SQL SERVER: ADO.NET

Softwarové inžinierstvo

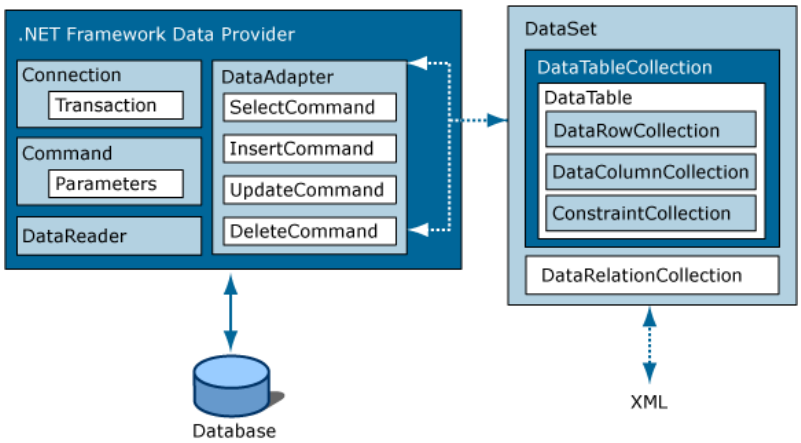
MARTIN TIMOTHY TIMKO

14.9. 2017

1 ÚVOD

ADO.NET je objektovo-orientovaná množina knižníc, ktorá poskytuje manipuláciu s dátovými zdrojmi. Štandardne sa považuje za dátový zdroj nejaká databáza, ale môže to byť aj nejaký obyčajný text, dokument programu *Microsoft Excel* alebo XML súbor.

.NET framework obsahuje v dnešnej dobe už množstvo rôznych knižníc, ktoré poskytujú rôzne služby, napr. v oblasti používateľského rozhrania ponúka .NET framework knižnice WinForms v oblasti desktopového vývoja, *ASP.NET* v oblasti webového vývoja, alebo najnovší, okrem iného, aj desktopový vývoj v podobe *WPF* (Windows Presentation Foundation), ktorý postupne nahrádza klasické *WinForms* (Windows Forms, ktorý nahradil *MFC* (Microsoft Foundation Class)), ďalej obsahuje tento framework aj knižnice v oblasti serverovej komunikácie v podobe *WCF* (Windows Communication Foundation) a neposlednom rade práve knižnice na prístup k dátam, kde patrí ADO.NET, objektové databázy typu *Entity Framework*, *NHibernate* a efektívnejší prístup k selektovaniu dát vo forme *LINQ* (Language Integrated Query).



Obrázok 1: Štruktúra ADO.NET

Samotná knižnica ADO.NET ponúka podľa diagramu nástroje a funkcie na prácu s dátami primárne neobjektového typu, pokiaľ sa nedochádza v interakcii s nejakou ďalšou objektovou databázou. Takže v zásade sa jedná o klasický prístup k databáze známy z prvopočiatkov databázového programovania, ktorý bude aj obsahom tohto článku.

V dnešnej dobe existuje niekoľko typov databáz, napr. *Microsoft SQL Server*, *Microsoft Access*, *Oracle*, *Borland Interbase* alebo *IBM DB2*. V každom prípade manipulácia s dátami je vo všetkých typoch databáz veľmi podobná. V tomto článku sa budeme zaoberať najrozšírenejším typom databázy **Microsoft SQL Server**.

Názov dátového poskytovateľa	API prefix	Popis dátového zdroja
ODBC	Odbc	Dátový zdroj s ODBC rozhraním.
OleDb	OleDb	Dátové zdroje, ktoré vystavujú OleDb rozhranie.
Oracle	Oracle	Oracle databázy.
SQL	Sql	Manipulácia s Microsoft SQL Server.
Borland	Bdp	Generický prístup k viacerým typom databáz.

2 POSKYTOVATELIA DÁT

Ako bolo spomenuté, existuje niekoľko typov databáz v závislosti od typu dát. Niektoré staršie dátové zdroje používajú ODBC protokol, niekoľko novších dátových zdrojov používa OleDb a najnovšie dátové zdroje komunikujú prostredníctvom .NET knižnice ADO.NET. Všetky tieto knižnice sa nazývajú ako tzv. poskytovatelia dát (Data Providers) a sú pomenované podľa dátového zdroja, ktorý ich reprezentuje, čo je možné vidieť v nasledujúcej tabuľke.

Ak by sme chceli napr. použiť poskytovateľa dát `OleDb` na pripojenie dátového zdroja, ktorý používa OleDb rozhranie, mohli by sme použiť pripojovací objekt s `OleDbConnection`. Podobne by to bolo aj s ostatnými dátovými poskytovateľmi. V našom prípade budeme používať poskytovateľa dát `SQL` na pripojenie dátového zdroja, ktorý používa SQL rozhranie s pripojovacím objektom `SqlConnection`.

3 INŠTALÁCIA DATABÁZY

Predtým než vôbec začneme pracovať s databázou, bude potrebné najskôr takúto SQL databázu vytvoriť. Principiálne nemusíme použiť vývojové prostredie *Visual Studio*, avšak pri programovaní s knižnicami .NET je veľkou výhodou. Teda, môžme použiť aj iné vývojové prostredia, napr. *NetBeans* alebo *SharpDevelop*, prípadne v prostredí Linux/UNIX aj *MonoDevelop*. V našom prípade použijeme vývojové prostredie Visual Studio verzie 2015.

Za týmto účelom je potrebné nainštalovať *Microsoft SQL Server Management Studio*. Následne je potrebné nainštalovať samotný *SQL Server*, v základnom nastavení. Po inštalácii je možné skopírovať tzv. pripojovací reťazec (connection string), ktorý neskôr použijeme pri pripájaní na databázu. Avšak nie je nutné tento reťazec uchovávať, pretože je možné tento pripojovací reťazec získať aj iným spôsobom. Po kompletnej inštalácii spustíme samotný program *Microsoft SQL Server Management Studio* a pri spustení uvidíme názov servera na ktorý sa môžme pripojiť. Samozrejme, názov servera je možné zmeniť, ale kvôli jednoduchosti to necháme v pôvodnom stave. Po pripojení na server je možné vytvoriť novú databázu. Po vytvorení databázy môžme vytvárať nové tabuľky, spúšťače, uložené procedúry a ostatné náležitosti, avšak je možné ich vytvárať aj priamo vo vývojovom prostredí Visual Studio. A to bude aj náš prípad.

Takže týmto postupom sme nainštalovali SQL Server a vytvorili zatiaľ prázdnu databázu.

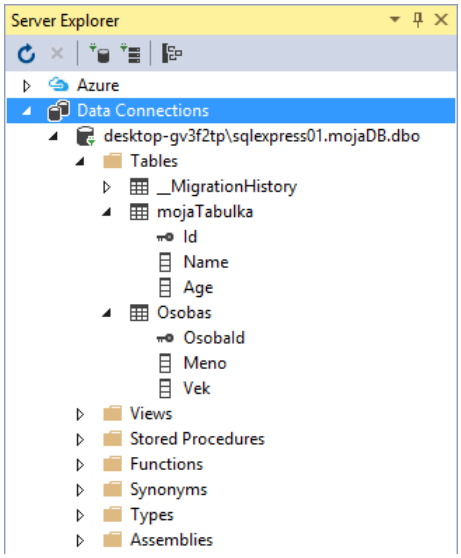
4 OBJEKTY ADO.NET

Ako už bolo spomínané, ADO.NET ponúka viac možností pre prácu s dátovými zdrojmi, ale v našom prípade sa budeme zaoberať len jedným typom a teda SQL prístupom. V nasledujúcich častiach budú postupne prezentované jednotlivé časti vývoja databázovej aplikácie.

4.1 Objekt SqlConnection

Ak chceme nejakým spôsobom komunikovať s databázou je potrebné vytvoriť pripojenie. Takéto pripojenie identifikuje databázový server, databázové meno, používateľské meno a heslo a prípadne ďalšie náležitosti súvisiace s databázou.

Za týmto účelom potrebujeme vytvoriť nejaký dátový zdroj, v našom prípade ako už bolo spomínané sa zameráme na SQL Server. Takže vo vývojom prostredí *Visual Studio* v *Server Explorer* pridáme nový dátový zdroj.



Obrázok 2: Vytvorenie dátového zdroja

Po tomto databázovom nastavení môžme vytvoriť vo *Visual Studio* prázdnu konzolovú aplikáciu s nasledujúcim kódom:

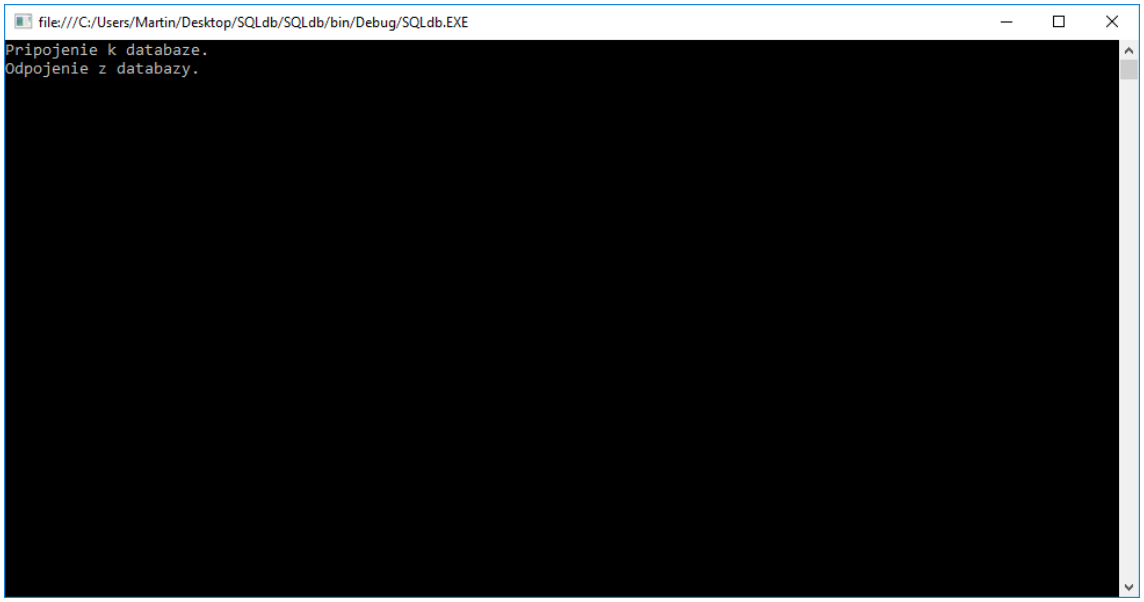
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace SQLdb
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            string connString = @"Server=localhost\SQLEXPRESS01;Database=mojaDB;
                                                 Trusted_Connection=True;";

            SqlConnection conn = new SqlConnection(connString);

            try
            {
                conn.Open();
                Console.WriteLine("Pripojenie k databaze.");
            }
            catch (SqlException e)
            {
                Console.WriteLine("Chyba: " + e);
            }
            finally
            {
                conn.Close();
                Console.WriteLine("Odpojenie z databazy.");
            }

            Console.ReadKey();
        }
    }
}
```

Ak chceme písať databázové programy, tak predtým než začneme je potrebné pridať odkazy pre manipuláciu s dátovými zdrojmi:

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

Pripojovací reťazec sme uviedli na začiatku po inštalácii *SQL Server*, teda konkrétne: `Serverl̄ocalhost SQLEXPRESS01 DatabasemojaDB Trusted_ConnectionTrue`. V tomto duchu môžme danú aplikáciu spustiť.



Obrázok 3: Pripojenie a odpojenie z databázy

Poznámka: Na konci zdrojového kódu sme napísali `Console.ReadKey();` *z dôvodu zastavenia toku programu, aby sme sprístupnili obsah dialógového okna k možnému čítaniu.*

Takže týmto postupom sme vytvorili pripojenie na databázový server k databáze.

4.2 Objekt Sqlcommand

Proces v interakcii s databázou znamená, že potrebujeme špecifikovať akciu, ktorá vykoná túto udalosť. Práve na tento účel slúžia príkazy, ktoré potrebujeme napísať a poslať do databázy. Každý príkazový objekt definovaný triedou `Sqlcommand` používa pripojovací objekt na danú databázu. Môžme použiť samostatný príkazový objekt, spustiť príkaz priamo alebo pripojiť odkazom na príkazový objekt prostredníctvom triedy `SqlDataAdapter`, ktorý obsahuje nejakú množinu príkazov pracujúcich na nejakej skupine dát.

Skôr než začneme písať príkazy je potrebné vložiť do tabuľky nejaké dáta. Takže vytvoríme novú jednoduchú tabuľku.

	Id	Name	Age
▶	0	Peter	17
	1	Fero	28
	2	Vlado	47
	3	Jozef	35
	100	Stano	59
*	*NULL*	*NULL*	*NULL*

Obrázok 4: Tabuľka s vytvorenými dátami

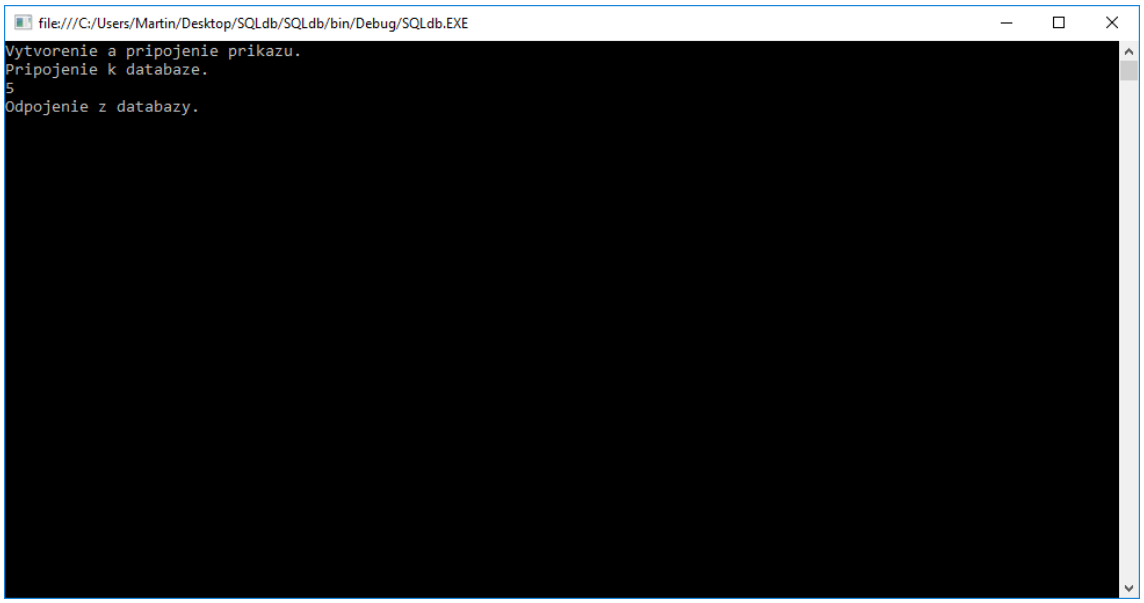
Vložíme do nej nejaké náhodné testovacie dáta a napíšeme nasledujúci kód:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace SQLdb
{
    class Program
    {
        static void Main(string[] args)
        {
            string connString = @"Server=localhost\SQLEXPRESS01;Database=mojaDB;
                                                     Trusted_Connection=True;";
            SqlConnection conn = new SqlConnection(connString);

            string sql = @"SELECT COUNT(*) FROM mojaTabulka";

            SqlCommand cmd = new SqlCommand(sql, conn);
            Console.WriteLine("Vytvorenie a pripojenie prikazu.");

            try
            {
                conn.Open();
                Console.WriteLine("Pripojenie k databaze.");

                //spustenie prikazu
                Console.WriteLine("{0}", cmd.ExecuteScalar());
            }
            catch (SqlException e)
            {
                Console.WriteLine("Chyba: " + e);
            }
            finally
            {
                conn.Close();
                Console.WriteLine("Odpojenie z databazy.");
            }

            Console.ReadKey();
        }
    }
}
```

Použili sme SQL príkaz v podobe:

```
SELECT COUNT(*) FROM mojaTabulka
```

Tento SQL príkaz zistí pomocou funkcie `COUNT()` počet prvkov v celej tabuľke `mojaTabulka`.



Obrázok 5: Počet prvkov v tabuľke

Ako sme si mohli všimnúť, použili sme príkazovú funkciu `ExecuteScalar()`, ktorá ako už z názvu vyplýva je schopná zobraziť nejakú skalárnu hodnotu, teda v našom prípade to bude reprezentovať počet všetkých záznamov v tabuľke. Nie je možné použiť túto funkciu pre klasický výpis riadkov a stĺpcov tabuľky. Práve z toho dôvodu existuje funkcia `ExecuteReader()`, ktorá umožňuje výpis po jednotlivých riadkoch tabuľky.

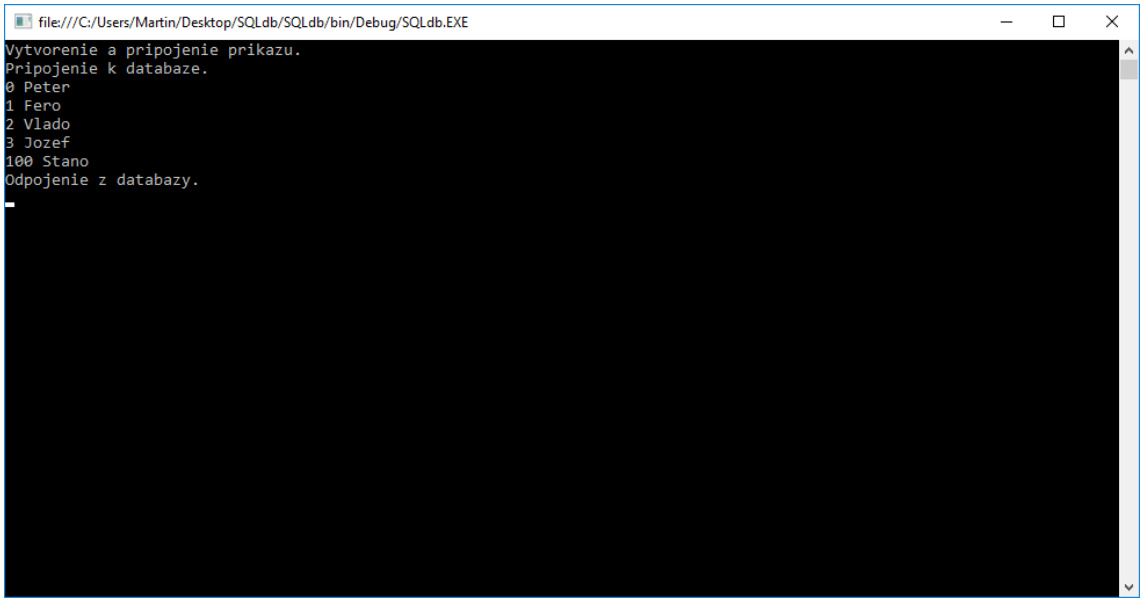
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace SQLdb
{
    class Program
    {
        static void Main(string[] args)
        {
            string connString = @"Server=localhost\SQLEXPRESS01;Database=mojaDB;
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Trusted_Connection=True;";
            SqlConnection conn = new SqlConnection(connString);

            string sql = @"SELECT␣*␣FROM␣mojaTabulka";

            SqlCommand cmd = new SqlCommand(sql, conn);
            Console.WriteLine("Vytvorenie␣a␣pripojenie␣prikazu.");

            try
            {
                conn.Open();
                Console.WriteLine("Pripojenie␣k␣databaze.");

                SqlDataReader dr = cmd.ExecuteReader();

                while(dr.Read())
                {
                    Console.WriteLine("{0}␣{1}", dr.GetValue(0), dr.GetValue(1));
                }
            }
            catch (SqlException e)
            {
                Console.WriteLine("Chyba:␣" + e);
            }
            finally
            {
                conn.Close();
                Console.WriteLine("Odpojenie␣z␣databazy.");
            }

            Console.ReadKey();
        }
    }
}
```

Použili sme SQL príkaz v podobe:

```
SELECT * FROM mojaTabulka
```

To znamená, že budeme požadovať všetky stĺpce tabuľky, ktoré budú následne obsahom príkazovej funkcie `ExecuteReader()`. Týmto obdržíme jednotlivé riadky tabuľky.



Obrázok 6: Výpis niektorých prvkov tabuľky v stĺpcoch

Čo sa týka jednotlivých SQL príkazov, tak je možné použiť v rámci pravidiel SQL jazyka akékoľvek príkazy, napr.:

```
SELECT * FROM auto WHERE celk_suma > 0;
SELECT * FROM zam ORDER BY priezvisko;
SELECT meno, funkcia FROM pracovnici WHERE id IN(1,3); // <-- id = <1 až 3>
INSERT INTO auto VALUES('KE0707', '0', 123);
UPDATE auto SET spz = 'BA0707', celk_suma = 123 WHERE dt = '2.3.1988';
DELETE FROM auto WHERE spz = 'KE0707';
```

Alebo komplikovanejšie príkazy:

```
SELECT pracovnici.meno, pracovnici.funkcia
FROM pracovnici, pobocky WHERE pracovnici.id_pobocky = pobocky.id_pobocky;

SELECT pr.meno, pr.funkcia
FROM pracovnici pr, pobocky po WHERE pr.id_pobocky = po.id_pobocky(+);

SELECT id_pobocky, sum(mzda)
FROM pracovnici WHERE id_pobocky IS NOT NULL GROUP BY id_pobocky
HAVING sum(mzda) >= 45000 ORDER BY sum(mzda) ASC;

SELECT nazov FROM project WHERE id_p
IN (SELECT id_p FROM project_zam GROUP BY id_p HAVING count(*) = 2 OR COUNT(*) = 3;
```

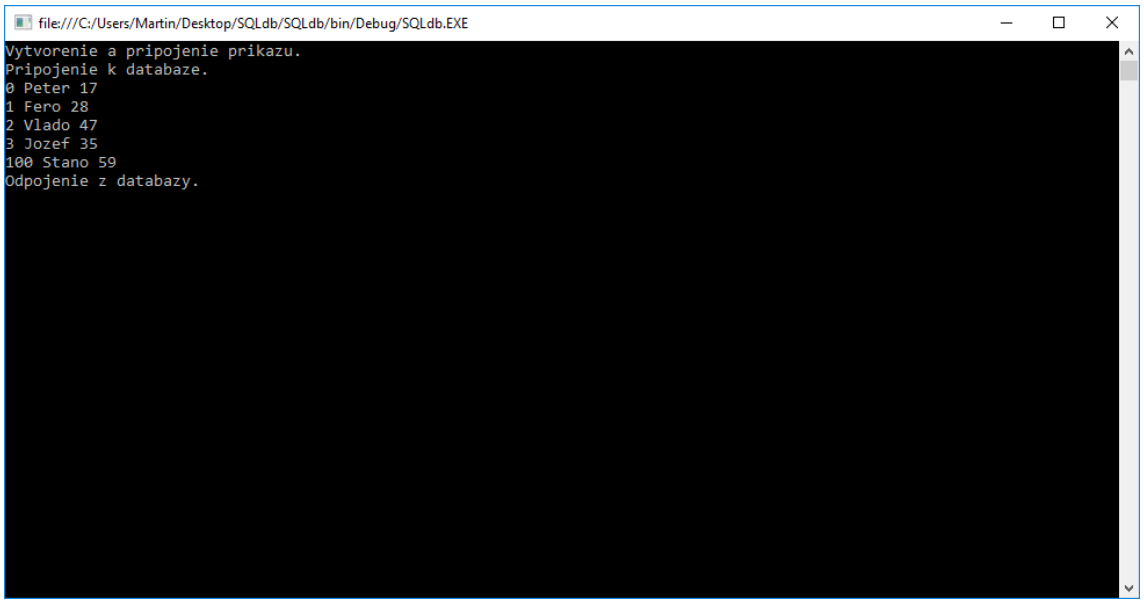
Niektoré ďalšie je možné preštudovať v publikácii *Softwarové inžiniestvo* v časti *Databázový jazyk SQL*.

4.3 Objekt SqlDataAdapter a DataSet

`DataSet` objekt je pamäťová reprezentácia dát, ktorá obsahuje rôznorodé `Datatable` objekty pozostávajúce z riadkov a stĺpcov. DataSet je špecifický v návrhu dát, ktorý umožňuje spravovať dáta v pamäti. Tento typ objektu je prístupný všetkým dátovým poskytovateľom. `SqlDataAdapter` umožňuje vykonávať SQL príkazy na danom `Datasete`.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace SQLdb
{
    class Program
    {
        static void Main(string[] args)
        {
            string connString = @"Server=localhost\SQLEXPRESS01;Database=mojaDB;
                                                Trusted_Connection=True;";
            SqlConnection conn = new SqlConnection(connString);

            string sql = @"SELECT * FROM mojaTabulka";

            SqlCommand cmd = new SqlCommand(sql, conn);
            Console.WriteLine("Vytvorenie a pripojenie prikazu.");

            try
            {
                conn.Open();
                Console.WriteLine("Pripojenie k databaze.");

                SqlDataAdapter da = new SqlDataAdapter(sql, conn);
                DataSet ds = new DataSet();
                da.Fill(ds, "mojaTabulka");

                DataTable dt = ds.Tables["mojaTabulka"];

                foreach(DataRow row in dt.Rows)
                {
                    foreach(DataColumn col in dt.Columns)
                    {
                        Console.Write("{0} ", row[col]);
                    }
                    Console.WriteLine();
```

```
                }

            }
            catch (SqlException e)
            {
                Console.WriteLine("Chyba: " + e);
            }
            finally
            {
                conn.Close();
                Console.WriteLine("Odpojenie z databazy.");
            }

            Console.ReadKey();
        }
    }
}
```



Obrázok 7: Výpis všetkých prvkov tabuľky v stĺpcoch

Týmto spôsobom môžme manipulovať s dátami v databáze. Problém môže nastať v prípade, ak budeme potrebovať nejaké dáta pridávať alebo odstraňovať z databázy. V tomto type programovania, teda v konzolovom rozhraní to nie je až také nebezpečné. V iných typoch aplikácií, napr. WinForms, kde existuje nejaká komponenta pomocou ktorej je možné zadávať vstup od používateľa pre prípadný zásah do databázy, tak v tom prípade hrozí určité nebezepečenstvo, ktoré sa nazýva *SQL injection*.

Jedná sa o prípad, kedy do nejakého zadávacieho poľa, aj vyhľadávacieho, zadáme text, ktorý nebude korešpondovať s priamou požiadavkou o nami zamyšľané dáta. Napr. namiesto klasického príkazu pre výber dát z tabuľky pridáme ešte ďalší príkaz oddelený podkočiarkou.

```
SELECT * FROM userinfo WHERE id=1; DROP TABLE users;
```

V tomto prípade vyhľadáme nejaké potencionálne dáta, ale následne v ďalšom príkaze odstránime tabuľku `users`.

Práve z toho dôvodu je potrebné zapisovať v prípade *CRUD* operácií požiadavky prostredníctvom **parametrov**. Napr. pre vloženie nového záznamu do tabuľky kvôli jednoduchosti s pevnou hodnotu `Id` a s nejakými danými parametrami bude vyzerať kód nasledovne:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace SQLdb
{
    class Program
    {
        static void Main(string[] args)
        {
            string connString = @"Server=localhost\SQLEXPRESS01;Database=mojaDB;
                                                Trusted_Connection=True;";
            SqlConnection conn = new SqlConnection(connString);

            string insName = "Juraj";
            int insAge = 24;

            string sqlins = @"INSERT INTO mojaTabulka(id, name, age)
                                                VALUES(101, @insname, @insAge)";

            SqlCommand cmdnon = new SqlCommand(sqlins, conn);
            Console.WriteLine("Vytvorenie a pripojenie prikazu.");

            cmdnon.Prepare();

            //pridanie parametrov do prikazu
            cmdnon.Parameters.Add("@insName", SqlDbType.NVarChar, 50);
            cmdnon.Parameters.Add("@insAge", SqlDbType.Int, 100);

            try
            {
                conn.Open();
```

```
                Console.WriteLine("Pripojenie k databaze.");

                cmdnon.Parameters["@insName"].Value = insName;
                cmdnon.Parameters["@insAge"].Value = insAge;

                cmdnon.ExecuteNonQuery();

            }
            catch (SqlException e)
            {
                Console.WriteLine("Chyba: " + e);
            }
            finally
            {
                conn.Close();
                Console.WriteLine("Odpojenie z databazy.");
            }

            Console.ReadKey();
        }
    }
}
```

Teda, klasický SQL príkaz `INSERT` nahradíme s parametrami:

```
INSERT INTO mojaTabulka(id, name, age) VALUES(101, @insname, @insAge)
```

Následne získame nový záznam v tabuľke:

	Id	Name	Age
▶	0	Peter	17
	1	Fero	28
	2	Vlado	47
	3	Jozef	35
	100	Stano	59
	101	Juraj	24
*	*NULL*	*NULL*	*NULL*

Obrázok 8: Výpis všetkých prvkov tabuľky v stĺpcoch

Takže takýmto spôsobom sme schopní manipulovať s dátami pomocou dátového objektu `SqlConnection` dátového poskytovateľa SQL.