zobrazí řada možností, ze kterých je nutno změnit dvě: Script database engine type = SQL Azure Database a dále Script Type = Schema and data:



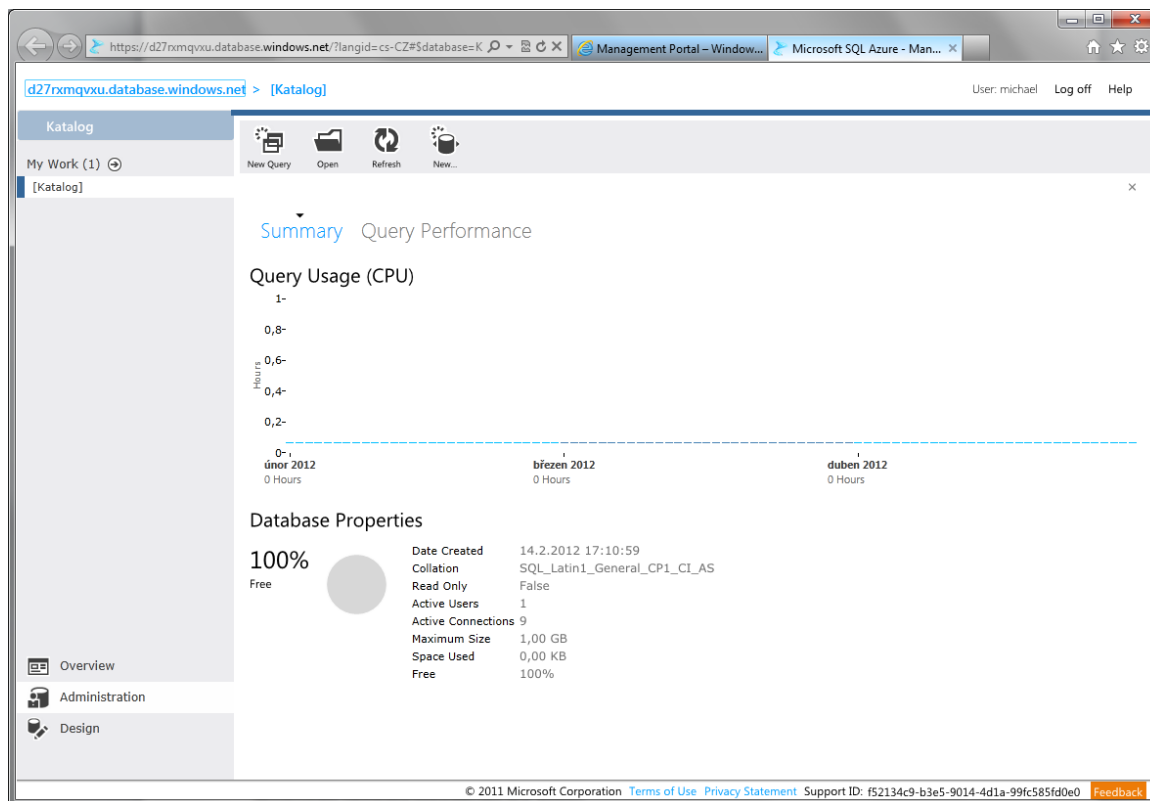
Po dokončení průvodce se zobrazí vygenerovaný skript:



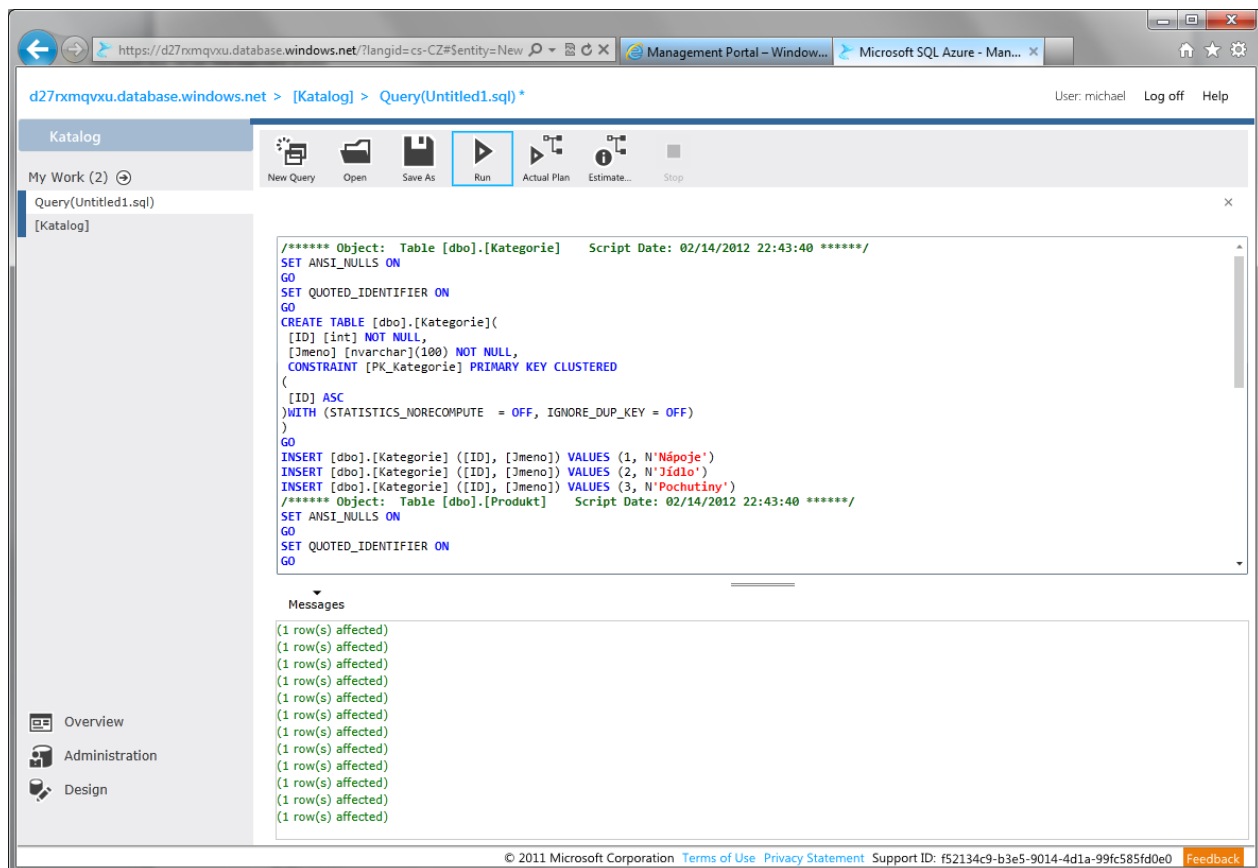
Nyní jsou dvě možnosti jak tento skript spustit proti SQL Azure databázi. První je připojení k SQL Azure serveru z Management Studia (stejně jako k lokálnímu SQL Serveru) a spuštění příslušného skriptu. Druhou (z pedagogického hlediska zajímavější) možností je využít portálu pro správu Windows Azure. V sekci Database zvolte v panelu nástrojů nahoře Database/Manage:



V zobrazeném dialogu zadejte jméno a heslo administrátora (pokud byste měli problémy s přihlášením, zkontrolujte nastavení firewallu pro SQL Azure). Po přihlášení by měl prohlížeč vypadat zhruba takto:



Zvolte možnost New Query, nakopírujte do okna SQL skript vygenerovaný Management Studiem a stiskněte Run, výsledek by měl vypadat zhruba následovně:



Nyní je databáze včetně dat korektně vytvořena v cloudu, posledním krokem bude změnit připojovací řetězec v naší aplikaci tak, aby aplikace využívala SQL Azure databázi. Ve Visual Studiu otevřete soubor web.config a změňte ho následovně:

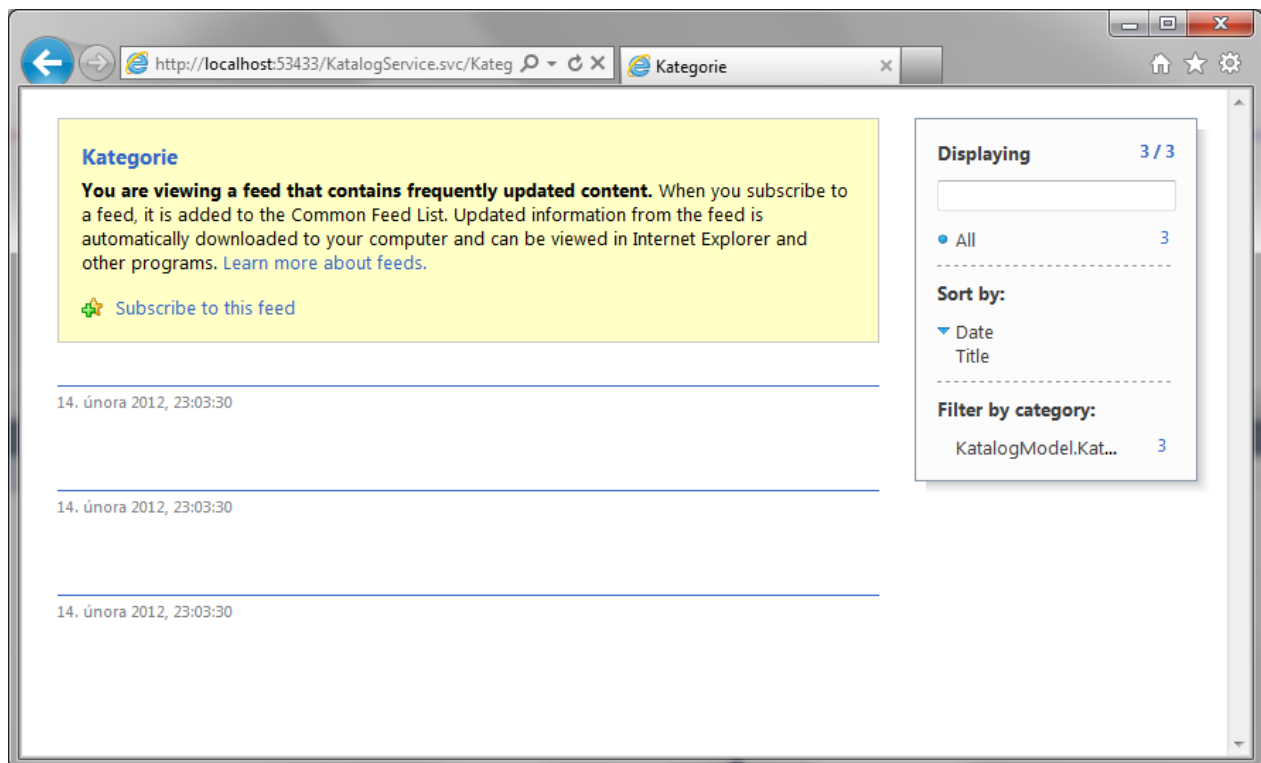
- Opravť jméno serveru (data source)
- Přidejte své uživatelské jméno a heslo (User ID, Password)
- Nastavte Integrated Security na false

```

...
<connectionStrings>
  <add name="KatalogEntities"
    connectionString="metadata=res://*/Katalog.csdl|res://*/Katalog.ssdl|res://*/Katalog.msl;provider=System.Data.SqlClient;provider connection string="data source=d27rxmqvxu.database.windows.net;initial catalog=Katalog;integrated security=False;User ID=michael; Password=tajne.heslo.324;multipleactiveresultsets=True;App=EntityFramework";providerName="System.Data.EntityClient" />
</connectionStrings>
...

```

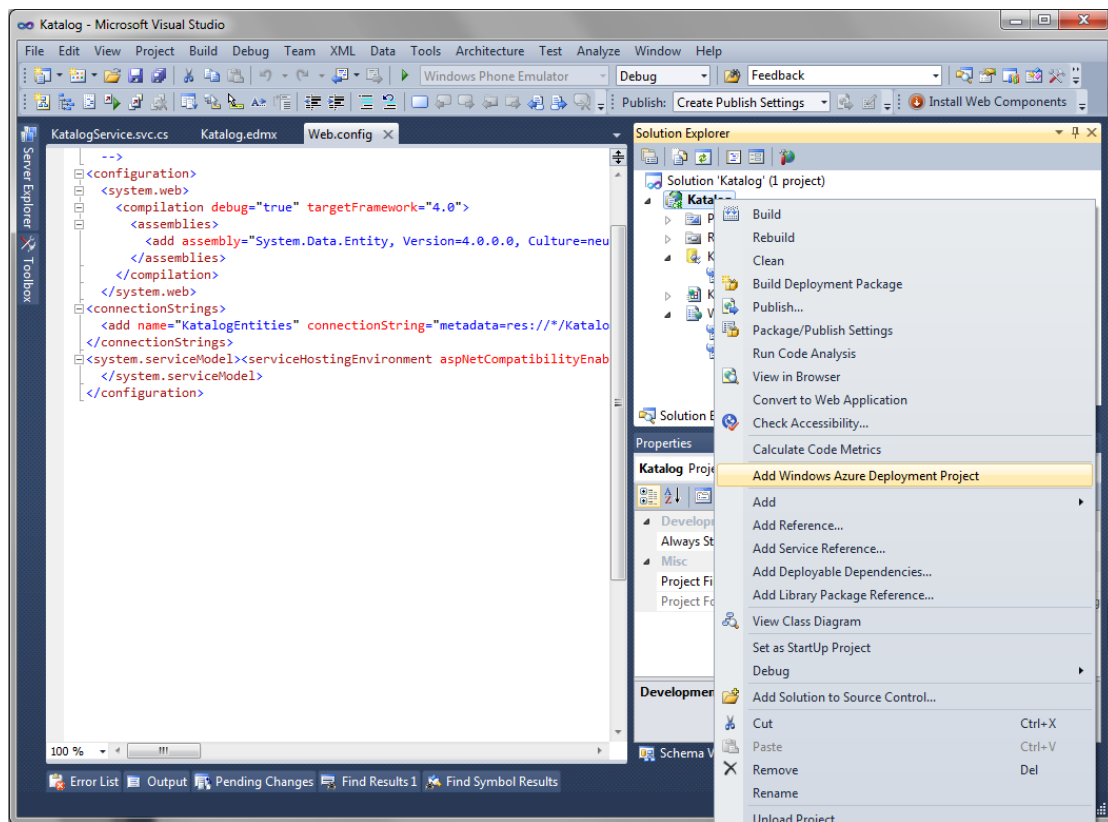
Vyžlucené údaje samozřejmě nahradte vlastními hodnotami. Po této změně by měla aplikace nadále fungovat, tj. například feed pro kategorie by měl stále ukázat 3 položky (vyzkoušejte Debug/Start Debugging, číslo portu může být ve vašem případě jiné):



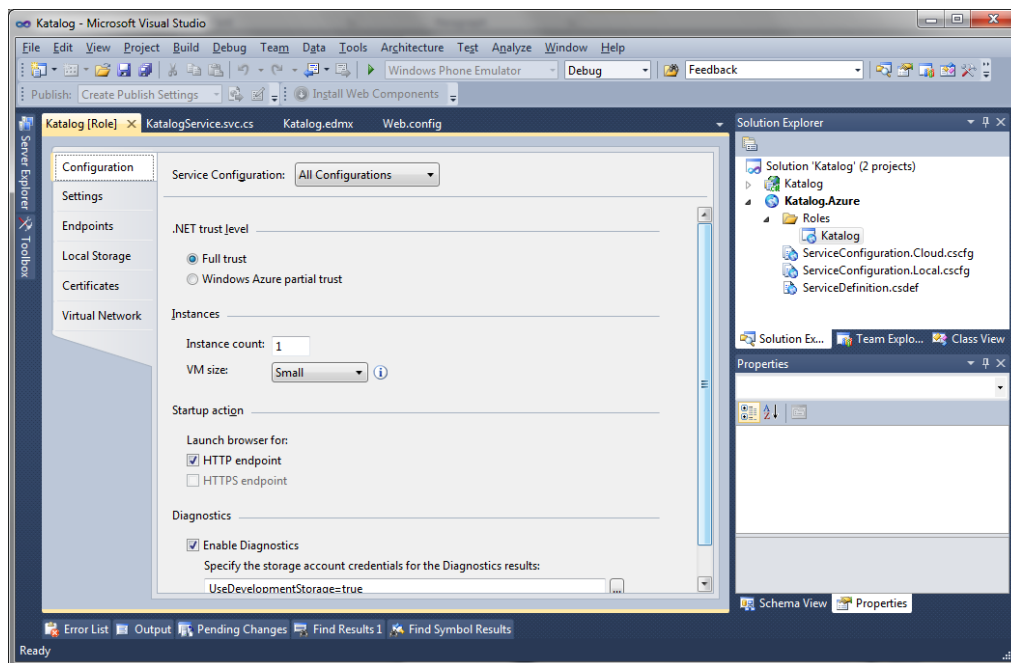
Databáze je tedy přenesena do cloudu, dalším krokem bude přenos aplikační vrstvy.

5. Přenos aplikační vrstvy do cloudu

Naše aplikace využívá pouze databázi a nemá žádné další závislosti na operačním systému, proto bude její převod pro běh v cloudovém operačním systému velmi hladký. Stačí pouze přidat speciální projekt, který aplikaci zabalí pro nasazení v cloudu, podobně jako se například vytváří projekty pro zabalení desktopových aplikací do instalačních balíčků. Klikněte pravým tlačítkem na svém webovém projektu a zvolte možnost Add Windows Azure Deployment Project:



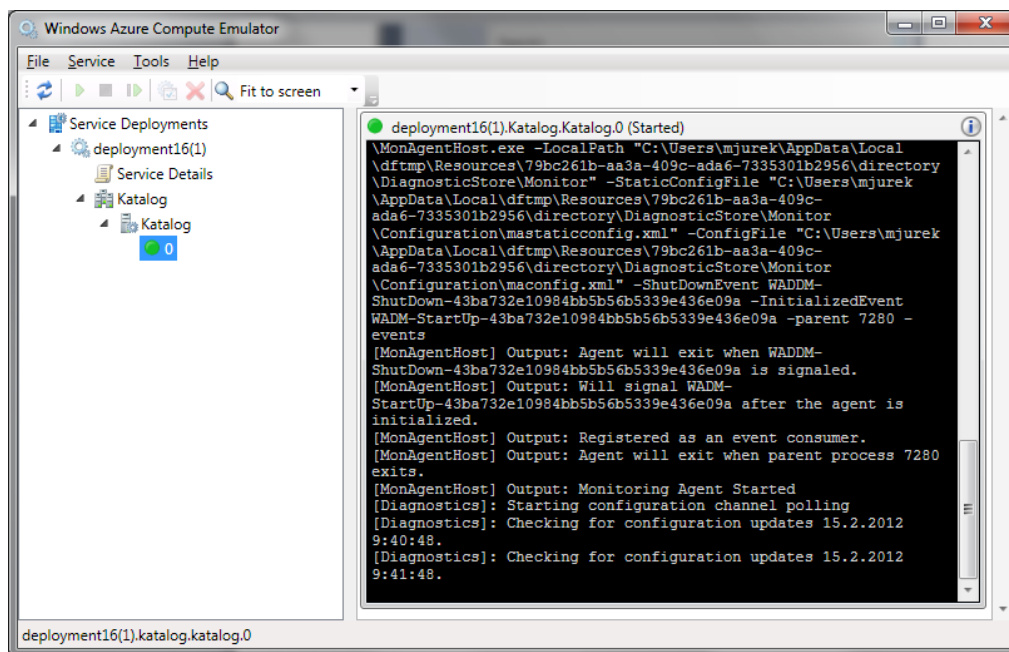
Jak se můžete přesvědčit, byla vytvořena nová definice služby pro budoucí běh ve Windows Azure. Obsahuje jednu webovou roli Katalog, která poběží v jedné instanci virtuálního počítače velikosti Small:



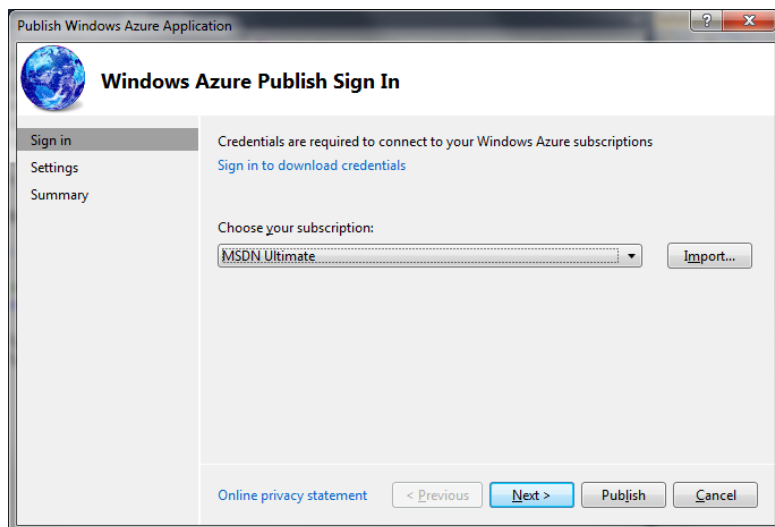
Aplikaci nyní můžete spustit v emulátoru Windows Azure prostředí prostým použitím Debug/Start Debugging, rovněž ji v emulátoru můžete běžným způsobem ladit. Poběží podobně jako předtím, jenom URL se změní, pravděpodobně na <http://localhost:81/KatalogService.svc>



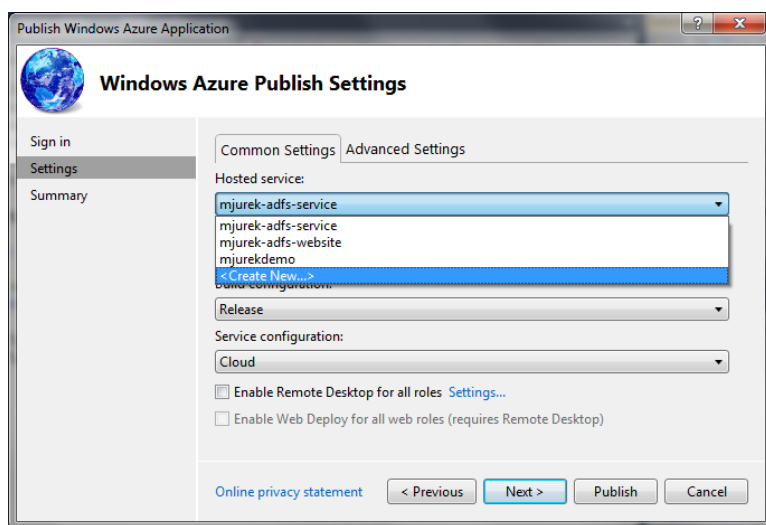
Dole na systémové liště se objeví ikona emulátoru (okno v azurové barvě), na které si můžete kontextovým menu zvolit Show Compute Emulator UI a přesvědčit se, že emulátor běží jak má a vaše webová aplikace běží v jedné instanci:



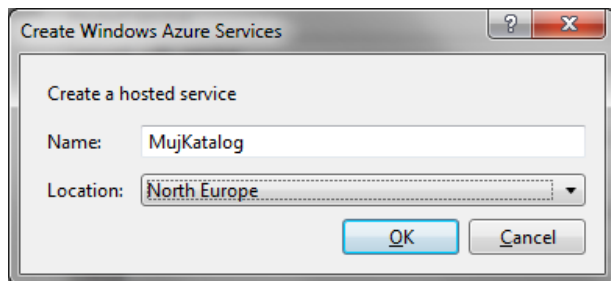
Zajímavější ovšem bude nasazení do opravdového cloudu Windows Azure. Klikněte pravým tlačítkem na projektu Katalog.Azure a zvolte možnost Publish:



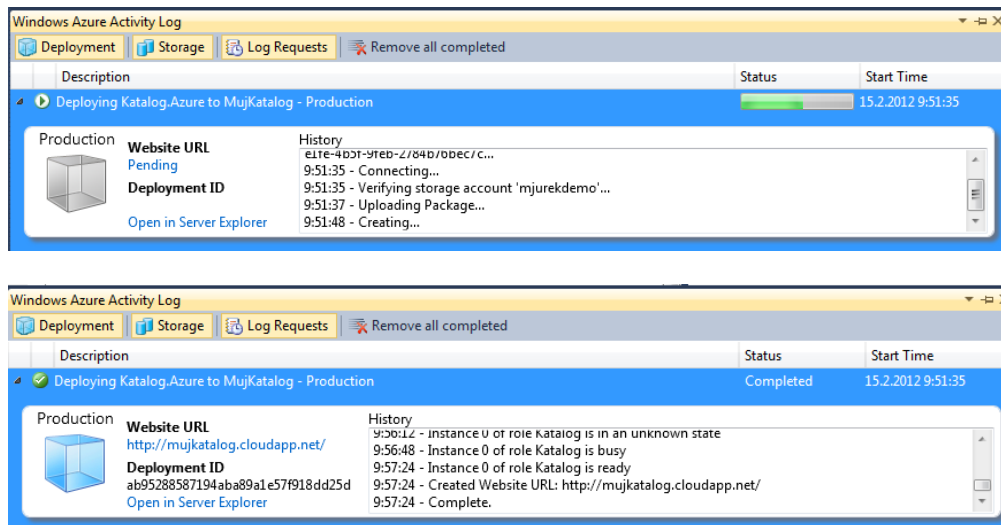
Možnost volby Azure subskripce máte zřejmě prázdnou, proto zvolte link nahoře Sign in to download credentials. Přihlaste se svým LiveID, které jste použili pro založení Azure účtu. Stáhněte si nabízený soubor *.publishsettings na bezpečné místo a poté stiskněte tlačítko Import... pro načtení informací z tohoto souboru. Přejděte na další stránku a zvolte v řádku Hosted Service možnost Create New:



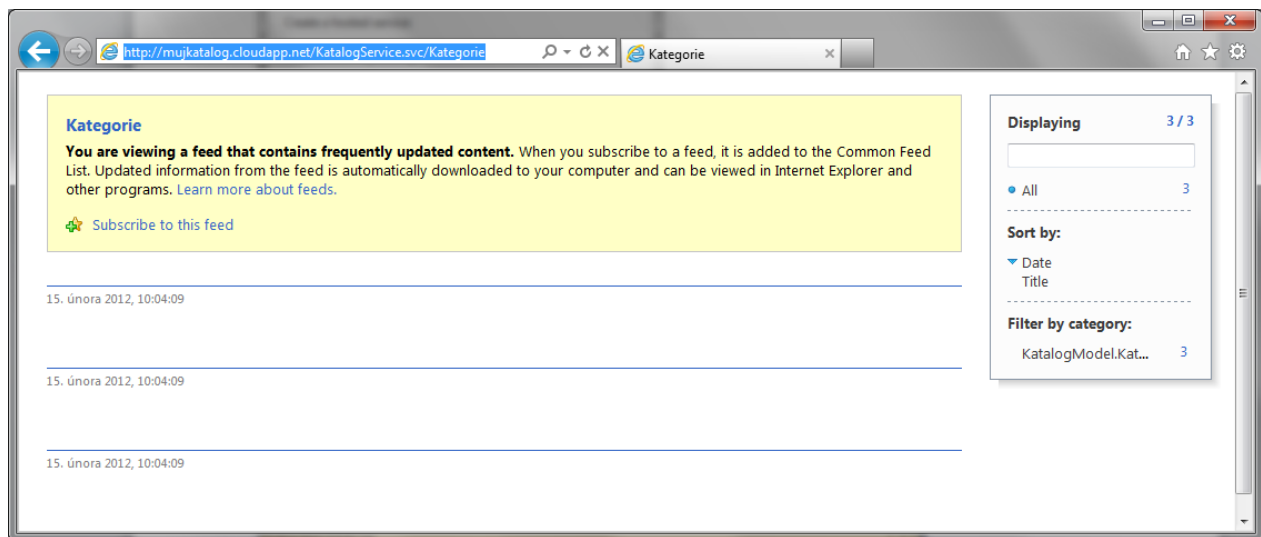
Zde musíte zadat první část DNS jména (bude pak to-co-zadate.cloudapp.net) a datové centrum – zvolte stejné datové centrum, ve kterém jste založili databázový server SQL Azure!



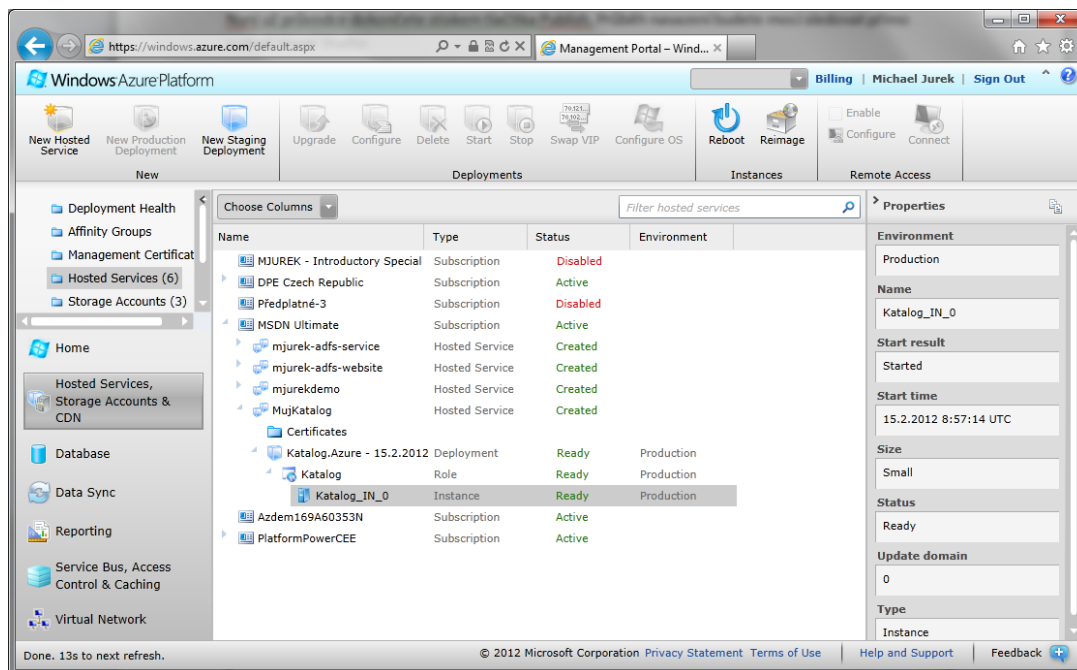
Nyní už průvodce dokončete stiskem tlačítka Publish. Průběh nasazení budete moci sledovat přímo v okně Visual Studio:



Po dokončení můžete zkontrolovat, zda služba korektně funguje, tj. např. na adrese <http://mujkatalog.cloudapp.net/KatalogService.svc/Kategorie> by se měly zobrazit 3 položky pro kategorie:



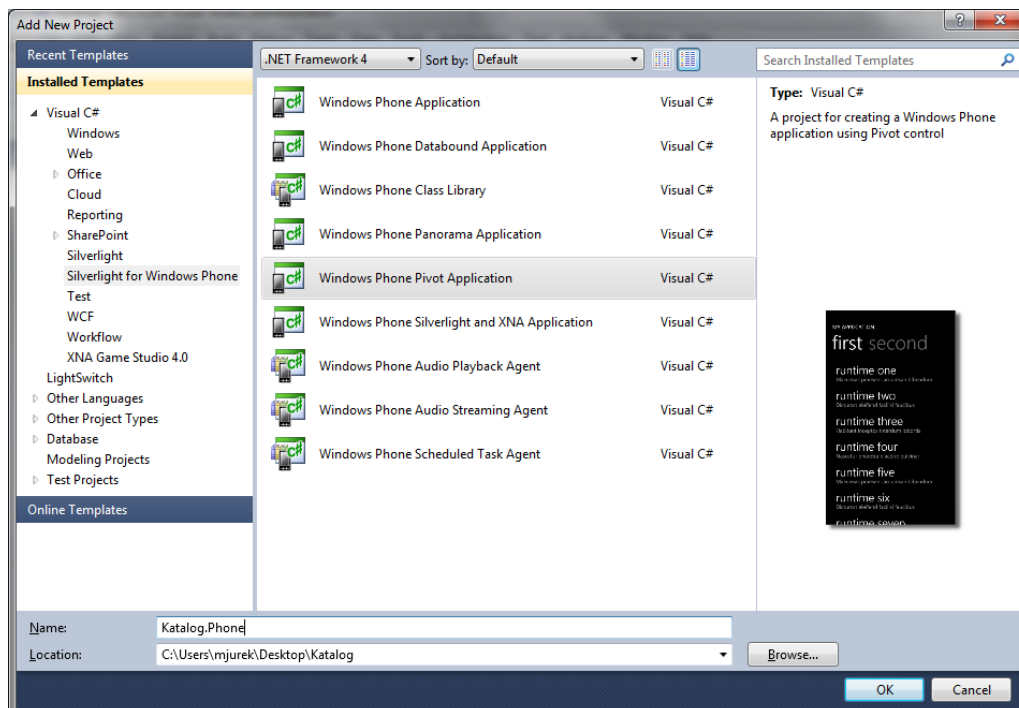
Rovněž na portálu <http://windows.azure.com> můžete přejít na položku Hosted Services a zkontrolovat, že instance virtuálu byla korektně vytvořena a je v pořádku:



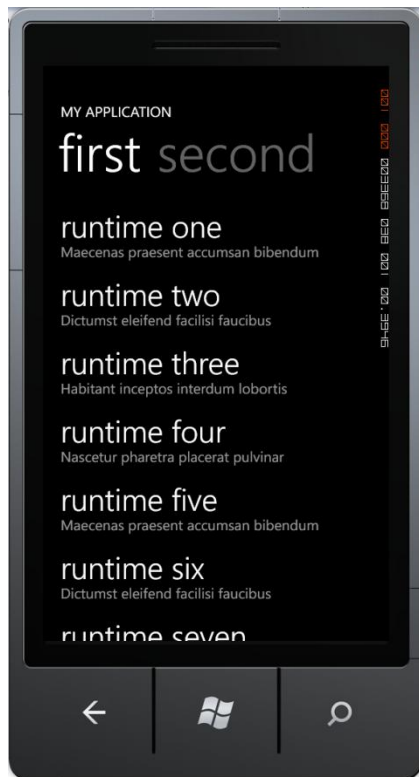
Zbývá přikročit k poslední části, kterou je vývoj Windows Phone aplikace konzumující naši datovou službu.

6. Vytvoření Windows Phone aplikace

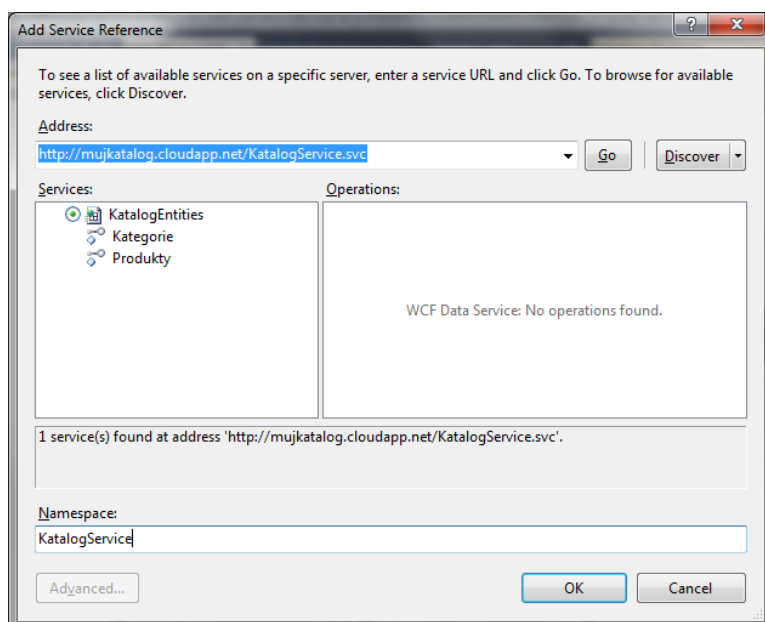
Začneme vytvořením nového projektu typu Windows Phone Pivot Application, který nazveme Katalog.Phone:



Vybereme verzi Windows Phone OS 7.1 a pomocí Debug/Start Debugging si ověříme, že vytvořený projekt korektně běží v emulátoru (pokud vše provádíte v jednom řešení, budete muset ještě nastavit projekt Katalog.Phone jako startovací pomocí volby Set as StartUp Project):



Jako první krok přidáme referenci na naši datovou službu. V kontextovém menu telefonní aplikace zvolíme Add Service Reference a zadáme adresu datové služby a potvrdíme Go. Namespace změníme na KatalogService a dokončíme stiskem OK:



De facto standardem pro vytváření Windows Phone aplikací pracujících s daty je používání návrhového vzoru Model-View-ViewModel (MVVM), který plně odděluje kód od XAML uživatelského rozhraní a maximálně využívá data-binding pro bezpracnou manipulaci s daty. Více o tomto návrhovém vzoru se můžete dočíst např. [zde](#). Soubory s kódem jsou již vygenerovány ve složce ViewModels. V ní můžete smazat soubor ItemViewModel, který nebudeme v tomto cvičení potřebovat, soubor MainViewModel.cs pak promažte, aby obsahoval pouze následující kód:

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;

namespace Katalog.Phone
{
    public class MainViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(String propertyName)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (null != handler)
            {
                handler(this, new PropertyChangedEventArgs(propertyName));
            }
        }
    }
}
```

Do třídy MainViewModel.cs přidejte:

- referenci na datovou službu včetně správného URL
- privátní člen pro kolekci kategorií typu DataServiceCollection<Kategorie>
- vlastnost zpřístupňující výše uvedený člen s notifikací o změně vlastnosti
- metodu LoadData, která asynchronně volá datovou službu a získá z ní seznam kategorií

Všechny změny vidíte v následujícím fragmentu kódu vyžlucené:

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Katalog.Phone.KatalogService;
using System.Data.Services.Client;

namespace Katalog.Phone
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private KatalogEntities _katalog = new KatalogEntities(new
Uri("http://mujkatalog.cloudapp.net/KatalogService.svc"));

        private DataServiceCollection<Kategorie> _seznamKategorii = new
DataServiceCollection<Kategorie>();

        public DataServiceCollection<Kategorie> SeznamKategorii
        {
            get
```

```

    {
        return _seznamKategorii;
    }
    set
    {
        if (value == _seznamKategorii) return;
        _seznamKategorii = value;
        NotifyPropertyChanged("SeznamKategorii");
    }
}

public void LoadData()
{
    DataServiceQuery<Kategorie> dotazSeznamKategorii = _katalog.Kategorie;
    _seznamKategorii.LoadAsync(dotazSeznamKategorii);
}

public event PropertyChangedEventHandler PropertyChanged;
... (atd.)

```

Dále otevřete soubor App.xaml.cs a odstraňte kompilační chybu v metodě Application_Activated následovně:

```

private void Application_Activated(object sender, ActivatedEventArgs e)
{
    App.ViewModel.LoadData();
}

```

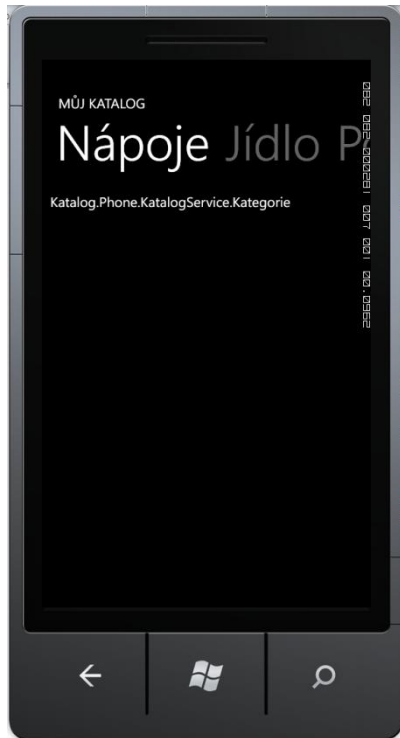
Analogicky odstraňte stejnou chybu v metodě MainPage_Loaded v souboru MainPage.xaml.cs. Pověšimněte si, že v konstruktoru této třídy se jako datový kontext nastavuje nová instance naší třídy MainViewModel. To umožňuje velmi elegantní zobrazení dat v uživatelském rozhraní pomocí data bindingu, v souboru MainPage.xaml stačí provést následující úpravy:

```

<!--Pivot Control-->
<controls:Pivot Title="MŮJ KATALOG" ItemsSource="{Binding SeznamKategorii}">
    <controls:Pivot.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Jmeno}"/>
        </DataTemplate>
    </controls:Pivot.HeaderTemplate>
</controls:Pivot>

```

Po spuštění byste již po natažení dat měli vidět v horním řádku ovládacího prvku Pivot seznam dostupných kategorií:



Zbývá dokončit zobrazování produktů v kategorii. Do třídy MainViewModel.cs přidáme:

- privátní proměnnou k uložení právě vybrané kategorie
- vlastnost zpřístupňující právě vybranou kategorii
- kód pro stažení produktů patřících do dané kategorie, pokud ještě nebyly staženy z datové služby

Finální kód třídy MainViewModel.cs vypadá následovně, přidané části kódu jsou vyzlucené:

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Katalog.Phone.KatalogService;
using System.Data.Services.Client;
using System.Linq;

namespace Katalog.Phone
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private KatalogEntities _katalog = new KatalogEntities(new
Uri("http://mujkatalog.cloudapp.net/KatalogService.svc"));

        private DataServiceCollection<Kategorie> _seznamKategorii = new
DataServiceCollection<Kategorie>();

        public DataServiceCollection<Kategorie> SeznamKategorii
        {
            get
            {
                return _seznamKategorii;
            }
            set
            {
            }
        }
    }
}
```

```

        if (value == _seznamKategorii) return;
        _seznamKategorii = value;
        NotifyPropertyChanged("SeznamKategorii");
    }
}

public void LoadData()
{
    DataServiceQuery<Kategorie> dotazSeznamKategorii = _katalog.Kategorie;
    _seznamKategorii.LoadAsync(dotazSeznamKategorii);
}

private Kategorie _vybranaKategorie;

public Kategorie VybranaKategorie
{
    get
    {
        return _vybranaKategorie;
    }
    set
    {
        if (value == _vybranaKategorie) return;
        _vybranaKategorie = value;
        NotifyPropertyChanged("VybranaKategorie");
        if (_vybranaKategorie.Produkty.Count == 0)
            _vybranaKategorie.Produkty.LoadAsync(_katalog.Produkty.Where(p =>
p.Kategorie.ID == _vybranaKategorie.ID));
    }
}

public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged(String propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (null != handler)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
}

```

Poslední úpravou je změna vzhledu stránky v souboru MainPage.xaml, kde je třeba:

- Přidat vazbu mezi vybranou položkou pivotu a příslušnou vlastností třídy MainViewModel s uvedením obousměrné vazby (UI též aktualizuje ViewModel)
- Přidat definici obsahu stránky pivotu jako ListBox s vazbou na vlastnost Produkty aktuální kategorie
- Přidat obsah položky ListBoxu s vazbou na vlastnosti Nazev, Mnozstvi a Cena

Konečný obsah souboru je následovný, změny jsou opět žlutě:

```

<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">
    <!--Pivot Control-->
    <controls:Pivot Title="MŮJ KATALOG" ItemsSource="{Binding SeznamKategorii}"
SelectedItem="{Binding VybranaKategorie, Mode=TwoWay}">

```

```

<controls:Pivot.HeaderTemplate>
    <DataTemplate>
        <TextBlock Text="{Binding Jmeno}"/>
    </DataTemplate>
</controls:Pivot.HeaderTemplate>
<controls:Pivot.ItemTemplate>
    <DataTemplate>
        <ListBox Margin="0,0,-12,0" ItemsSource="{Binding Produkty}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Margin="0,0,0,12">
                        <TextBlock Text="{Binding Nazev}"
Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                        <StackPanel Orientation="Horizontal">
                            <TextBlock Text="{Binding Mnozstvi}" Width="120"
Style="{StaticResource PhoneTextSubtleStyle}"/>
                            <TextBlock Text="{Binding Cena,
StringFormat='0.00'}" Style="{StaticResource PhoneTextSubtleStyle}"/>
                        </StackPanel>
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </DataTemplate>
</controls:Pivot.ItemTemplate>
</controls:Pivot>
</Grid>

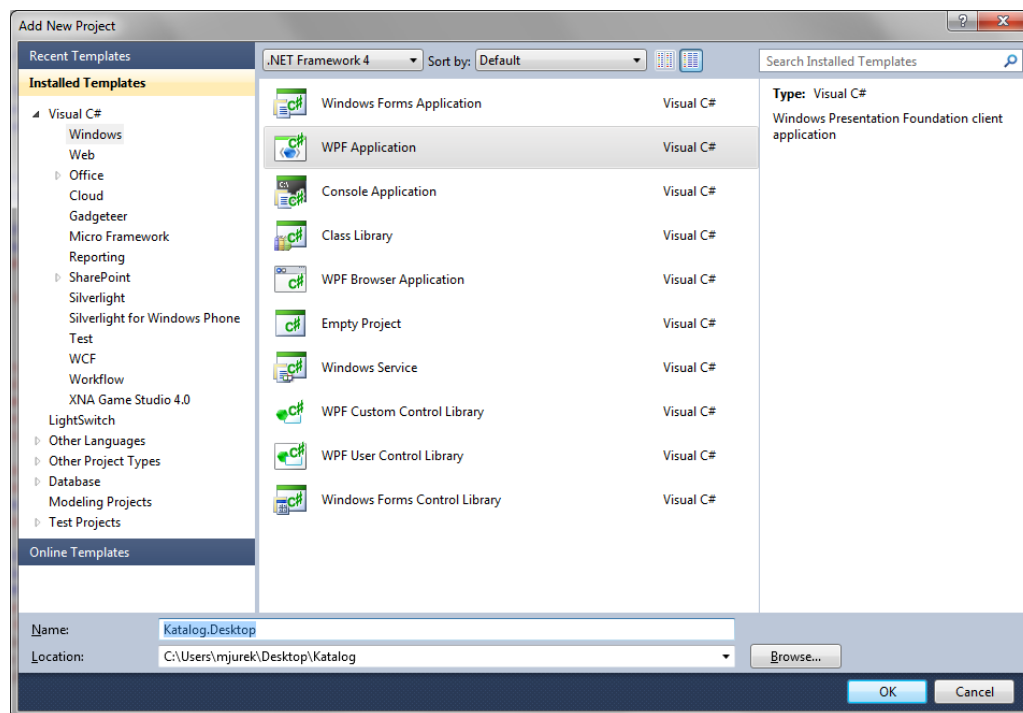
```

Po spuštění by aplikace měla korektně fungovat a vypadat následovně:



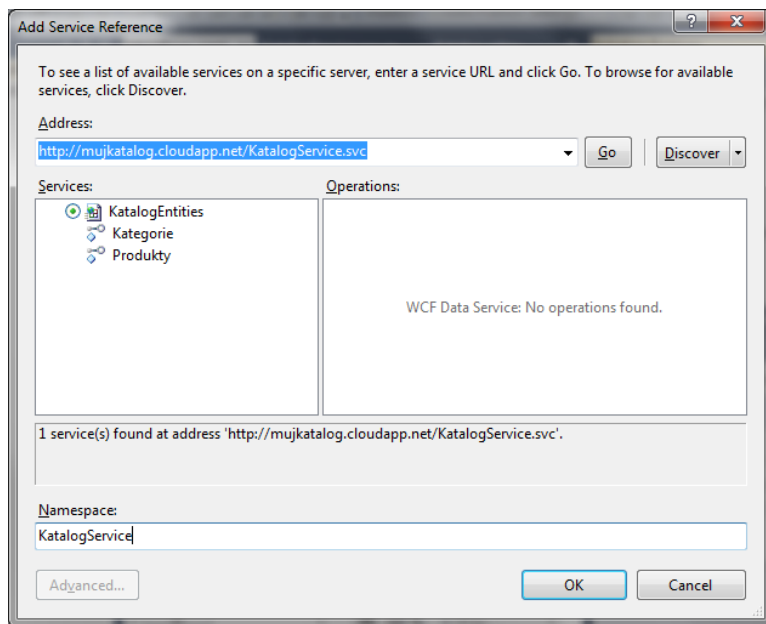
7. Vytvoření desktopové aplikace

Kdo preferuje desktopové aplikace, máme variantu i pro něj. V tradičních aplikacích se sice častěji používají webové služby volané podle SOAP konvencí, ale to není na závadu – postup by byl velmi podobný. Začneme vytvořením nového projektu typu WPF Application, který nazveme Katalog.Desktop:



Pokud vše provádíte v jednom řešení, budete muset ještě nastavit projekt Katalog.Desktop jako startovací pomocí volby Set as StartUp Project).

Jako první krok přidáme referenci na naši datovou službu. V kontextovém menu telefonní aplikace zvolíme Add Service Reference a zadáme adresu datové služby a potvrdíme Go. Namespace změníme na KatalogService a dokončíme stiskem OK:



De facto standardem pro vytváření WPF aplikací pracujících s daty je používání návrhového vzoru Model-View-ViewModel (MVVM), který plně odděluje kód od XAML uživatelského rozhraní a maximálně využívá data-binding pro bezpracnou manipulaci s daty. Více o tomto návrhovém vzoru se můžete dočíst např. [zde](#). V projektu pro desktopovou aplikaci si založte novou třídu MainViewModel.cs, do které doplňte následující kód implementující rozhraní INotifyPropertyChanged, které je nezbytné pro správnou funkci data bindingu:

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;

namespace Katalog.Desktop
{
    public class MainViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(String propertyName)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (null != handler)
            {
                handler(this, new PropertyChangedEventArgs(propertyName));
            }
        }
    }
}
```

Do třídy MainViewModel.cs přidejte:

- referenci na datovou službu včetně správného URL
- privátní člen pro kolekci kategorií typu DataServiceCollection<Kategorie>
- vlastnost zpřístupňující výše uvedený člen s notifikací o změně vlastnosti
- metodu LoadData, která asynchronně volá datovou službu a získá z ní seznam kategorií

Všechny změny vidíte v následujícím fragmentu kódu vyžlucené:

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Katalog.Desktop.KatalogService;
using System.Data.Services.Client;

namespace Katalog.Desktop
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private KatalogEntities _katalog = new KatalogEntities(new
        Uri("http://mujkatalog.cloudapp.net/KatalogService.svc"));

        private DataServiceCollection<Kategorie> _seznamKategorii = new
        DataServiceCollection<Kategorie>();

        public DataServiceCollection<Kategorie> SeznamKategorii
        {
            get
            {
                return _seznamKategorii;
            }
            set
            {
                if (value == _seznamKategorii) return;
                _seznamKategorii = value;
                NotifyPropertyChanged("SeznamKategorii");
            }
        }

        public void LoadData()
        {
            DataServiceQuery<Kategorie> dotazSeznamKategorii = _katalog.Kategorie;
            _seznamKategorii.Load(dotazSeznamKategorii);
        }

        public event PropertyChangedEventHandler PropertyChanged;
    }
    ... (atd.)
}
```

Pro dobrou odezvu aplikace by bylo vhodnější používat asynchronní volání služby na pozadí, ale nebudeme si komplikovat v tomto jednoduchém příkladě život. Naši třídu musíme nastavit jako datový kontext pro okno MainWindow.xaml, proto otevřte soubor MainWindow.xaml.cs a doplňte následující kód:

```
public MainWindow()
{
    InitializeComponent();
    this.Loaded += new RoutedEventHandler(MainWindow_Loaded);
}

void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    MainViewModel mvm = new MainViewModel();
    mvm.LoadData();
}
```

```

        this.DataContext = mvm;
    }

```

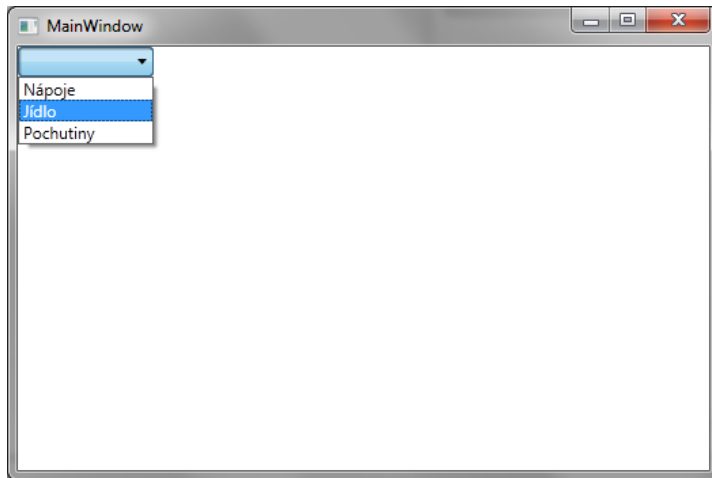
Návrhový vzor umožňuje velmi elegantní zobrazení dat v uživatelském rozhraní pomocí data bindingu, v souboru MainWindow.xaml stačí přidat ComboBox pro kategorie:

```

<Grid>
    <ComboBox ItemsSource="{Binding SeznamKategorii}" VerticalAlignment="Top"
HorizontalAlignment="Left" Width="100">
        <ComboBox.ItemTemplate>
            <DataTemplate>
                <TextBlock Text="{Binding Jmeno}"/>
            </DataTemplate>
        </ComboBox.ItemTemplate>
    </ComboBox>
</Grid>

```

Po spuštění byste již po natažení dat měli vidět v horním řádku formuláře seznam dostupných kategorií:



Zbývá dokončit zobrazování produktů v kategorii. Do třídy MainViewModel.cs přidáme:

- privátní proměnnou k uložení právě vybrané kategorie
- vlastnost zpřístupňující právě vybranou kategorii
- kód pro stažení produktů patřících do dané kategorie, pokud ještě nebyly staženy z datové služby

Finální kód třídy MainViewModel.cs vypadá následovně, přidané části kódu jsou vyzlucené:

```

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Katalog.Desktop.KatalogService;
using System.Data.Services.Client;
using System.Linq;

namespace Katalog.Desktop
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private KatalogEntities _katalog = new KatalogEntities(new
Uri("http://mujkatalog.cloudapp.net/KatalogService.svc"));

```

```

        private DataServiceCollection<Kategorie> _seznamKategorii = new
        DataServiceCollection<Kategorie>();

        public DataServiceCollection<Kategorie> SeznamKategorii
        {
            get
            {
                return _seznamKategorii;
            }
            set
            {
                if (value == _seznamKategorii) return;
                _seznamKategorii = value;
                NotifyPropertyChanged("SeznamKategorii");
            }
        }

        public void LoadData()
        {
            DataServiceQuery<Kategorie> dotazSeznamKategorii = _katalog.Kategorie;
            _seznamKategorii.Load(dotazSeznamKategorii);
        }

        private Kategorie _vybranaKategorie;

        public Kategorie VybranaKategorie
        {
            get
            {
                return _vybranaKategorie;
            }
            set
            {
                if (value == _vybranaKategorie) return;
                _vybranaKategorie = value;
                NotifyPropertyChanged("VybranaKategorie");
                if (_vybranaKategorie.Produkty.Count == 0)
                    _vybranaKategorie.Produkty.Load(_katalog.Produkty.Where(p =>
p.Kategorie.ID == _vybranaKategorie.ID));
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(String propertyName)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (null != handler)
            {
                handler(this, new PropertyChangedEventArgs(propertyName));
            }
        }
    }
}

```

Poslední úpravou je změna vzhledu stránky v souboru MainWindow.xaml, kde je třeba:

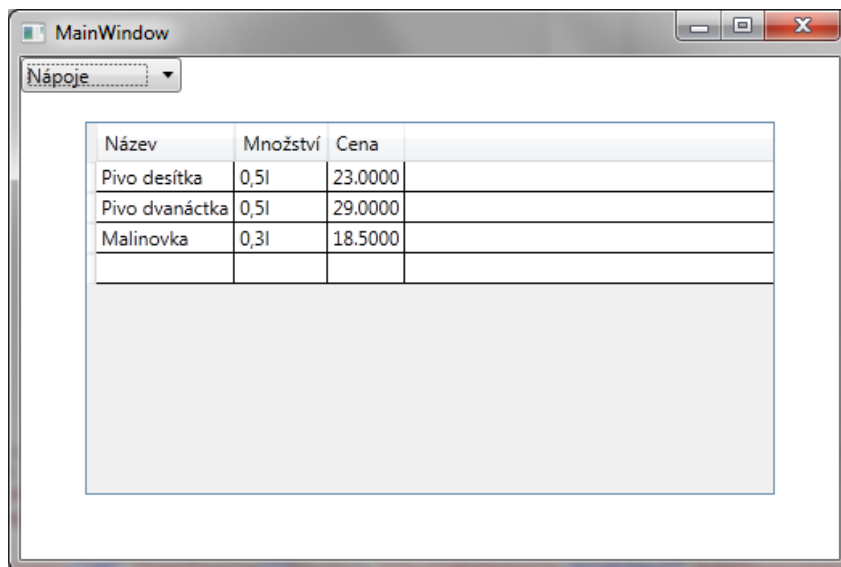
- Přidat vazbu mezi vybranou položkou combo boxu a příslušnou vlastností třídy MainViewModel

- Přidat definici obsahu stránky pivotu jako DataGrid s vazbou na vlastnost Produkty aktuální kategorie se třemi definovanými sloupcečky

Změny jsou opět žlutě:

```
<Grid>
  <ComboBox ItemsSource="{Binding SeznamKategorii}" SelectedItem="{Binding
VybranaKategorie}" VerticalAlignment="Top" HorizontalAlignment="Left" Width="100">
    <ComboBox.ItemTemplate>
      <DataTemplate>
        <TextBlock Text="{Binding Jmeno}" />
      </DataTemplate>
    </ComboBox.ItemTemplate>
  </ComboBox>
  <DataGrid Margin="40" ItemsSource="{Binding VybranaKategorie.Produkty}"
AutoGenerateColumns="False">
    <DataGrid.Columns>
      <DataGridTextColumn Binding="{Binding Nazev}" Header="Název" />
      <DataGridTextColumn Binding="{Binding Mnozstvi}" Header="Množství" />
      <DataGridTextColumn Binding="{Binding Cena}" Header="Cena" />
    </DataGrid.Columns>
  </DataGrid>
</Grid>
```

Po spuštění by aplikace měla korektně fungovat a vypadat následovně:



A máme hotovo. Případné připomínky můžete zaslat na vyvojar@microsoft.com.

Příjemné zkoušení.