**Rukovanje podacima u povezanom okruženju**

**00:22:59**

Jedinica: 2 od 19

**Rezime**

* Pre bilo kakve direktne manipulacije podacima, bilo da je reč o čitanju, brisanju, unosu ili ažuriranju, neophodno je postojanje otvorene konekcije sa bazom.
* ConnectionString definiše sve informacije koje su potrebne aplikaciji kako bi pronašla izvor podataka, povezala se na njega i odabrala odgovarajuću bazu sa čijim podacima je potrebno raditi.
* SqlDataReader klasa se koristi za čitanje podataka u povezanom modu.
* SqlDataReader za rad koristi aktivnu konekciju, tako da je potrebno koristiti ga brzo i efikasno.
* Za manipulaciju podacima koriste se objekti klase Command.
* ADO.NET omogućava elegantan način za kreiranje parametrizovanih upita korišćenjem SqlParameter.

Najniži pristup rukovanju podacima korišćenjem ADO.NET-a predstavlja rad sa podacima u povezanom okruženju. Korišćenjem ovog pristupa programer je zadužen za kreiranje SQL naredbi i za njihovo izvršavanje. Pri tome, prilikom izvršavanja SQL naredbi, neophodno je postojanje direktne veze sa bazom podataka. Takva veza se naziva konekcija, a ovaj pristup, upravo zbog toga, konektovani ili povezani pristup.

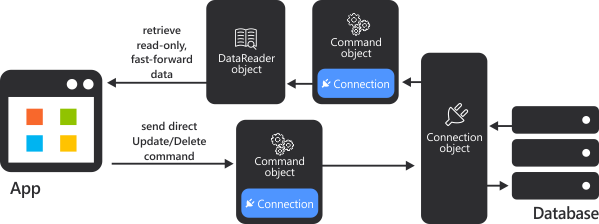
Tako se korišćenjem ovog pristupa podacima rukuje samo kratak vremenski period koji je potreban da se izvrši naredba i podaci pročitaju. Nakon toga konekcija se zatvara.

|  |
| --- |
| ***Napomena:****ovakav pristup je posebno pogodan za korišćenje u web aplikacijama, s obzirom na to da se podaci ne čuvaju u memoriji duže nego što je potrebno. I same web stranice ne žive mnogo duže. Naime, web stranica se učitava u trenutku kada server dobije zahtev za njom, a napušta se onoga trenutka kada se odgovor prosledi klijentu. Tako je životni vek same stranice veoma kratak, pa nema ni potrebe za smeštanjem podataka u memoriju.* |

Postupak direktnog pristupa podacima baze podataka korišćenjem ADO.NET-a se može podeliti u nekoliko koraka:

* + kreiranje objekta konekcije,
  + kreiranje objekta komande sa SQL naredbom za određenu konekciju,
  + izvršavanje komande,
  + čitanje podataka korišćenjem DataReader objekta, ukoliko je reč o naredbi za čitanje podataka.

Ovaj proces prikazan je na slici 2.1.



*Slika 2.1 – Rad sa podacima u konektovanom okruženju*

Kao što je već više puta rečeno, da bi se direktno pristupilo podacima u bazi, neophodno je postojanje konekcije. Stoga je prvi korak u radu sa podacima kreiranje konekcije.

**Kreiranje konekcije**

Pre bilo kakve direktne manipulacije podacima, bilo da je reč o čitanju, brisanju, unosu ili ažuriranju, neophodno je postojanje otvorene konekcije sa bazom. Konekciju sa bazom je potrebno držati otvorenom što je kraće moguće, a takođe je potrebno voditi računa o tome da se otvorena konekcija uvek zatvori.

Prilikom kreiranja objekta konekcije, neophodno je priložiti vrednost za svojstvo *ConnectionString*. Ovo svojstvo definiše sve informacije koje su potrebne aplikaciji kako bi pronašla izvor podataka, povezala se na njega i odabrala odgovarajuću bazu sa čijim podacima je potrebno raditi.

Sledeći primer ilustruje kreiranje objekta konekcije i definisanje vrednosti konekcionog stringa za bazu podataka Northwind.

|  |  |
| --- | --- |
| 1  2 | SqlConnection myConnection = new SqlConnection();  myConnection.ConnectionString = "Data Source=localhost; Initial Catalog=NORTHWND;Integrated Security=SSPI"; |

|  |  |  |
| --- | --- | --- |
| ***Napomena:****za rad sa ADO.NET tipovima, neophodno je uključiti odgovarajuće prostore imena. Ukoliko se radi sa SQL Server provajderom, potrebno je dodati prostor imena:*     |  |  | | --- | --- | | 1 | using System.Data.SqlClient; | |

**Konekcioni string**

Konekcioni string je serija informacija koje su odvojene karakterom tačka zapeta (;). Svaka ova informacija unutar konekcionog stringa naziva se svojstvo konekciong stringa.

*Data source* svojstvo ukazuje na naziv servera gde se podaci nalaze. Ukoliko se server nalazi na lokalnom računaru, dovoljno je napisati *localhost*.

*Initial catalog* svojstvo ukazuje na naziv baze podataka čijim podacima će se baratati. Stoga je na SQL serveru potrebno imati bazu podataka sa nazivom NORTHWND. Jasno je da je uloga prikazanog konekcionog stringa povezivanje sa bazom Northwind, a može se desiti da ona na Vašem serveru ima neki drugi naziv.

*Initial catalog* svojstvo ukazuje samo na bazu koja će se inicijalno koristiti, te je stoga moguće lako promeniti bazu sa kojom se radi, korišćenjem ChangeDatabase metode Connection klase.

*Integrated security*ukazuje na to da je sa SQL Serverom potrebno ostvariti konekciju korišćenjem Windows korisničkog naloga koji je aktivan u trenutku izvršavanja koda. [SSPI](https://www.link-elearning.com/linkdl/opisPojma.php?id=142478) se odnosi  na *Security Support Provider Interface*. Pored ovog načina, za proveru identiteta korisnika moguće je koristiti i korisničko ime i lozinku korisnika baze podataka, ali se takav pristup smatra manje sigurnim.

Pored navedenih svojstava, prilikom definisanja konekcionog stringa, moguće je navesti vrednost i za svojstvo *ConnectionTimeout*. Ovo svojstvo se koristi za definisanje broja sekundi koje će okruženje čekati na uspostavljanje konekcije, pre nego što generiše grešku zbog nemogućnosti konekcije. U primeru u ovoj lekciji pomenuti podatak nije naveden, pa se koristi podrazumevana vrednost ovog svojstva, što je 15 sekundi. Moguće je koristiti vrednosti 0, kako bi se definisalo odsustvo limita, ali je tako nešto loša praksa, jer bi u slučaju nemogućnosti uspostavljanja konekcije, kod ostao blokiran zauvek.

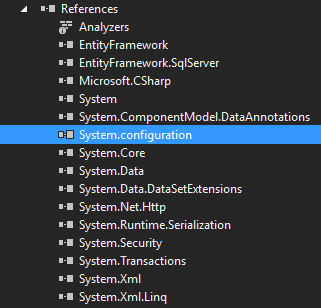
Česta praksa jeste čuvanje konekcionog stringa unutar [konfiguracionog fajla](https://www.link-elearning.com/linkdl/opisPojma.php?id=142479). Unutar konfiguracionog fajla, konekcioni string se smešta unutar <connectionStrings> sekcije, kao u sledećem primeru:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <configuration>    <connectionStrings>      <add name="Northwind" connectionString="Data Source=localhost;Initial Catalog=NORTHWND;Integrated Security=SSPI"/>    </connectionStrings>  ...  </configuration> |

Kada se konekcioni string nalazi u konfiguracionom fajlu, on se može iskoristiti na sledeći način:

|  |  |
| --- | --- |
| 1  2 | SqlConnection conn = new SqlConnection();  conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString; |

Kako bi se koristila klasa ConfigurationManager, koja omogućava čitanje podataka iz konfiguracionog fajla, neophodno je dodati referencu na biblioteku System.Configuration.dll, kao na slici.



*Slika 2.2 – System.configuration referenca*

|  |  |  |
| --- | --- | --- |
| ***Napomena:****da bi se identično postiglo u slučaju razvoja web aplikacije, kada se umesto fajla app.config koristi web.config, moguće je napisati sledeće:*     |  |  | | --- | --- | | 1  2 | SqlConnection conn= newSqlConnection();  conn.ConnectionString = WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString; | |

Sledeći primer ilustruje kompletnu strukturu koda pomoću koga se uspostavlja direktna veza sa bazom podataka:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | SqlConnection conn = new SqlConnection();              conn.ConnectionString =    ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;              try              {                  // Try to open the connection.                  conn.Open();                  Console.WriteLine("Server Version: " + conn.ServerVersion);                  Console.WriteLine("Connection Is: " + conn.State.ToString());              }              catch (Exception err)              {                  // Handle an error by displaying the information.              }              finally              {                  // Either way, make sure the connection is properly closed.                  // (Even if the connection wasn't opened successfully,                  // calling Close() won't cause an error.)                  conn.Close();                  Console.WriteLine("Now Connection Is:" + conn.State.ToString());              } |

Kao što je već više puta rečeno, prilikom direktnog rada sa bazom podataka neophodno je konekciju držati što kraće otvorenom i odmah nakon završenog rada tu istu konekciju zatvoriti. Takođe, prilikom otvaranja konekcije i rada sa podacima, vrlo lako može doći do pojave izuzetka. Zbog toga je kod za otvaranje konekcije smešten unutar try bloka. Unutar catch bloka se obrađuju eventualni izuzeci do kojih može doći, dok finally blok služi da se konekcija zatvori. Ovaj blok će se izvršiti u svakom slučaju, bilo da je do izuzetka došlo ili ne, tako da je idealan za zatvaranje konekcije.

Pored prikazanog šablona za konektovanje na bazu, moguće je iskoristiti i drugačiji pristup, koji podrazumeva korišćenje using bloka. Sledeći primer ilustruje upotrebu using bloka i ekvivalentan je prethodnom primeru:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | SqlConnection conn = new SqlConnection();              conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;              try             {                 using (conn)                  {                      // Try to open the connection.                      conn.Open();                      Console.WriteLine("Server Version: " + conn.ServerVersion);                      Console.WriteLine("Connection Is: " + conn.State.ToString());                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. ");                  Console.WriteLine(err.Message);              }              Console.WriteLine("Now Connection Is:" + conn.State.ToString()); |

Blok koda using omogućava automatsko oslobađanje resursa pozivanjem Dispose metode, za instance koje su kreirane unutar zaglavlja ovog bloka. Tako u prikazanom primeru nije neophodno eksplicitno pozivati metodu za zatvaranje konekcije, jer će se tako nešto dogoditi čim se završi izvršavanje using bloka.

**Za kreiranje konekcije sa bazom podataka neophodno je postojanje:**

 konekcije

 konekcionog stringa

 konfiguracionog fajla

 DataSet objekta

**Čitanje podataka (Read)**

Nakon uspostavljanja konekcije sa bazom podataka može se preći na manipulaciju podacima. U ovoj lekciji će prvo biti prikazan postupak za čitanje podataka.

Za kreiranje konekcije korišćen je objekat Connection klase. Za manipulaciju podacima koriste se objekti klase Command. Korišćenjem Command objekta, definiše se SQL naredba koju je potrebno proslediti bazi podataka na izvršavanje. Sledeći kod ilustruje način na koji je moguće pročitati nazive i cene proizvoda iz tabele *Products*, *Northwind* baze podataka.

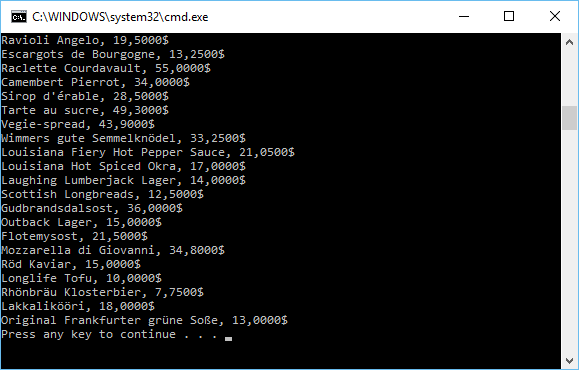
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | SqlConnection conn = new SqlConnection();              conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;              try              {                  using (conn)                  {                      conn.Open();                      SqlCommand command = new SqlCommand("SELECT ProductName, UnitPrice FROM Products", conn);                      using (SqlDataReader reader = command.ExecuteReader())                      {                          while (reader.Read())                          {                              Console.WriteLine(reader["ProductName"] + ", " + reader["UnitPrice"] + "$");                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Kao i do sada, kod za komunikaciju sa bazom podataka započinje kreiranjem konekcije, instanciranjem objekta SqlConnection tipa. Kako bi se bazi uputila komanda, koristi se objekat tipa SqlCommand. Komandi se prosleđuje SQL upit i objekat konekcije. SQL upit obavlja čitanje naziva i cene proizvoda iz tabele *Products* iz baze *Northwind*. Komanda se izvršava pozivanjem metode ExecuteReader, koja kao povratnu vrednost ima objekat tipa SqlDataReader.

SqlDataReader za rad koristi aktivnu konekciju, tako da je potrebno koristiti ga brzo i efikasno. Takođe, SqlDataReader omogućava čitanje samo unapred. Za pristup podacima unutar SqlDataReader objekta koriste se nazivi kolona u bazi podataka. Takođe, za prelazak na sledeći red unutar SqlDataReader kolekcije, koristi se metoda Read.

|  |  |  |
| --- | --- | --- |
| ***Napomena:****čitanje podataka iz SqlDataReader instance moguće je obaviti i korišćenjem indeksa. Tako je ekvivalentna linija za čitanje sledeća:*     |  |  | | --- | --- | | 1 | Console.WriteLine(reader[0] + ", " + reader[1] + "$"); | |

*Slika 2.3 prikazuje efekat koji prikazani kod ima.*

**

*Slika 2.3 – Pročitani podaci iz baze Northwind*

**Kreiranje podataka (Create)**

Nakon čitanja biće prikazan način na koji se može obaviti upisivanje novih podataka u bazu. Sledeći kod ilustruje način na koji je moguće obaviti upisivanje podataka.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | SqlConnection conn = new SqlConnection();              conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;              try              {                  using (conn)                  {                      conn.Open();                        SqlCommand command = new SqlCommand("INSERT INTO Products (ProductName, Discontinued) VALUES ('NewProduct', 0)", conn);                      int added = command.ExecuteNonQuery();                      Console.WriteLine("Just added " + added + " record.");                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Kao i do sada, kod započinje kreiranjem objekta konekcije. Komanda se i ovoga puta kreira instanciranjem objekta, tipa SqlCommand, a za unos podataka se koristi INSERT SQL naredba. Prilikom unosa novog zapisa u neku tabelu, potrebno je definisati vrednosti svih kolona koje ne smeju biti null. U primeru se zapis dodaje u tabelu *Products*, a kolone koje ne smeju imati null vrednost u ovoj tabeli su *ProductId*, *ProductName* i *Discontinued*. *ProductID* je kolona primarnog ključa, i njena vrednost se formira u bazi podataka, zato što je ova kolona definisana kao Identity kolona, što znači da ima [*Auto-Increment*](https://www.link-elearning.com/linkdl/opisPojma.php?id=142447) osobinu. Zbog toga je dovoljno da se prilikom unosa novog reda definišu vrednosti samo za kolone  *ProductName* i *Discontinued*.

Vrednosti kolona *ProductName* i *Discontinued*za red koji se unosi, u prikazanom primeru su ugrađene u sam upit. ADO.NET omogućava elegantan način za kreiranje parametrizovanih upita korišćenjem SqlParameter tipa. Na taj način se može postići kreiranje dinamičkih upita, što poboljšava preglednost, ali i smanjuje rizik od [SQL Injection](https://www.link-elearning.com/linkdl/opisPojma.php?id=142469) napada. Da bi se prikazana naredba transformisala korišćenjem tipa SqlParameter, moguće je napisati sledeće:

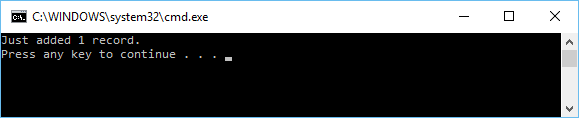
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | SqlConnection conn = new SqlConnection();              conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;              try              {                  using (conn)                  {                      conn.Open();                      SqlCommand command = new SqlCommand("INSERT INTO Products (ProductName, Discontinued) VALUES (@ProductName, @Discontinued)", conn);                      SqlParameter productNameParam = new SqlParameter("@ProductName", SqlDbType.NVarChar);                      productNameParam.Value = "NewProduct";                      SqlParameter discontinuedParam = new SqlParameter("@Discontinued", SqlDbType.Bit);                      discontinuedParam.Value = false;                      command.Parameters.Add(productNameParam);                      command.Parameters.Add(discontinuedParam);                      int added = command.ExecuteNonQuery();                      Console.WriteLine("Just added " + added + " record.");                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Sada su u prikazanom primeru, umesto vrednosti parametara, definisani takozvani *placeholderi*. *Placeholderi* su mesta u upitu na koja će dinamički biti unete vrednosti. Placeholderi su prepoznaju po karakteru @, kojim započinju.

Da bi se definisale vrednosti ovako definisanih parametara, koristi se tip SqlParameter. Konstruktor ovog tipa prihvata dva parametra. Prvi se odnosi na naziv *placeholdera*, a drugi na tip vrednosti. Vrednost ovako definisanog parametra se postavlja korišćenjem svojstva Value. Na kraju, kreirane parametre je potrebno dodati objektu konekcije, korišćenjem kolekcije Parameters i metode Add.

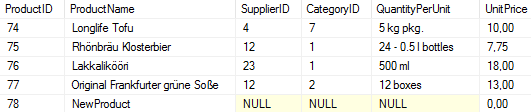
Komanda se ovoga puta izvršava pozivom metode ExecuteNonQuery, pošto se od baze ne očekuju podaci. Ova metoda, ipak, ima povratnu vrednost u obliku jedne celobrojne vrednosti koja ukazuje na to koliko je redova u bazi podataka zahvaćeno SQL komandom koja je izvršena.

Nakon izvršenja koda za unos novog zapisa, izgled konzolnog prozora će biti kao na slici 2.4.



*Slika 2.4 – Poruka nakon dodavanja novog reda u tabelu u bazi*

Nakon uspešnog dodavanja novog proizvoda u tabelu *Products*, u bazi podataka postojaće dodati red, kao na slici 2.5.



*Slika 2.5 – Novi red sa brojem 78 u bazi podataka*

**Ažuriranje podataka (Update)**

Nakon čitanja i unosa biće prikazan i postupak koji se koristi za ažuriranje podataka u bazi. Ponovo će biti potrebno koristiti objekat Command klase, koji mora da dobije objekat Connection tipa.

Sledeći kod ilustruje logiku za ažuriranje podataka.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | SqlConnection conn = new SqlConnection();              conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;                try              {                  using (conn)                  {                      conn.Open();                      SqlCommand command = new SqlCommand("UPDATE Products SET ProductName = @ProductName WHERE ProductID = @ProductID", conn);                      SqlParameter productNameParam = new SqlParameter("@ProductName", SqlDbType.NVarChar);                      productNameParam.Value = "UpdatedName";                      SqlParameter productIdParam = new SqlParameter("@ProductID", SqlDbType.Int);                      productIdParam.Value = 78;                      command.Parameters.Add(productNameParam);                      command.Parameters.Add(productIdParam);                      int rows = command.ExecuteNonQuery();                      Console.WriteLine("Just updated " + rows + " record.");                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Nakon uobičajenog kreiranja i otvaranja konekcije, kreiran je objekat SqlCommand tipa. Ovaj objekat je ovoga puta dobio kod SQL UPDATE naredbe. Kao i prilikom unosa podataka, u primeru se koristi parametrizovani upit.

|  |
| --- |
| ***Napomena:****u primeru se ažurira red koji ima vrednost ProductID kolone 78. Sa slike 2.5 možete da vidite da je taj Id dobio red koji je dodat prilikom demonstracije dodavanja zapisa, tako da kod Vas situacija može biti drugačija. Takođe, unutar WHERE klauzule mogu se koristiti i druge kolone za filtriranje.* |

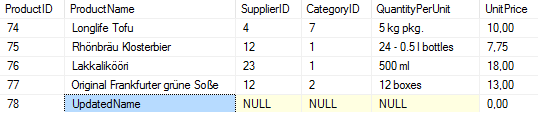
Komanda se ovoga puta izvršava pozivom metode ExecuteNonQuery, pošto se od baze ne očekuju podaci. Ova metoda, ipak, ima povratnu vrednost u obliku jedne celobrojne vrednosti koja ukazuje na to koliko je redova u bazi podataka ažurirano SQL komandom koja je izvršena.

Nakon izvršavanja prikazanog koda, izgled konzolnog prozora je kao na slici 2.6.



*Slika 2.6 – Poruka nakon ažuriranja reda u tabeli u bazi*

Nakon uspešnog ažuriranja reda u tabeli Products baze Northwind situacija će biti kao na slici 2.7.



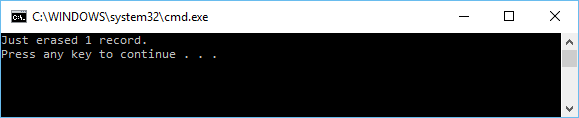
*Slika 2.7 – Ažurirani red u bazi podataka*

**Brisanje podataka (Delete)**

Na kraju će biti prikazan i postupak za brisanje zapisa iz baze podataka. Biće obrisan zapis koji je u prethodnim primerima kreiran i ažuriran. Kod za brisanje dat je u nastavku:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | SqlConnection conn = new SqlConnection();              conn.ConnectionString = ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;              try              {                  using (conn)                  {                      conn.Open();                      SqlCommand command = new SqlCommand("DELETE FROM Products WHERE ProductID=@ProductID", conn);                      SqlParameter productIdParam = new SqlParameter("@ProductID", SqlDbType.Int, 11);                      productIdParam.Value = 78;                      command.Parameters.Add(productIdParam);                      int rows = command.ExecuteNonQuery();                      Console.WriteLine("Just erased " + rows + " record.");                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

U kodu koji je prikazan nema nikakvih nepoznatih elemenata. Kreiran je, otvara konekciju i upućuje komadu bazi, korišćenjem SqlCommand instance. SQL naredba koja se prosleđuje bazi je DELETE naredba. Parametar se dodaje dinamički, korišćenjem SqlParameter tipa. Vrednost parametra je 78, što je vrednost ID zapisa, koja je nešto ranije upisana u bazu. Nakon uspešnog izvršavanja prikazanog koda izgled konzolnog prozora će biti kao na slici 2.8.



*Slika 2.8 – Poruka nakon brisanja jednog reda u bazi*