Kapitola 9

Seznámení s knihovnou ADO.NET

Většinu profesionálních aplikací nelze sestavovat bez toho, aby nějakým způsobem pracovaly s databází. Databáze totiž řeší problémy spojené se získáváním a uchováváním dat. Téměř každá softwarová aplikace pracuje s jednou či více databázemi. Klientská část musí mít k dispozici nějaké mechanismy pro připojení k databázím, k čemuž slouží právě knihovna ADO.NET. Každá aplikace postavená na platformě .NET vyžadující přístup k databázi je tak závislá na knihovně ADO.NET.

V této kapitole se budeme věnovat následujícím tématům:

- Představení knihovny ADO.NET
- Motivace pro vznik knihovny ADO.NET
- Přechod od knihovny ADO k ADO.NET
- Seznámení s architekturou knihovny ADO.NET
- Práce s poskytovatelem dat pro SQL Server
- Práce s poskytovatelem dat pro OLE DB
- Práce s poskytovatelem dat pro ODBC
- Poskytovatelé dat jako aplikační rozhraní

Představení knihovny ADO.NET

Před vznikem platformy .NET používali vývojáři technologie pro přístup k datům ve formě ODBC, OLE DB nebo ADO (ActiveX Data Objects). S uvedením platformy .NET vytvořila společnost Microsoft nový způsob práce s daty nazvaný *ADO.NET*.

ADO.NET je sada tříd, které programátorům na platformě .NET exponují služby pro přístup k datům, čímž jim poskytují sadu komponent pro tvorbu distribuovaných aplikací sdílejících data. Knihovna ADO.NET je nedílnou součástí rozhraní .NET Framework, které poskytuje přístup k relačním, XML a aplikačním datům. Třídy knihovny ADO.NET se nacházejí v souboru *System.Data.dll*.

Technologie ADO.NET vychází vstříc nejrůznějším požadavkům na vývoj, mezi něž patří tvorba databázových klientů a podnikových objektů střední vrstvy používaných aplikacemi, nástroji, jazyky a internetovými prohlížeči.

Motivace pro vznik knihovny ADO.NET

S rozvojem aplikačního vývoje se aplikace stávají *volně propojené*, což je architektura umožňující snazší údržbu a znovupoužití jednotlivých komponent (více informací najdete na adrese *http://www.serviceoriented.org/loosely_coupled.html*). Stále více současných aplikací používá pro zakódování dat přenášených přes síťová připojení jazyk XML, což je také způsob, jakým mohou různé aplikace běžící na různých platformách vzájemně spolupracovat.

Knihovna ADO.NET byla navržena tak, aby podporovala rozpojenou architekturu dat, těsnou integraci s jazykem XML, společnou reprezentaci dat s možností kombinovat data z několika odlišných datových zdrojů a optimalizované příslušenství pro interakci s databází, to vše přes nativní rozhraní .NET Framework.

Během vývoje knihovny ADO.NET se společnost Microsoft rozhodla implementovat následující prvky:

Rozvoj současných znalostí o knihovně ADO Návrh knihovny ADO.NET řeší řadu požadavků současného modelu vývoje aplikací. Programový model zároveň zůstává v maximální míře podobný knihovně ADO, takže vývojáři používající knihovnu ADO se nemusí vše učit znovu. Knihovna ADO.NET je nedílnou součástí rozhraní .NET Framework, přitom je však koncipována tak, aby se v ní snadno orientovali také programátoři pracující s knihovnou ADO.

Knihovna ADO.NET navíc koexistuje s knihovnou ADO. I když většina aplikací založených na platformě .NET bude psána pomocí ADO.NET, mají programátoři vyvíjející na platformě .NET prostřednictvím služeb pro spolupráci .NET COM k dispozici také knihovnu ADO.

Podpora vícevrstvého programového modelu Princip práce s rozpojenou sadou záznamů se stal ústředním bodem programového modelu. Knihovna ADO.NET nabízí podporu speciálních tříd pro rozpojené, vícevrstvé programové prostředí. Jejím řešením v oblasti budování vícevrstvých databázových aplikací je datová sada.

Integrace podpory jazyka XML Jazyk XML a přístup k datům spolu úzce souvisejí. Jazyk XML se točí kolem kódování dat a přístup k datům se stále více týká jazyka XML. Rozhraní .NET Framework nejen že webové standardy podporuje, ale je na nich zcela vystavěno.

Podpora jazyka XML je do knihovny ADO.NET zabudována na naprosto základní úrovni. Třídy pro práci s jazykem XML v rozhraní ADO.NET jsou součástí téže architektury, přičemž k jejich integraci dochází na mnoha různých úrovních. Nemusíte si tedy již vybírat mezi sadou služeb pro přístup k datům a jejich protějšky pro práci s jazykem XML, protože schopnost přecházet od jednoho k druhému je podstatnou součástí návrhu obou z nich.

Přechod od knihovny ADO k ADO.NET

Knihovna ADO je kolekcí objektů ActiveX, které jsou navrženy tak, aby pracovaly v trvale *propojeném* prostředí. Byla vystavěna nad knihovou OLE DB (na tu se podíváme v části „Práce s poskytovatelem dat pro OLE DB"), která umožňuje přístup k datům odlišný od jazyka SQL i k databázím na bázi jazyka SQL, přičemž knihovna ADO přináší rozhraní navržené pro usnadnění práce s poskytovateli knihovny OLE DB.

Nicméně přístup k datům pomocí knihovny ADO (a s knihovnou OLE DB v základech) znamená, že před dosažením datového zdroje musíte projít přes několik vrstev zajišťujících propojení. Kromě knihovny OLE DB nezbytné pro připojení k bezpočtu datových zdrojů se zde vyskytuje také starší technologie pro přístup k datům označovaná jako ODBC (Open Database Connectivity), která slouží pro připojení k ještě starším datovým zdrojům, jako je dBase nebo Paradox. Pro přístup k datovým zdrojům ODBC pomocí knihovny ADO musíte použít poskytovatele OLE DB pro ODBC (poněvadž knihovna ADO pracuje pouze přímo s knihovnou OLE DB), což přidává další vrstvy k již tak mnohovrstvému modelu.

V mnohovrstvém modelu pro přístup k datům a propojené povaze knihovny ADO byste mohli snadno spotřebovat prostředky serveru a vytvořit tak překážku ve výkonu. Knihovna ADO sloužila ve své době dobře, ale knihovna ADO.NET nabízí několik skvělých prvků, které z ní činí mnohem lepší technologii pro přístup k datům.

Knihovna ADO.NET není novou verzí knihovny ADO

Knihovna ADO.NET představuje zcela novou technologii přístupu k datům s novým návrhem, který byl vytvořen celý znovu. Nejdříve je třeba si ujasnit následující: knihovna ADO.NET nezastupuje datové objekty ActiveX (ADO). Proč? Důvodů je mnoho, ovšem dva nejdůležitější z nich jsou tyto:

- Knihovna ADO.NET není externí entitou, ale integrální součástí platformy .NET.
- Knihovna ADO.NET není kolekcí komponent ActiveX.

Název ADO.NET připomíná knihovnu ADO, protože společnost Microsoft chtěla, aby se vývojáři při používání knihovny ADO.NET cítili jako doma a nemysleli si, že se musí vše učit znovu, jak jsme si řekli již dříve, a proto ji záměrně pojmenovala tak, aby nabízela podobné prvky implementované odlišným způsobem.

Během návrhu platformy .NET si společnost Microsoft uvědomila, že do ní knihovna ADO nezapadá. Ta byla totiž dostupná jako externí balíček na bázi objektů COM (Component Object Model), což by vyžadovalo, aby se na ni aplikace postavené na platformě .NET explicitně odkazovaly. Jenže tyto aplikace jsou navržené tak, aby sdílely jediný model, kde jsou všechny knihovny integrované do jediného rámce, uspořádané do logických oborů názvů a deklarované jako veřejné pro jakoukoli aplikaci, která by je chtěla používat. Bylo tedy moudře rozhodnuto, že by se technologie přístupu k datům na platformě .NET měla přizpůsobit architektonickému modelu platformy .NET, což dalo vzniknout knihovně ADO.NET.

Knihovna ADO.NET je navržena tak, aby vyhovovala jak propojenému, tak i rozpojenému přístupu. Kromě toho obsahuje také v mnohem větší míře veledůležitý standard jazyka XML, poněvadž explozivní nárůst nasazení technologie XML přišel až po vývoji knihovny ADO. Prostřednictvím knihovny ADO.NET můžete jazyk XML využívat nejen pro přenos dat mezi aplikacemi, ale také pro export dat z vlastní aplikace do souboru XML, který uložíte lokálně ve svém systému a později v případě potřeby opět načtete.

Výkon obvykle něco stojí, ovšem v případě knihovny ADO.NET je tato cena rozhodně přiměřená. Na rozdíl od knihovny ADO knihovna ADO.NET nezabaluje transparentně poskytovatele knihovny OLE DB, ale místo toho používá *řízené poskytovatele dat* (*managed data providers)*, kteří jsou navržení vždy pro konkrétní typ datového zdroje, což jim umožňuje rozvinout jejich skutečnou sílu a pozvednout celkovou rychlost a výkon aplikace.

Knihovna ADO.NET také funguje v propojených i rozpojených prostředích. Můžete se tak připojit k databázi, zůstat při obyčejném čtení dat připojeni a poté připojení uzavřít, což je proces podobný knihovně ADO. Kde však knihovna ADO.NET opravdu zazáří, jsou rozpojená prostředí. Pokud potřebujete upravovat data v databázi, bylo by udržování nepřetržitého připojení pro server nákladné. Knihovna ADO.NET tento problém řeší tak, že nabízí sofistikovaný rozpojený model. Data se odešlou ze serveru a uloží se do mezipaměti na straně klienta. Jakmile jste připraveni provést aktualizaci databáze, pošlete data zpět na server, který za vás vyřeší aktualizace a konflikty.

V knihovně ADO.NET se při načítání dat používá objekt označovaný jako *objekt pro čtení dat* (*data reader*). Když pracujete s rozpojenými daty, ukládají se data do relační datové struktury v lokální mezipaměti buď jako *datová tabulka* nebo jako *datová sada*.

Knihovna ADO.NET a základní knihovna tříd platformy .NET

Datová sada (objekt typu `DataSet`) může uchovávat velké množství dat ve formě tabulek (objektů typu `DataTable`), jejich vztahů (objektů typu `DataRelation`) a omezení (objektů typu `Constraint`) v operační paměti počítače, které lze poté exportovat do externího souboru nebo jiné datové sady. Díky tomu, že podpora jazyka XML je integrována do knihovny ADO.NET, můžete vytvářet schémata XML a stejně tak přenášet a sdílet data pomocí dokumentů XML.

Tabulka 9.1 popisuje obory názvů, do nichž jsou seskupeny komponenty knihovny ADO.NET.

Tabulka 9.1: Obory názvů knihovny ADO.NET

Obor názvů	Popis
`System.Data`	Třídy, rozhraní, delegáty a výčty, které definují a částečně implementují architekturu knihovny ADO.NET.
`System.Data.Common`	Třídy sdílené poskytovateli dat z rozhraní .NET Framework.
`System.Data.Design`	Třídy, které lze použít pro generování datové sady vlastního typu.
`System.Data.Odbc`	Poskytovatelé dat z rozhraní .NET Framework pro ODBC.
`System.Data.OleDb`	Poskytovatelé dat z rozhraní .NET Framework pro OLE DB.
`System.Data.Sql`	Třídy, jež podporují funkcionalitu specifickou pro SQL Server.
`System.Data.OracleClient`	Poskytovatelé dat z rozhraní .NET Framework pro Oracle.
`System.Data.SqlClient`	Poskytovatelé dat z rozhraní .NET Framework pro SQL Server.
`System.Data.SqlServerCe`	Poskytovatelé dat z rozhraní .NET Compact Framework pro SQL Server Mobile.
`System.Data.SqlTypes`	Třídy pro nativní datové typy SQL Serveru.
`Microsoft.SqlServer.Server`	Komponenty pro integraci SQL Serveru a běhového prostředí CLR.

Vzhledem k tomu, že podpora jazyka XML je úzce spojená s knihovnou ADO.NET, opírají se některé její komponenty v oboru názvů `System.Data` o komponenty z oboru názvů `System.Xml`. Takže někdy je nutné uvést v okně Solution Explorer odkazy na oba obory názvů.

Tyto obory názvů jsou fyzicky implementovány jako prvky sestavení, takže pokud v aplikaci VCSE vytvoříte nový projekt aplikace, měly by se společně s odkazem na prvek sestavení `System` automa-

ticky vytvořit odkazy i na tyto prvky sestavení. Pokud však nejsou k dispozici, můžete je jednoduše přidat takto:

1. Klepněte v okně Solution Explorer pravým tlačítkem na položku References a poté zvolte příkaz Add Reference.
2. Zobrazí se dialogové okno se seznamem dostupných odkazů. Vyberte postupně (přidržením klávesy Ctrl) `System.Data`, `System.Xml` a `System` (pokud již není přítomen) a poté klepněte na tlačítko Select.
3. Klepnutím na tlačítko OK pak zvolené odkazy přidáte do svého projektu.

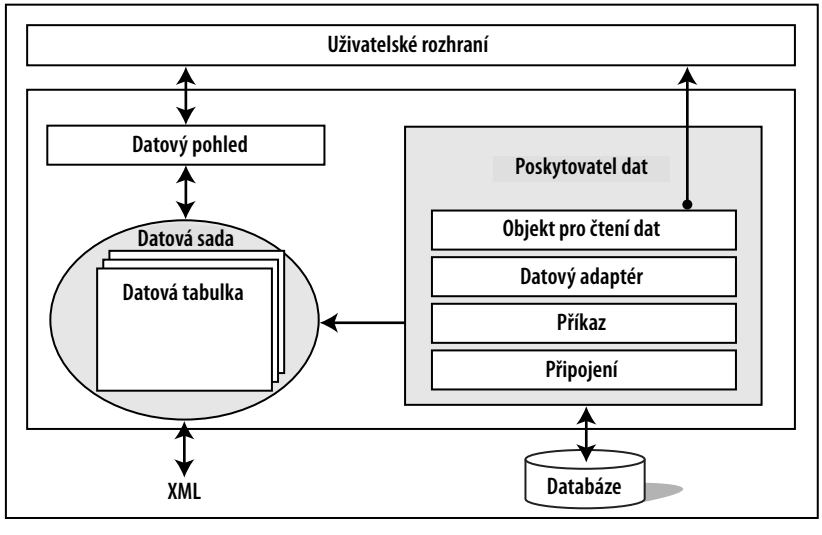
Tip: Pokud používáte kompilátor jazyka C# na příkazovém řádku (my jej v této knize používat nebudeme), můžete použít následující volby pro začlenění odkazů na požadované prvky sestavení: `/r:System.dll /r:System.Data.dll /r:System.Xml.dll`.

Jak je z oborů názvů patrné, knihovna ADO.NET umí pracovat i se staršími technologiemi, jako je OLE DB a ODBC (Open Database Connectivity). Nicméně poskytovatel dat pro SQL Server komunikuje přímo s SQL Serverem bez dodatečné vrstvy pro OLE DB nebo ODBC, takže se jedná o nejefektivnější formu připojení. Rovněž i poskytovatel dat pro Oracle přistupuje přímo k databázovému serveru Oracle.

Poznámka: Všichni hlavní dodavatelé databázových systémů podporují své vlastní poskytovatele dat v knihovně ADO.NET. V této knize zůstaneme u SQL Serveru, ovšem stejný typ kódu jazyka C# je možné použít pro jakéhokoliv poskytovatele.

Seznámení s architekturou knihovny ADO.NET

Obrázek 9.1 představuje nejdůležitější architektonické prvky knihovny ADO.NET. Podrobněji se jimi budeme zabývat v pozdějších kapitolách.



Obrázek 9.1: Architektura knihovny ADO.NET

Knihovnu ADO.NET tvoří dvě ústřední komponenty: poskytovatelé dat a datové sady.

Poskytovatel dat (*data provider*) se připojuje k datovému zdroji a podporuje přístup a manipulaci s daty. Později v této kapitole si vyzkoušíte práci se třemi různými poskytovateli dat.

Datová sada (*dataset*) podporuje rozpojené, nezávislé ukládání dat do mezipaměti v relačním stylu, přičemž dle potřeby aktualizuje datový zdroj. Datová sada obsahuje jednu nebo více tabulek. *Datová tabulka* (*data table*) je řádkovo-sloupcovou reprezentací, která poskytuje bezmála stejný logický pohled jako tabulka v databázi. Můžete tak například uložit data z tabulky `Employees` (zaměstnanci) v databázi Northwind do datové tabulky ADO.NET a pracovat s nimi dle libosti. O datových sadách a datových tabulkách se více dozvíte v kapitole 13.

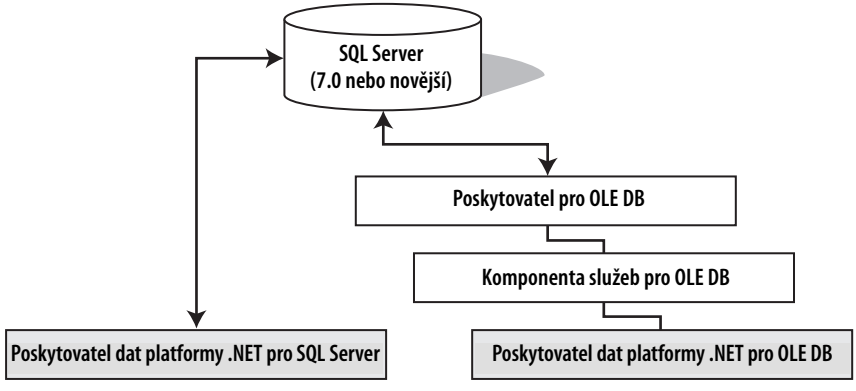
Všimněte si datového pohledu na obrázku 9.1 (jedná se o třídu `DataView`, která se nachází v oboru názvů `System.Data`). Nejedná se o komponentu poskytovatele dat. Datové pohledy se používají především pro svázání dat s formuláři v systému Windows a na webu.

Jak jste již viděli v tabulce 9.1, každý poskytovatel dat má svůj vlastní obor názvů. Ve skutečnosti je každý poskytovatel dat v podstatě implementací rozhraní v oboru názvů `System.Data`, které je specializované pro specifický typ datového zdroje.

Pokud například používáte SQL Server, pak byste měli použít poskytovatel dat pro SQL Server (`System.Data.SqlClient`), protože se jedná o nejefektivnější způsob přístupu k SQL Serveru.

Poskytovatel dat pro OLE DB podporuje přístup ke starším verzím SQL Serveru i k dalším databázím, jako je Access, DB2, MySQL a Oracle. Nicméně kvůli výkonu jsou vhodnější nativní poskytovatelé dat (tedy např. `System.Data.OracleClient`), protože poskytovatel dat pro OLE DB musí před tím, než se dostane k datovému zdroji, projít skrze dvě další vrstvy (komponenta služeb pro OLE DB a poskytovatel pro OLE DB).

Obrázek 9.2 znázorňuje rozdíl mezi použitím poskytovatelů dat pro SQL Server a OLE DB pro přístup k databázi SQL Server.



Obrázek 9.2: Odlišnosti v poskytovatelích dat pro SQL Server a pro OLE DB

Poskytovatele dat pro OLE DB byste měli použít pouze tehdy, jestliže se vaše aplikace připojuje ke starší verzi SQL Serveru (6.5 a starší) nebo k více než jednomu druhu databázového serveru současně (např. současné připojení k databázím Access a Oracle).

Neexistují žádná pevně daná pravidla. Jestli chcete, můžete klidně používat poskytovatele dat pro OLE DB pro přístup k SQL Serveru i poskytovatele dat pro Oracle (`System.Data.OracleClient`), důležité ovšem je, abyste si pro své účely vybrali toho nejvhodnějšího poskytovatele. Vezmeme-li v potaz výkonnostní výhody poskytovatelů dat specifických pro konkrétní servery, pak při práci s SQL Serverem byste měli v 99 % případů sáhnout po třídách v oboru názvů `System.Data.SqlClient`.

Ještě před tím, než se podíváme, co jednotliví poskytovatelé nabízejí a jak se používají, musíme si ujasnit jejich hlavní funkcionalitu. Každý poskytovatel dat platformy .NET je navržen tak, aby perfektně prováděl následující dvě věci:

- Poskytování přístupu k datům s aktivním připojením k datovému zdroji.
- Poskytování přenosu dat do a z rozpojených datových sad a datových tabulek.

Databázová připojení se ustavují pomocí připojovací třídy poskytovatele dat (např. `System.Data.SqlClient.SqlConnection`). Další komponenty, mezi něž patří objekty pro čtení dat, příkazy a datové adaptéry, podporují získávání dat, provádění příkazů jazyka SQL a čtení nebo zápis do datových sad nebo datových tabulek.

Jak můžete vidět, každý poskytovatel dat má prefix podle typu datového zdroje, k němuž se připojuje (např. poskytovatel dat pro SQL Server má prefix `Sql`), takže například jeho připojovací třída nese název `SqlConnection`. Připojovací třída poskytovatele dat pro OLE DB se jmenuje `OleDbConnection`.

Podívejme se nyní, jak s těmito třemi poskytovateli dat, které lze použít pro přístup k SQL Serveru, můžeme pracovat.

Práce s poskytovatelem dat pro SQL Server

Poskytovatel dat platformy .NET pro SQL Server se nachází v oboru názvů `System.Data.SqlClient`. I když můžete pro připojení k SQL Serveru použít také obor názvů `System.Data.OleDb`, společnost Microsoft navrhla obor názvů `System.Data.SqlClient` speciálně pro použití s SQL Serverem, takže pracuje efektivnějším a optimalizovanějším způsobem než `System.Data.OleDb`. Příčina tohoto efektivního a optimalizovaného přístupu spočívá v tom, že tento poskytovatel dat nekomunikuje se serverem přes několik vrstev, ale přímo pomocí svého nativního síťového protokolu.

Tabulka 9.2 popisuje několik důležitých tříd v oboru názvů `SqlClient`.

Tabulka 9.2: Běžně používané třídy v oboru názvů SqlClient

Třída	Popis
`SqlCommand`	Provádí dotazy, příkazy a uložené procedury jazyka SQL.
`SqlConnection`	Představuje připojení k databázi SQL Serveru.
`SqlDataAdapter`	Představuje most mezi datovou sadou a datovým zdrojem.
`SqlDataReader`	Poskytuje přístup k datovému proudu s výsledky, které lze pouze číst a mezi kterými se lze pohybovat pouze vpřed.
`SqlError`	Uchovává informace o chybách a upozorněních SQL Serveru.
`SqlException`	Definuje výjimku vyvolanou chybou či upozorněním SQL Serveru.
`SqlParameter`	Představuje parametr příkazu.
`SqlTransaction`	Představuje transakci SQL Serveru.

Další obor názvů `System.Data.SqlTypes` mapuje datové typy SQL Serveru na typy platformy .NET, což zlepšuje výkon a usnadňuje vývojářům práci.

Podívejme se nyní na příklad, který používá poskytovatele dat pro SQL Server. Nezaměřuje se ani tak na připojení a získávání dat, ale spíše vás seznámí s tím, co vás čeká v následných kapitolách.

Vyzkoušejte: Vytvoření jednoduché konzolové aplikace pomocí poskytovatele dat pro SQL Server

Sestavíte jednoduchý projekt typu Console Application, který otevře připojení a spustí nějaký dotaz, a to pomocí oboru názvů `SqlClient` a databáze Northwind. Načtená data zobrazíte v okně příkazového řádku.

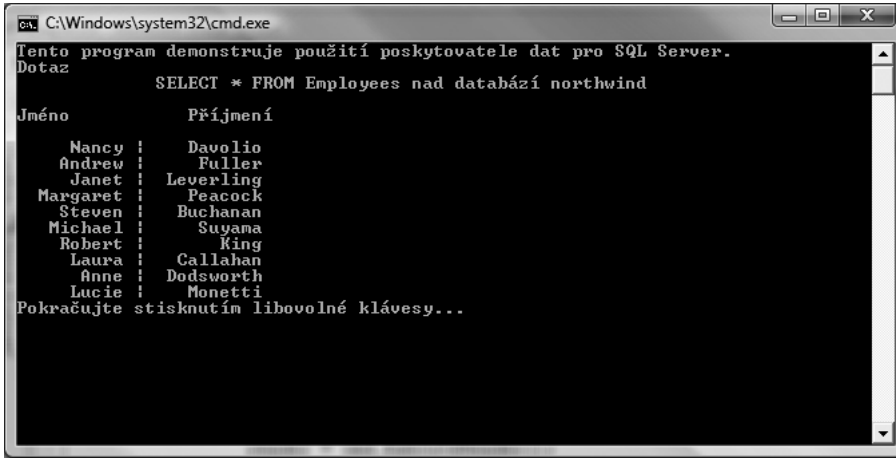
1. Otevřete Visual Studio 2008 a vytvořte nový projekt Visual C# typu Console Application s názvem Chapter09.
2. Klepněte pravým tlačítkem na projekt Chapter09 a přejmenujte jej na „SqlServerProvider".
3. Klepněte pravým tlačítkem na soubor *Program.cs* a přejmenujte jej na *SqlServerProvider.cs*. Jakmile budete dotázáni, zda přejmenovat všechny odkazy na slovo „Program", můžete klepnout buď na tlačítko Yes nebo No.
4. Vzhledem k tomu, že tento příklad budete vytvářet úplně od začátku, otevřete soubor *SqlServerProvider.cs* v okně editoru a nahraďte jeho obsah kódem ve výpisu 9.1.

Výpis 9.1: Soubor SqlServerProvider.cs

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace Chapter09
{
   class SqlServerProvider
   {
      static void Main(string[] args)
      {
         // připrav připojovací řetězec
         string connString = @"
          server = .\sqlexpress;
          integrated security = true;
          database = northwind";

         // připrav řetězec s dotazem
         string sql = @"
          SELECT * FROM Employees";

         // deklaruj proměnné pro připojení a čtení dat
         SqlConnection conn = null;
```

```
            SqlDataReader reader = null;

            try
            {
               // otevři připojení
               conn = new SqlConnection(connString);
               conn.Open();

               // proveď dotaz
               SqlCommand cmd = new SqlCommand(sql, conn);
               reader = cmd.ExecuteReader();

               // zobraz záhlaví výstupu
               Console.WriteLine("Tento program demonstruje "
                + "použití poskytovatele dat pro SQL Server.");
               Console.WriteLine("Dotaz {1} nad databází {0}\n",
                conn.Database, cmd.CommandText);
               Console.WriteLine("Jméno\t\tPříjmení\n");

               // zpracuj výslednou sadu
               while(reader.Read())
               {
                  Console.WriteLine("{0} | {1}",
                   reader["FirstName"].ToString().PadLeft(10),
                   reader[1].ToString().PadLeft(10));
               }
            }
            catch (Exception e)
            {
               Console.WriteLine("Chyba: " + e);
            }
            finally
            {
               // uzavři připojení
               reader.Close();
               conn.Close();
            }
         }
      }
   }
```

5. Uložte projekt a stiskem kláves Ctrl+F5 jej spusťte. Měl by se objevit výsledek jako na obrázku 9.3.



Obrázek 9.3: Přístup k databázi Northwind přes poskytovatele dat pro SQL Server

Jak to funguje

Pojďme se nyní podívat na to, jak výše uvedený kód funguje, přičemž začneme direktivami `using`:

```
using System;
using System.Data;
using System.Data.SqlClient;
```

Odkaz na obor názvů `System.Data` ve skutečnosti není v takto malém prográmku nutný, poněvadž žádné jeho členy přímo nepoužíváte, je však dobrým zvykem jej vždy uvádět. Odkaz na obor názvů `System.Data.SqlClient` je nezbytný, protože chcete používat jednoduché názvy jeho členů.

Připojovací řetězec uvádíte s *parametry* (dvojice klíč-hodnota) vhodnými pro relaci serveru SQL Server Express:

```
// připrav připojovací řetězec
string connString = @"
 server = .\sqlexpress;
 integrated security = true;
 database = northwind";
```

Tento připojovací řetězec obsahuje následující parametr:

```
integrated security = true;
```

který stanoví autentizaci pomocí systému Windows, takže k instanci SQLEXPRESS mohou přistupovat všichni uživatelé přihlášení do systému Windows.

Poté píšete dotaz jazyka SQL:

```
// připrav řetězec s dotazem
string sql = @"
 SELECT * FROM Employees";
```

Tip: Připojovací řetězec a dotaz jazyka SQL je zapsán jako doslovný řetězec (verbatim string) jazyka C#, což nám umožňuje odsazovat kód pohodlným způsobem. Připojovací řetězec je ve skutečnosti analyzován ještě před přiřazením do vlastnosti `ConnectionString`, takže nové řádky a dodatečné mezery jsou bezvýznamné. Jazyk SQL může obsahovat dodatečné nové řádky a mezery, takže jej můžete libovolně formátovat pro lepší čitelnost a udržovatelnost.

Dále deklarujete proměnné pro připojení a čtení dat, aby byly k dispozici pro zbývající část kódu:

```
// deklaruj proměnné pro připojení a čtení dat
SqlConnection conn = null;
SqlDataReader reader = null;
```

Poté vytvoříte připojení a otevřete jej:

```
try
{
   // otevři připojení
   conn = new SqlConnection(connString);
   conn.Open();
```

To vše (a zbývající práci s databází) provedete v bloku `try` pro ošetření výjimek, konkrétně pak výjimek vyvolaných knihovnou ADO.NET v reakci na databázové chyby. Knihovna ADO.NET vyvolá výjimku, pokud připojovací parametry nejsou syntakticky správné, takže i na to musíte být připraveni. Pokud byste deklaraci připojení (a objektu pro čtení dat) odložili až do bloku `try`, neměli byste jej k dispozici v bloku `finally`, kde musíte připojení uzavřít. Je třeba poznamenat, že při vytváření připojení ve skutečnosti nedojde k připojení k databázi, které proběhne až v okamžiku, kdy na tomto připojení zavoláte metodu `Open`.

K provedení dotazu nejdříve vytvoříte objekt pro příkaz, jehož konstruktoru předáte kód jazyka SQL, který se má provést, a připojení, na kterém se má provést. Dále zavoláním metody `ExecuteReader()` na objektu pro příkaz vytvoříte objekt pro čtení dat. Tím se nejen provede připravený příkaz, ale také dojde ke zřízení objektu pro čtení dat. Všimněte si, že na rozdíl od většiny objektů nemůžete objekt pro čtení dat vytvořit pomocí výrazu `new`.

```
// proveď dotaz
SqlCommand cmd = new SqlCommand(sql, conn);
reader = cmd.ExecuteReader();
```

Nyní vytvoříte pomocí vlastností připojení a příkazu obsahujících název databáze (`Database`) a text dotazu (`CommandText`) záhlaví pro svůj výstup:

```
// zobraz záhlaví výstupu
Console.WriteLine("Tento program demonstruje "
 + "použití poskytovatele dat pro SQL Server.");
Console.WriteLine("Dotaz {1} nad databází {0}\n",
 conn.Database, cmd.CommandText);
Console.WriteLine("Jméno\t\tPříjmení\n");
```

Všechny řádky výsledné sady získáváte voláním metody `Read` na objektu pro čtení dat, která vrací hodnotu `true`, je-li další řádek k dispozici, nebo `false` v opačném případě. Všimněte si, že objekt pro čtení dat je umístěn těsně před první řádek předcházející prvnímu volání metody `Read`.

```
// zpracuj výslednou sadu
while(reader.Read())
{
   Console.WriteLine("{0} | {1}",
    reader["FirstName"].ToString().PadLeft(10),
    reader[1].ToString().PadLeft(10));
}
```

Ke sloupci každého řádku přistupujete pomocí indexeru (zde jde o vlastnost `SqlDataReader.Item`) objektu pro čtení dat, který je přetížen tak, aby přijímal buď název sloupce, nebo index počítaný od nuly. Ve výše uvedeném kódu používáte obě možnosti, abyste viděli, jak se indexer používá. Je však třeba říci, že přístup ke sloupcům přes číselné indexy je ve srovnání s názvy efektivnější.

Dále ošetřujete výjimky. Ne sice úplně sofistikovaně, alespoň si však osvojujete dobrý zvyk. Obsluze výjimek se budeme důkladně věnovat v kapitole 16.

```
catch (Exception e)
{
   Console.WriteLine("Chyba: " + e);
}
```

Nakonec v bloku `finally` zavíráte voláním příslušných metod `Close` objekt pro čtení dat a připojení. Obecně lze říci, že byste vše měli zavírat v bloku `finally`, abyste měli jistotu, že budou uzavřeny bez ohledu na to, co se stane v bloku `try`.

```
finally
{
   // uzavři připojení
   reader.Close();
   conn.Close();
}
```

Technicky vzato zavře uzavření připojení také objekt pro čtení dat, ale zavření obou (ve výše uvedeném pořadí) je dalším dobrým zvykem. Připojení s otevřeným objektem pro čtení dat nelze použít k jinému účelu, dokud nebude tento objekt uzavřen.

Práce s poskytovatelem dat pro OLE DB

Mimo platformu .NET představuje knihovna OLE DB stále výkonnou technologii společnosti Microsoft pro přístup k datům. Poskytovatel dat pro OLE DB je tu již řadu let. Pokud jste někdy v minulosti programovali s databází MS Access, tak si možná vzpomenete, že jste se k ní připojovali pomocí poskytovatele Microsoft Jet OleDb 3.5 nebo 4.0. Jeho prostřednictvím můžete přistupovat k datům uloženým v libovolném formátu, takže i v knihovně ADO.NET hraje významnou roli při přístupu k datovým zdrojům, které nemají v knihovně ADO.NET vlastní poskytovatele dat.

Poskytovatel dat v rozhraní .NET Framework pro knihovnu OLE DB je umístěn v oboru názvů `System.Data.OleDb`. Tabulka 9.3 nabízí popis několika důležitých tříd v oboru názvů `OleDb`.

Tabulka 9.3: Běžně používané třídy v oboru názvů OleDb

Třída	Popis
`OleDbCommand`	Provádí dotazy, příkazy a uložené procedury jazyka SQL.
`OleDbConnection`	Představuje připojení k datovému zdroji OLE DB.
`OleDbDataAdapter`	Představuje most mezi datovou sadou a datovým zdrojem.
`OleDbDataReader`	Poskytuje přístup k datovému proudu s výsledky, které lze pouze číst a mezi kterými se lze pohybovat pouze vpřed.
`OleDbError`	Uchovává informace o chybách a upozorněních vrácených datovým zdrojem.
`OleDbParameter`	Představuje parametr příkazu.
`OleDbTransaction`	Představuje transakci SQL.

Všimněte si podobnosti s poskytovatelem dat v oboru názvů `SqlClient`. Rozdíly v jejich implementaci jsou transparentní, přičemž uživatelské rozhraní zůstává v podstatě stejné.

Poskytovatel dat knihovny ADO.NET pro OLE DB vyžaduje, aby byl v připojovacím řetězci uveden poskytovatel knihovny OLE DB. Některé z nich uvádí tabulka 9.4.

Tabulka 9.4: Několik poskytovatelů knihovny OLE DB

Poskytovatel	Popis
`DB2OLEDB`	Poskytovatel knihovny Microsoft OLE DB pro DB2
`SQLOLEDB`	Poskytovatel knihovny Microsoft OLE DB pro SQL Server
`Microsoft.Jet.OLEDB.4.0`	Poskytovatel knihovny Microsoft OLE DB pro Access (používá ovladač „Jet Engine“)
`MSDAORA`	Poskytovatel knihovny Microsoft OLE DB pro Oracle.
`MSDASQL`	Poskytovatel knihovny Microsoft OLE DB pro ODBC.

Nyní si vyzkoušíte použití poskytovatele dat knihovny OLE DB pro přístup k databázi Northwind (`SQLOLEDB`), což bude spočívat v provedení několika přímočarých změn v kódu ve výpisu 9.1. (Při vývoji reálné aplikace byste samozřejmě sáhli po poskytovateli dat přímo pro SQL Server, který je efektivnější.)

Vyzkoušejte: Vytvoření jednoduché konzolové aplikace pomocí poskytovatele dat pro OLE DB

V tomto příkladu uvidíte, jak přistupovat k databázi Northwind pomocí poskytovatele dat pro OLE DB.

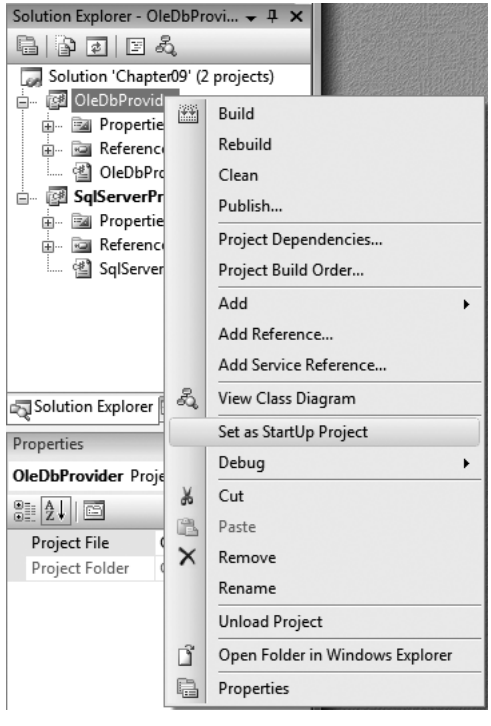
1. V okně Solution Explorer přidejte do řešení Chapter09 nový projekt typu Console Application s názvem „OleDbProvider“. Přejmenujte soubor *Program.cs* na *OleDbProvider.cs*. V okně editoru nahraďte vygenerovaný kód kódem ve výpisu 9.2, v němž jsou tučně vyznačeny změny oproti výpisu 9.1.

Výpis 9.2: Soubor OleDbProvider.cs

```
using System;
using System.Data;
using System.Data.OleDb;

namespace Chapter09
{
   class OleDbProvider
   {
      static void Main(string[] args)
      {
         // připrav připojovací řetězec
         string connString = @"
          provider = sqloledb;
          data source = .\sqlexpress;
          integrated security = sspi;
          initial catalog = northwind";

         // připrav řetězec s dotazem
         string sql = @"
          SELECT * FROM Employees";

        // deklaruj proměnné pro připojení a čtení dat
        OleDbConnection conn = null;
        OleDbDataReader reader = null;

        try
        {
           // otevři připojení
           conn = new OleDbConnection(connString);
           conn.Open();

           // proveď dotaz
           OleDbCommand cmd = new OleDbCommand(sql, conn);
           reader = cmd.ExecuteReader();

            // zobraz záhlaví výstupu
            Console.WriteLine("Tento program demonstruje "
             + "použití poskytovatele dat pro OLE DB.");
            Console.WriteLine("Dotaz {1} nad databází {0}\n",
             conn.Database, cmd.CommandText);
            Console.WriteLine("Jméno\t\tPříjmení\n");

            // zpracuj výslednou sadu
            while(reader.Read())
            {
               Console.WriteLine("{0} | {1}",
```

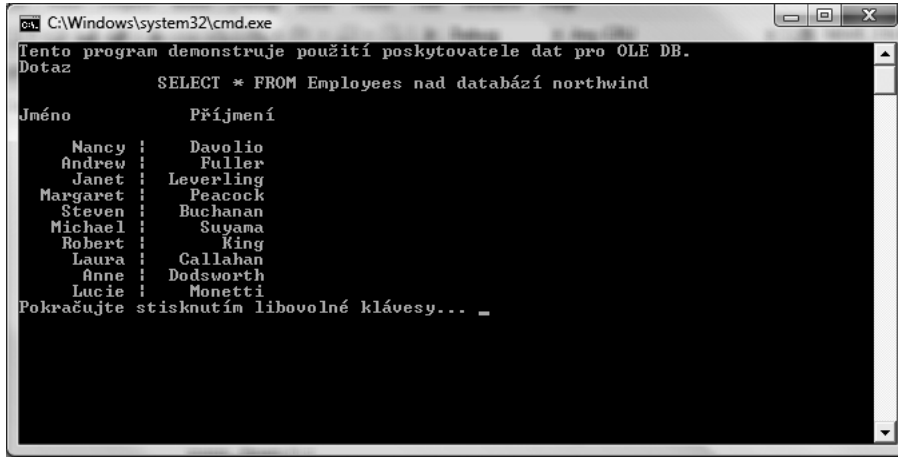
```
                reader["FirstName"].ToString().PadLeft(10),
                reader[1].ToString().PadLeft(10));
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("Chyba: " + e);
        }
        finally
        {
            // uzavři připojení
            reader.Close();
            conn.Close();
        }
    }
  }
}
```

2. Protože ve svém řešení máte nyní dva projekty, musíte učinit tento projekt hlavním projektem, aby se po stisku kláves Ctrl+F5 spustil právě on. Klepněte v okně Solution Explorer pravým tlačítkem na název projektu a poté zvolte příkaz Set as StartUp Project (viz obrázek 9.4).



Obrázek 9.4: Nastavení hlavního projektu

3. Stiskem kláves Ctrl+F5 spusťte aplikaci. Měl by se objevit výsledek jako na obrázku 9.5.



Obrázek 9.5: Přístup k databázi Northwind přes poskytovatele dat pro OLE DB

Jak to funguje

Tento program dělá úplně to samé jako první příklad, takže se podíváme jen na to, co se změnilo. Nejdříve ve třetí direktivě `using` nahradíte `SqlClient` oborem názvů `OleDb`:

```
using System;
using System.Data;
using System.Data.OleDb;
```

Připojovací řetězec vyžaduje nejvíce změn, protože poskytovatel dat pro OLE DB nepřijímá stejné parametry jako poskytovatel dat pro SQL Server. Kromě toho vyžaduje parametr s názvem `provider`.

```
// připrav připojovací řetězec
string connString = @"
 provider = sqloledb;
 data source = .\sqlexpress;
 integrated security = sspi;
 initial catalog = northwind";
```

Pro použití tříd poskytovatele dat pro OLE DB pro připojení, příkaz a objekt pro čtení dat je nutné změnit jen čtyři další řádky:

```
// deklaruj proměnné pro připojení a čtení dat
OleDbConnection conn = null;
OleDbDataReader reader = null;

try
{
```

```
// otevři připojení
conn = new OleDbConnection(connString);
conn.Open();

// proveď dotaz
OleDbCommand cmd = new OleDbCommand(sql, conn);
reader = cmd.ExecuteReader();
```

Poslední změna je sémantického rázu a nemá nic společného s knihovnou ADO.NET:

```
// zobraz záhlaví výstupu
Console.WriteLine("Tento program demonstruje "
 + "použití poskytovatele dat pro OLE DB.");
```

Práce s poskytovatelem dat pro ODBC

ODBC je původní, všestranně použitelná technologie společnosti Microsoft pro přístup k datům. Stále se široce používá pro datové zdroje, které nemají poskytovatele pro OLE DB nebo poskytovatele dat v rozhraní .NET Framework. Knihovna ADO.NET obsahuje poskytovatele dat pro ODBC v oboru názvů `System.Data.Odbc`.

Architektura knihovny ODBC představuje v podstatě třívrstvý proces. Aplikace předkládají databázi požadavky pomocí funkcí knihovny ODBC, která převádí jejich volání na protokol (*call-level interface* neboli *rozhraní na úrovni volání*) *ovladače* specifického pro daný datový zdroj. Ovladač komunikuje s datovým zdrojem a předává všechny výsledky a chyby zpět knihovně ODBC. To je zřejmě méně efektivní než přímá komunikace s databází poskytovatele dat specifického pro tuto databázi, takže s ohledem na výkon je vhodnější se poskytovateli dat pro knihovnu ODBC raději vyhnout, poněvadž i přes to, že nabízí pouze jednodušší rozhraní ke knihovně ODBC, zahrnuje veškerou zátěž spojenou s technologií ODBC. Tabulka 9.5 nabízí popis několika důležitých tříd v oboru názvů `Odbc`.

Tabulka 9.5: Běžně používané třídy v oboru názvů Odbc

Třída	Popis
`OdbcCommand`	Provádí dotazy, příkazy a uložené procedury jazyka SQL.
`OdbcConnection`	Představuje připojení k datovému zdroji ODBC.
`OdbcDataAdapter`	Představuje most mezi datovou sadou a datovým zdrojem.
`OdbcDataReader`	Poskytuje přístup k datovému proudu s výsledky, které lze pouze číst a mezi kterými se lze pohybovat pouze vpřed.
`OdbcError`	Uchovává informace o chybách a upozorněních vrácených datovým zdrojem.
`OdbcParameter`	Představuje parametr příkazu.
`OdbcTransaction`	Představuje transakci SQL.

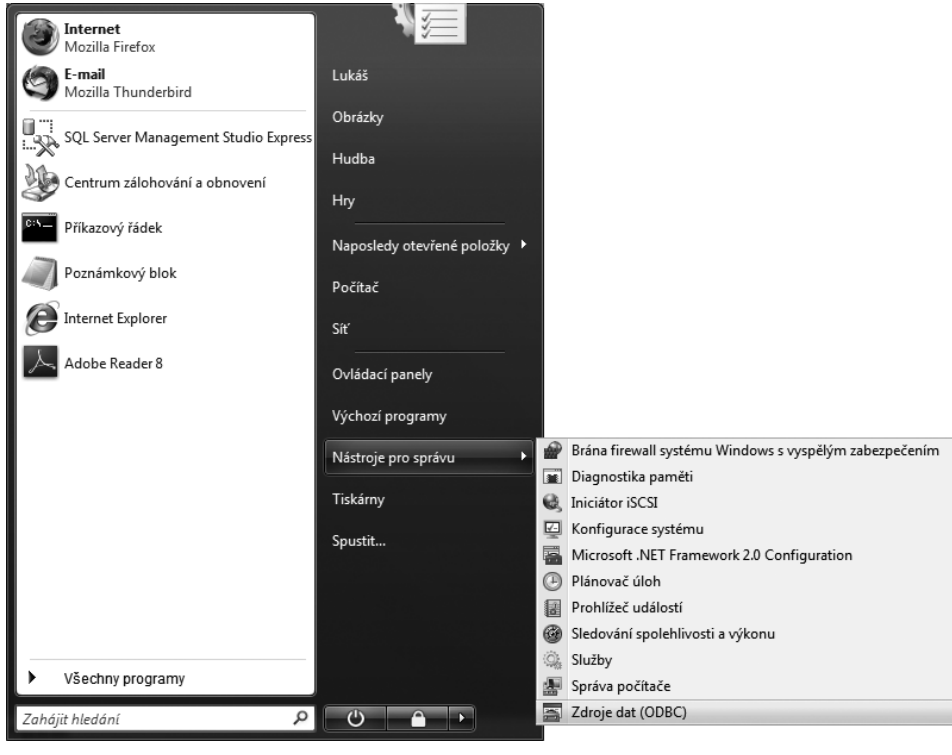
Nyní si vyzkoušíte použití poskytovatele dat knihovny ODBC pro přístup k databázi Northwind, což bude spočívat v provedení stejně jednoduchých změn (tučně zvýrazněných v níže uvedeném výpisu 9.3) v kódu výpisu 9.1, které jste provedli při použití poskytovatele dat pro knihovnu OLE DB.

Ovšem ještě předtím, než začnete, musíte vytvořit datový zdroj pro ODBC (přesněji nakonfigurovat název DSN – Data Source Name – pro použití s datovým zdrojem přístupným v rámci ODBC) pro databázi Northwind, protože na rozdíl od poskytovatelů dat pro SQL Server a OLE DB, neumožňuje poskytovatel dat pro ODBC stanovení serveru nebo databáze v připojovacím řetězci. (Následující postup je určen pro systém Windows Vista, přičemž u ostatních verzí systému Windows funguje podobně.)

Vytvoření datového zdroje pro ODBC

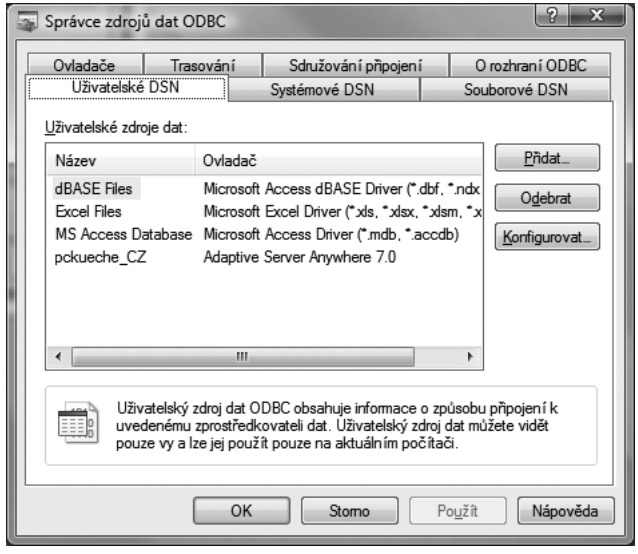
K vytvoření datového zdroje pro ODBC postupujte podle následujících kroků:

1. V hlavní nabídce systému Windows klepněte na položku Nástroje pro správu a spusťte nástroj Zdroje dat (ODBC) (viz obrázek 9.6).



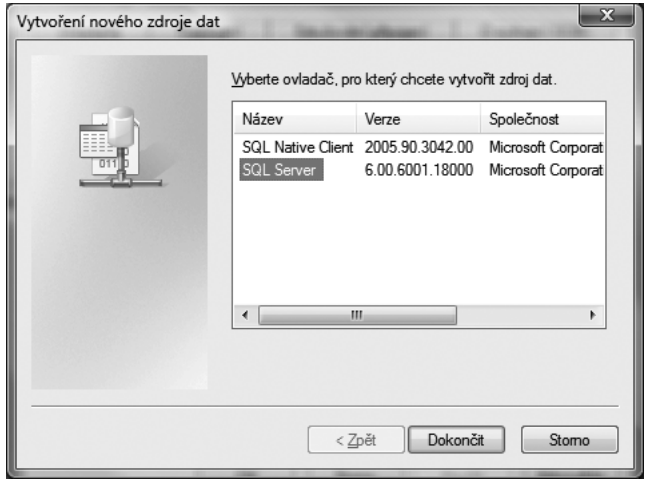
Obrázek 9.6: Nástroje pro správu: Zdroje dat (ODBC)

2. Jakmile se otevře okno Správce zdrojů dat ODBC, klepněte na záložku Uživatelské DSN a poté klepněte na tlačítko Přidat (viz obrázek 9.7).



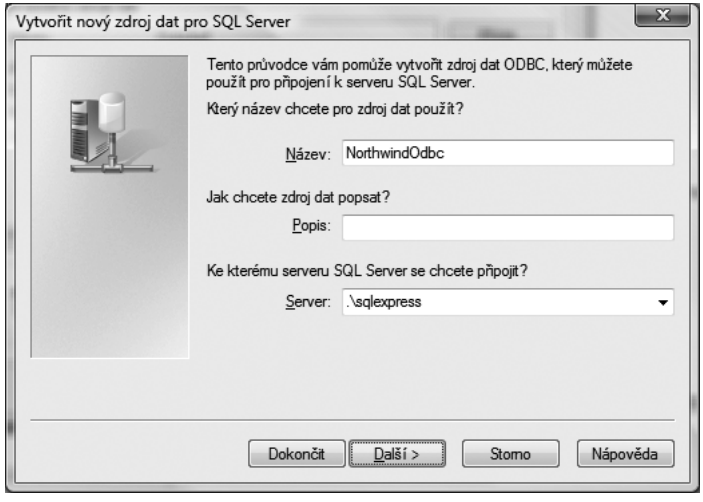
Obrázek 9.7: Dialogové okno Správce zdrojů dat ODBC

3. Spustí se průvodce Vytvoření nového zdroje dat. Postupujte pozorně podle jeho instrukcí! Nejdříve vyberte ovladač SQL Serveru a pak klepněte na tlačítko Dokončit (viz obrázek 9.9).



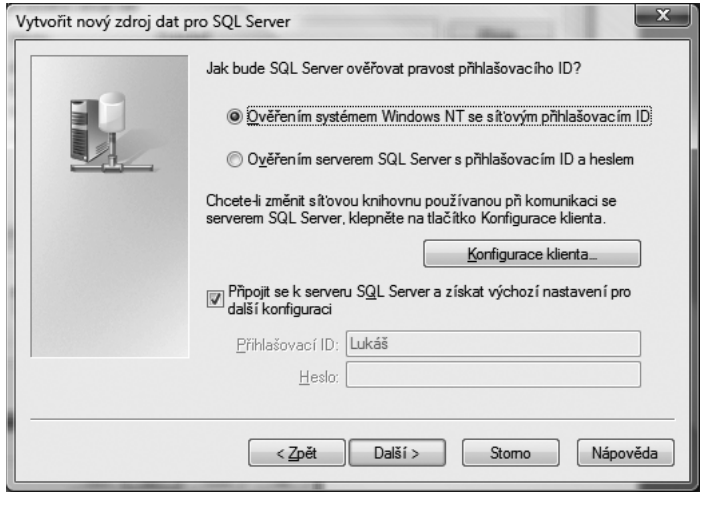
Obrázek 9.8: Průvodce Vytvoření nového zdroje dat

4. Další okno vás vyzývá k uvedení názvu datového zdroje a serveru. Do pole Název napište „NorthwindOdbc“ a do pole Server zadejte „.\sqlexpress“ (viz obrázek 9.9) a poté klepněte na tlačítko Další.



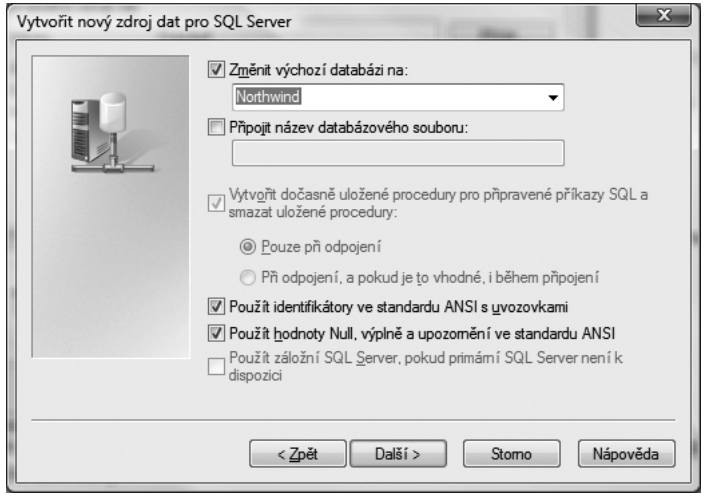
Obrázek 9.9: Stanovení názvu datového zdroje a SQL Serveru, k němuž se chceme připojit

5. V dalším okně přijměte klepnutím na tlačítko Další výchozí nastavení (viz obrázek 9.10).



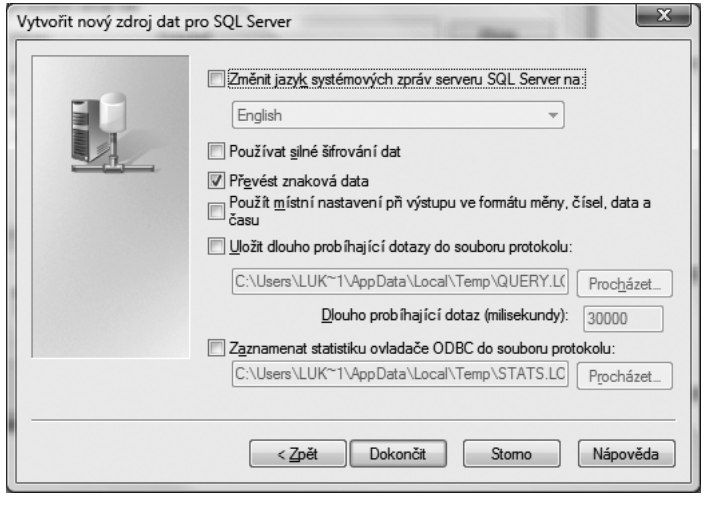
Obrázek 9.10: Stanovení autentizace pro připojení k SQL Serveru

6. V dalším okně zvolte volbu Změnit výchozí databázi na, z rozbalovacího seznamu vyberte databázi Northwind a klepněte na tlačítko Další (viz obrázek 9.11).



Obrázek 9.11: Stanovení výchozí databáze

7. V následujícím okně jednoduše klepněte na tlačítko Dokončit (viz obrázek 9.12).

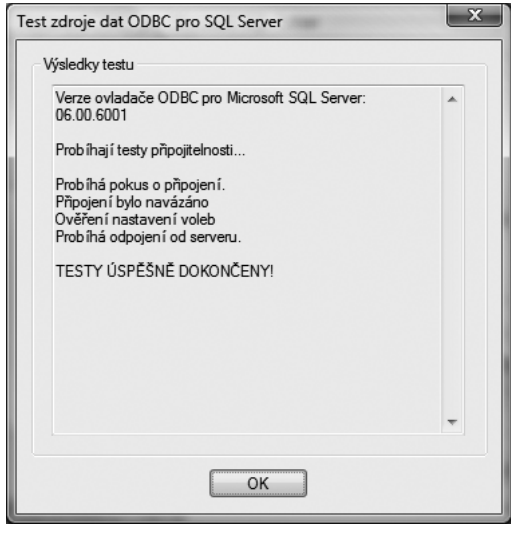


Obrázek 9.12: Dokončení tvorby názvu DSN

8. Objeví se okno s potvrzením popisující nový datový zdroj. Klepněte na tlačítko Test zdroje dat (viz obrázek 9.13).

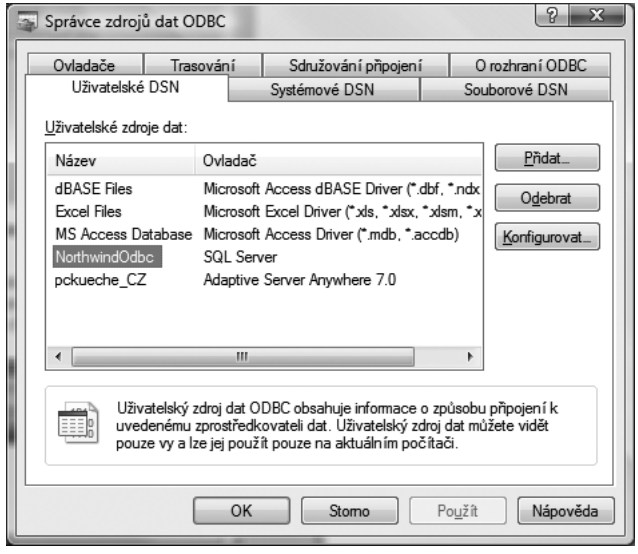
Obrázek 9.13: Testování připojení k datovému zdroji NorthwindOdbc

9. Mělo by se objevit okno oznamující úspěšné provedení testu (viz obrázek 9.14). (Pokud se neobjeví, svou práci anulujte a zkuste vše ještě jednou.) Klepněte na tlačítko OK.



Obrázek 9.14: Připojení k datovému zdroji bylo úspěšné

10. V okně s potvrzením klepněte znovu na tlačítko OK. Jakmile se objeví okno Správce zdrojů dat ODBC, bude v seznamu uveden nově vytvořený datový zdroj (viz obrázek 9.15). Klepněte na tlačítko OK.



Obrázek 9.15: V seznamu datových zdrojů se objeví nový datový zdroj

Nyní máte svůj datový zdroj NorthwindOdbc připravený k práci. V následujícím příkladu jej použijete při tvorbě připojovacího řetězce.

Vyzkoušejte: Vytvoření jednoduché konzolové aplikace pomocí poskytovatele dat pro ODBC

Vyzkoušejte si přístup k databázi Northwind přes ODBC:

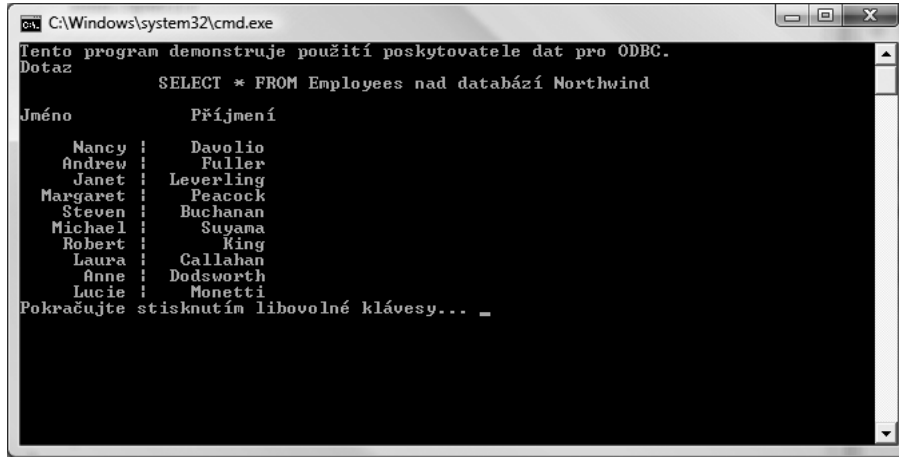
1. V okně Solution Explorer přidejte do řešení Chapter09 nový projekt typu Console Application s názvem „OdbcProvider". Přejmenujte soubor *Program.cs* na *OdbcProvider.cs*. V okně editoru nahraďte vygenerovaný kód kódem ve výpisu 9.3, v němž jsou tučně vyznačeny změny oproti výpisu 9.1.

Výpis 9.3: Soubor OdbcProvider.cs

```
using System;
using System.Data;
using System.Data.Odbc;

namespace Chapter09
{
   class OdbcProvider
   {
      static void Main(string[] args)
      {
         // připrav připojovací řetězec
         string connString = @"dsn=northwindodbc";
```

```
            // připrav řetězec s dotazem
            string sql = @"
              SELECT * FROM Employees";

            // deklaruj proměnné pro připojení a čtení dat
            OdbcConnection conn = null;
            OdbcDataReader reader = null;

            try
            {
               // otevři připojení
               conn = new OdbcConnection(connString);
               conn.Open();

               // proveď dotaz
               OdbcCommand cmd = new OdbcCommand(sql, conn);
               reader = cmd.ExecuteReader();

                // zobraz záhlaví výstupu
                Console.WriteLine("Tento program demonstruje "
                 + "použití poskytovatele dat pro ODBC.");
                Console.WriteLine("Dotaz {1} nad databází {0}\n",
                 conn.Database, cmd.CommandText);
                Console.WriteLine("Jméno\t\tPříjmení\n");

                // zpracuj výslednou sadu
                while(reader.Read())
                {
                   Console.WriteLine("{0} | {1}",
                    reader["FirstName"].ToString().PadLeft(10),
                    reader[1].ToString().PadLeft(10));
                }
            }
            catch (Exception e)
            {
               Console.WriteLine("Chyba: " + e);
            }
            finally
            {
               // uzavři připojení
               reader.Close();
               conn.Close();
            }
         }
      }
   }
```

2. Nastavte tento projekt jako hlavní, což provedete tak, že v okně Solution Explorer klepnete pravým tlačítkem na název projektu a poté zvolíte příkaz Set as StartUp Project (viz dříve uvedený obrázek 9.4).
3. Stiskem kláves Ctrl+F5 spusťte aplikaci. Měl by se objevit výsledek jako na obrázku 9.16.



Obrázek 9.16: Přístup k databázi Northwind přes ODBC

Jak to funguje

Jakmile vytvoříte název DSN, tak už je to hračka. V názvech tříd (a samozřejmě také v záhlaví výstupu) jednoduše změníte `Sql` na `Odbc` úplně stejně jako při úpravě programu pro poskytovatele OLE DB. Největší a jediná změna, která si opravdu zaslouží pozornost, se týká připojovacího řetězce:

```
// připrav připojovací řetězec
string connString = @"dsn=northwindodbc";
```

Připojovací řetězec pro ODBC není omezen pouze na název DSN, v řetězci však není možné použít mezery ani nové řádky.

Tip: Každý poskytovatel dat má svá vlastní pravidla týkající se jak parametrů, tak i syntaxe svého připojovacího řetězce. Při programování připojovacích řetězců vždy nahlédněte do dokumentace pro poskytovatele, kterého používáte. Připojovací řetězce mohou být velmi komplikované. Podrobnostem se zde sice věnovat nebudeme, ale dokumentace pro připojovací řetězce je součástí popisu vlastnosti `ConnectionString` třídy pro připojení u každého poskytovatele dat.

Nyní, když jste si vyzkoušeli práci se všemi poskytovateli dat umožňujícími přístup k SQL Serveru (popis poskytovatele dat SQL Server CE je nad rámec této knihy), se ještě ujistěte, že skutečně rozumíte tomu, co poskytovatel dat vlastně je a jak lze přistupovat k datům pomocí různých poskytovatelů dat.

Poskytovatelé dat jsou aplikační rozhraní

Poskytovatelé dat z rozhraní .NET Framework jsou ve své sofistikovanosti (o které se více dozvíte v dalších kapitolách) prostá aplikační rozhraní pro přístup k datovým zdrojům, což jsou nejčastěji relační databáze. (Knihovna ADO.NET je v podstatě jedno velké aplikační rozhraní, jehož velkou část tvoří právě poskytovatelé dat.)

Nováčci ve světe ADO.NET jsou často z pochopitelného důvodu z dokumentace společnosti Microsoft zmateni. Čtou totiž o objektech `Connection`, `Command`, `DataReader` a dalších, ale v žádném oboru názvů knihovny ADO.NET nevidí žádné třídy pojmenované `Connection`, `Command` nebo `DataReader`. To je dáno tím, že třídy poskytovatelů dat implementují rozhraní z oboru názvů `System.Data`. Toto rozhraní definuje metody poskytovatele dat v rámci aplikačního rozhraní knihovny ADO.NET.

Klíčová koncepce je velice jednoduchá. Poskytovatel dat, jako je kupříkladu `System.Data.SqlClient`, se skládá ze tříd, jejichž metody poskytují jednotný způsob přístupu k určitému druhu datového zdroje. V této kapitole jste používali tři poskytovatele dat (SQL Server, OLE DB a ODBC) pro přístup k téže databázi. Jedinou skutečnou odlišností v kódu byl připojovací řetězec. Až na výběr vhodného poskytovatele dat je zbytek programování vesměs naprosto stejný. A to platí pro všechny služby knihovny ADO.NET bez ohledu na to, k jakému druhu datového zdroje přistupujete.

Poskytovatel dat pro SQL Server je optimalizován pro přístup k SQL Serveru a nelze jej použít pro žádný jiný databázový systém. Poskytovatel dat pro OLE DB může přistupovat ke kterémukoliv datovému zdroji OLE DB – použili jste jej, aniž byste věděli cokoliv o OLE DB (což by samo o sobě vydalo na pořádnou studii). Díky poskytovateli dat pro ODBC můžete používat dokonce ještě starší technologii pro přístup k datům, a opět aniž byste o ní cokoliv věděli. Práce na takto abstraktní úrovni vám umožňuje v ještě kratším čase zvládnout větší kus práce.

Knihovna ADO.NET je nejen efektivní technologií pro přístup k datům, ale také elegantní. Poskytovatelé dat jsou jen jedním z jejích aspektů. Umění programování v knihovně ADO.NET je založeno spíše na správném chápání pojmů než na klasickém programování. Nejdříve je nutné získat jasnou představu o tom, co knihovna ADO.NET nabízí, a pak tuto představu pomocí správné metody ve správné třídě realizovat.

Vzhledem k tomu, že na prvním místě stojí správné pochopení pojmů, můžete na připojení, příkazy, objekty pro čtení dat a další komponenty knihovny ADO.NET nahlížet (a odkazovat se na ně) především jako na abstrakce a ne jako na pouhé objekty používané při programování databází. Pokud se soustředíte na pojmy, bude samotné osvojení toho, kdy a jak příslušné objekty a metody použít, již velice snadné.

Shrnutí

V této kapitole jste viděli, proč byla knihovna ADO.NET vyvinuta a jak nahrazuje jiné technologie platformy .NET pro přístup k datům. Seznámili jste se s její architekturou a poté jste se soustředili na jednu z jejích hlavních komponent, poskytovatele dat. Sestavili jste tři ukázkové příklady sloužící k procvičení základního používání poskytovatelů dat a vyzkoušeli jste si jednotný způsob zápisu

kódu pro přístup k datům, který je nezávislý na konkrétním poskytovateli dat. Nakonec jste se seznámili s názorem, že srozumitelnost pojmů je klíčem k pochopení a používání jak poskytovatelů dat, tak i zbývajícího aplikačního rozhraní knihovny ADO.NET.

Dále se ponoříme do studia detailů knihovny ADO.NET, přičemž začneme připojeními.