**Rukovanje podacima u nepovezanom okruženju**

**00:00:00**

Jedinica: 3 od 19

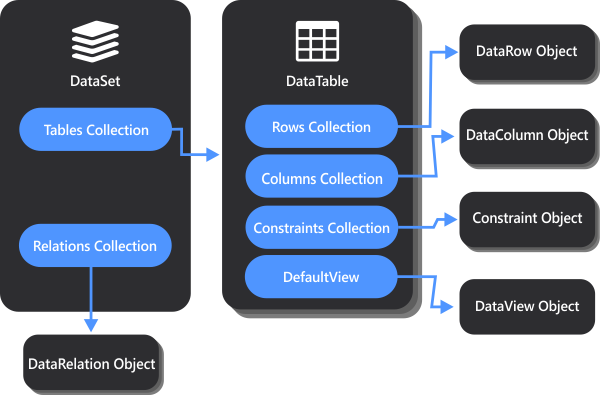
**Rezime**

* Korišćenjem nepovezanog modela za pristup podacima, kreira se kopija stvarnih podataka u memoriji.
* Konekcija sa bazom u nepovezanom scenariju traje samo onoliko koliko je potrebno da se podaci pročitaju i smeste unutar specijalnog objekta tipa DataSet.
* Objekti klase DataSet su reprezentacija podataka iz baze u memoriji.
* DataSet objekat može da sadrži kolekciju jednog ili više DataTable objekata, koji predstavljaju tabele baze podataka.
* DataTable objekat može da sadrži kolekciju redova, kolona i ograničavača.
* DataRow tip predstavlja red podataka u DataTable objektu.
* DataColumn tip predstavlja šemu jedinstvene kolone u tabeli.

ADO.NET, pored modela za direktan pristup, omogućava i model za nekonektovani (nepovezani) pristup podacima. Korišćenjem nepovezanog modela za pristup podacima, kreira se kopija stvarnih podataka u memoriji. Konekcija sa bazom traje samo onoliko koliko je potrebno da se podaci pročitaju i smeste unutar specijalnog objekta tipa DataSet, nakon čega se rad sa podacima nastavlja u potpuno nekonektovanom okruženju, gde konekcija sa bazom više nije potrebna.

**DataSet**

Da bi nekonektovani model rada sa podacima bio moguć, koristi se klasa DataSet. Objekti ove klase su zapravo reprezentacija podataka iz baze u memoriji. DataSet objekat može da sadrži kolekciju jednog ili više DataTable objekata, koji predstavljaju tabele baze podataka. DataTableobjekat može da sadrži kolekciju redova, kolona i [ograničavača](https://www.link-elearning.com/linkdl/opisPojma.php?id=142470). Kompletna opisana hijerarhija klasa, sa osnovnom DataSet klasom, prikazana je slikom 3.1.



*Slika 3.1 – Struktura podataka DataSet objekta*

Sve klase prikazane na slici 3.1 nalaze se u prostoru imena System.Data. Takođe, ove klase se mogu koristiti za predstavljanje podataka iz bilo koje baze koju ADO.NET podražava, bez ikakve potrebe za korišćenjem tipova specifičnih za određeno skladište.

Tabela 3.1 prikazuje uporedni pregled klasa karakterističnih za nekonektovani model.

|  |  |
| --- | --- |
| **Klasa** | **Opis** |
| **DataSet** | Predstavlja bazu podataka u memoriji. **DataSet**može da sadrži jedan ili više objekata tipa **DataTable**i **DataRelation**. Objekti ove klase se mogu serijalizovati i prenositi putem mreže. |
| **DataTable** | Jedna tabela podataka u memoriji. |
| **DataRelation** | Roditelj / dete odnos između dva DataTable objekta; dozvoljava navigaciju između dve vezane tabele. |
| **DataRow** | Red podataka u DataTable objektu. |
| **DataColumn** | Šema jedinstvene kolone u tabeli. |
| **Constraint** | Ograničenje, koje se može primeniti na jedan ili više DataColumn objekata. Ograničenje je pravilo koje čuva integritet podatka u tabeli. Apstraktna Constraint klasa ima dve izvedene klase, **UniqueConstraint**i **ForeignKeyConstraint**, koje se koriste za primarne ključeve i/ili jedinstvene vrednosti kolone. |
| **DataView** | Prilagođeni prikaz DataTablea, koji se može koristiti za sortiranje, filtriranje, pretraživanje, editovanje i navigaciju. |

*Tabela 3.1 – Uporedni pregled klasa karakterističnih za nekonektovani model*

**Data Row**

DataRow objekat predstavlja jedan red podataka unutar objekta DataTable i pristupa mu se preko Rows svojstva. Objekti DataRow klase omogućavaju preuzimanje, insertovanje, ažuriranje i brisanje podataka tabele. Tip DataRow definisan je u System.Data prostoru imena.

Svaki DataRow objekat skladišti i originalne podatke za svaki red, ali i trenutne podatke, u slučaju da je modifikovan. Kada se promene u DataSet objektu sačuvaju, DataSet objekti mogu koristiti ovu informaciju da bi generisali naredbu koja će ažurirati odgovarajući red u bazi. DataRow objekat poseduje informacije koje ukazuju na to šta je potrebno uraditi sa podacima. Tako red može biti označen za brisanje i u tom slučaju potrebno ga je ukloniti u trenutku kada se promene sačuvaju u bazi. Red takođe može biti i upravo kreirani, novi red koji je potrebno ubaciti u bazu nakon čuvanja promena.

U pasusima koji dolaze, kroz demonstraciju osnovnih CRUD operacija, biće ilustrovan rad sa tipom DataRow.

**DataColumn**

DataColumn predstavlja jednu kolonu podataka neke tabele.

DataRow objekat sadrži podatke za jednu ili više kolona. Kolona je modelovana od strane DataColumn objekta koji sadrži metainformaciju o trenutnoj koloni u bazi, kao što su ime i tip. DataTable objekat obezbeđuje pristup DataColulmn objektima za tu tabelu, kroz kolekciju Columns.

Najvažnija svojstva DataColumn objekta prikazan su tabelom 3.2.

|  |  |
| --- | --- |
| **Naziv svojstva** | **Opis** |
| **AllowDbNull** | Indicira da li kolona može imati null vrednost. Podrazumevana vrednost je true. |
| **AutoIncrement** | Indicira da li će se vrednost numeričke kolone automatski uvećavati pri dodavanju novih redova. Podrazumevana vrednost je false. |
| **ColumnName** | Naziv kolone. |
| **DataType** | Tip podataka koji kolona može sadržati. Tip podataka DataTable objekta se mora poklapati sa tipom podataka u izvoru podataka. |
| **DefaultValue** | Podrazumevana vrednost. |
| **ReadOnly** | Indicira da li su promene nad kolonom dozvoljene, nakon što se red doda u tabelu. Podrazumevana vrednost je false. |
| **Table** | Vraća referencu na DataTable objekat kojem dati DataColumn objekat pripada. |
| **Unique** | Definiše da li vrednosti ove kolone u svakom redu moraju biti jedinstvene. |

*Tabela 3.2 – Najvažnija svojstva DataColumn objekta*

DataColumn kolekciji se pristupa preko svojstva Columns, DataTable objekta. Ova DataColumnCollection kolekcija, zajedno sa ograničenjima, definiše šemu, odnosno strukturu tabele. Instanca klase DataColumn može se kreirati upotrebom nekog od konstruktora, prikazanih tabelom 3.3.

|  |  |
| --- | --- |
| **Konstruktor** | **Opis** |
| **DataColumn()** | Inicijalizuje novu instancu klase DataColumn. |
| **DataColumn(String)** | Parametar predstavlja naziv kolone. |
| **DataColumn(String, Type)** | Drugi parametar predstavlja tip podataka date kolone. |
| **DataColumn (String, Type, String)** | Treći parametar predstavlja izraz koji će se koristiti za kreiranje ove kolone. Vrednost se izračunava na osnovu vrednosti drugih kolona ili pomoću agregatnih funkcija. Ovom izrazu se može pristupiti preko DataColumn osobine Expression. |

*Tabela 3.3 – DataColumn konstruktori*

**Zašto koristiti DataSet?**

Smeštanje podataka u memoriju donosi nekoliko prednosti. Ukoliko je potrebno obaviti neku vremenski zahtevnu operaciju nad podacima, smeštanjem podataka unutar DataSet objekta osigurava se što kraće držanje konekcije otvorenom. Ukoliko je potrebno koristiti povezivanje podataka (binding), mnogo bolji izbor predstavlja korišćenje DataSeta. Takođe, DataSet omogućava kretanje kroz kolekciju podataka u oba smera, što nije slučaj sa DataReader objektom. Korišćenjem DataSet objekta moguće je u memoriju smestiti i više tabela, pa čak i međusobne zavisnosti koje postoje među njima. Nepovezan model rada sa podacima omogućava i čuvanje podataka u [XML](https://www.link-elearning.com/linkdl/opisPojma.php?id=142476) formi za kasniju upotrebu. Na kraju, korišćenjem DataSeta mogu se sačuvati podaci za kasniju obradu, što je tehnika koja se naziva keširanje.

**DataTable objekat sadrži kolekciju objekata tipa DataSet.**

 Tačno

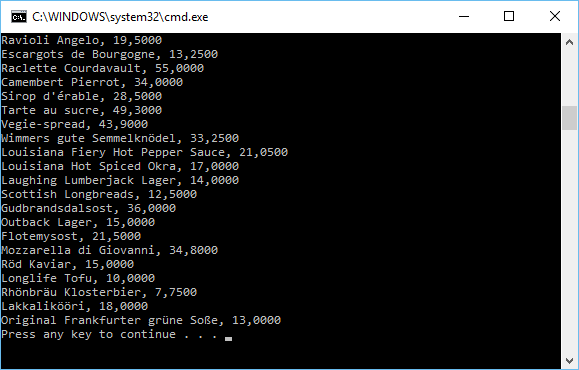
 Netačno

**Čitanje podataka**

Osnovna razlika u korišćenju DataSet i DataReader klasa je ta što DataReader zahteva otvorenu konekciju sa bazom za rad sa podacima. Prilikom rada sa DataSet objektom, tako nešto nije potrebno. Sledeći primer ilustruje tehniku za čitanje svih proizvoda iz baze podataka *Northwind*, koja je sada realizovana upotrebom nepovezanog modela, korišćenjem DataSetklase.

|  |  |
| --- | --- |
| 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(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              try              {                  using (conn)                  {   SqlCommand cmd = new SqlCommand("SELECT ProductName, UnitPrice FROM Products", conn);                      using (SqlDataAdapter dataAdapter = new SqlDataAdapter(cmd))                      {                          using (DataSet dataSet = new DataSet())                          {                              dataAdapter.Fill(dataSet, "Products");                              foreach (DataRow row in dataSet.Tables["Products"].Rows)                              {                       Console.WriteLine(row["ProductName"] + ", " + row["UnitPrice"]);                              }                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

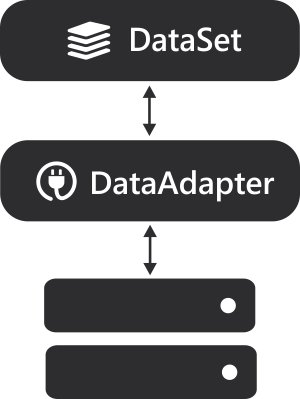
Nakon izvršavanja prikazanog koda, izgled konzolnog prozora će biti kao na slici 3.2.



*Slika 3.2 – Podaci pročitani iz baze*

U prikazanom kodu, kreiranje objekata konekcije i komande je identično kao u primeru konektovanog modela, iz prethodne lekcije. Novine su objekti klasa SqlDataAdapter i DataSet.

DataAdapter objekat se koristi za dohvatanje podataka iz baze podataka i smeštanje tih podataka u DataSet objekat, što je ilustrovano na slici 3.3. Ovo je praktično jedina operacija koja zahteva otvorenu konekciju sa bazom. U prikazanom primeru se može videti da je ovo punjene podacima jedina operacija, koja se obavlja sa otvorenom konekcijom. Nakon završetaka using bloka konekcija sa bazom se zatvara.



*Slika 3.3 – Uloga DataAdapter objekta*

Svaki Data provajder ima svoju verziju DataAdaptera, pa se tako za rad sa MS SQL Serverom koristi SqlDataAdapter klasa.

DataAdapter objekat olakšava sinhronizaciju podataka između klijenta i servera. Poseduje set naredbi koje se koriste da bi se popunio DataSet objekat podacima. DataAdapter koristi Connection objekat za povezivanje sa bazom podataka i Command objekte za implementiranje komandi za upravljanje podacima.

Postoji nekoliko konstruktora koji se mogu koristiti za instanciranje SqlDataAdapter klase. Oni su prikazani tabelom 3.4.

|  |  |
| --- | --- |
| **Konstruktor** | **Opis** |
| **SqlDataAdapter()** | Konstruktor bez parametara. SQL naredba i objekat konekcije dodeljuju se naknadno korišćenjem svojstava. |
| **SqlDataAdapter(SqlCommand)** | Konstruktor koji prihvata objekat SqlCommand klase. |
| **SqlDataAdapter(String, SqlConnection)** | Konstruktor koji prihvata string SQL komande i objekat konekcije. |
| **SqlDataAdapter(String, String)** | Konstruktor koji prihvata string SQL komande i konekcioni string. |

*Tabela 3.4 – DataColumn konstruktori*

Slede primeri upotrebe navedenih konstruktora:

|  |  |
| --- | --- |
| 1  2  3  4 | SqlDataAdapter da = new SqlDataAdapter();  SqlDataAdapter da = new SqlDataAdapter(cmd);  SqlDataAdapter da = new SqlDataAdapter(sql, conn);  SqlDataAdapter da = new SqlDataAdapter(sql, connString); |

DataAdapter klasa poseduje metodu Fill, koja prihvata DataSet objekat i naziv tabele, čije podatke će učitati u DataSet objekat. U prikazanom primeru naziv tabele je *Products.*

Za pristup konkretnim redovima tabele *Products*, koristi se klasa DataRows, tako što se prolazi kroz Rows kolekciju. Do konkretnih podataka se dolazi korišćenjem naziva kolone, na isti način kao što je to postignuto i korišćenjem DataReader objekta.

**Kreiranje podataka (Create)**

Nakon čitanja podataka biće prikazana i tehnika za unos novih redova u tabele baze podataka korišćenjem nepovezanog pristupa. Kod za dodavanje novog reda u tabelu *Products*, baze podataka *Northwind* je sledeći:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              try              {                  using (conn)                  {                      using (SqlDataAdapter dataAdapter = new SqlDataAdapter())                      {                          dataAdapter.SelectCommand = new SqlCommand("SELECT ProductName, UnitPrice, Discontinued FROM Products", conn);                          dataAdapter.InsertCommand = new SqlCommand("INSERT INTO Products (ProductName, Discontinued) values (@ProductName, @Discontinued)", conn);                          dataAdapter.InsertCommand.Parameters.Add(new SqlParameter("ProductName", SqlDbType.NVarChar, 40, "ProductName"));                          dataAdapter.InsertCommand.Parameters.Add(new SqlParameter("Discontinued", SqlDbType.Bit, 1, "Discontinued"));                          using (DataSet dataSet = new DataSet())                          {                              dataAdapter.Fill(dataSet, "Products");                              Console.WriteLine("There are {0} rows in the table", dataSet.Tables["Products"].Rows.Count);                              DataRow newRow = dataSet.Tables["Products"].NewRow();                              newRow["ProductName"] = "NewProduct";                              newRow["Discontinued"] = false;                              dataSet.Tables["Products"].Rows.Add(newRow);                              dataAdapter.Update(dataSet, "Products");                          }                          //Just to prove we inserted                          using (DataSet newDataSet = new DataSet())                          {                              dataAdapter.Fill(newDataSet, "Products");                              Console.WriteLine("There are {0} rows in the table", newDataSet.Tables["Products"].Rows.Count);                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

S obzirom da se DataAdapter klasa koristi kao posrednik između fizičkih podataka i DataSetobjekta, ona poseduje četiri osnovna svojstva, kojima se definišu naredbe za obavljanje osnovnih CRUD operacija. Ta četiri svojstva su sledeća:

* + SelectCommand – obezbeđuje SQL SELECT naredbu za čitanje podataka.
  + UpdateCommand – obezbeđuje SQL UPDATE naredbu za ažuriranje podataka.
  + InsertCommand – obezbeđuje SQL INSERT naredbu za ubacivanje novih podataka.
  + DeleteCommand – obezbeđuje SQL DELETE naredbu za brisanje podataka.

|  |
| --- |
| ***Napomena:****prilikom čitanja podataka u prethodnom primeru ova svojstva nisu korišćena, zato što je naredba prosleđena SqlDataAdapter objektu korišćenjem konstruktora. Konstruktor SqlDataAdapter klase omogućava navođenje samo naredbe za selekciju podataka a ostale naredbe se moraju definisati korišćenjem prikazanih svojstava.* |

U prikazanom primeru su korišćenjem svojstava SelectCommand i InsertCommand definisane naredbe za čitanje i unos novih zapisa. Adapter je prvo iskorišćen za popunjavanje DataSet objekta podacima, a zatim i za ažuriranje.

Da bi se dodao novi red u neku tabelu, takav red je prvo potrebno kreirati. To je u primeru i postignuto korišćenjem klase DataRow, koja predstavlja jedan red u tabeli baze podataka. Kreiranom redu postavljene su vrednosti kolona, korišćenjem njihovih naziva, a na kraju je takav red dodat u kolekciju pod nazivom Rows, korišćenjem metode Add. Na kraju, kako bi DataAdapter objekat uistinu i izmenio stanje DataSet objekta, pozvana je metoda Update.

Iz svega do sada navedenog može se zaključiti da DataAdapter objekat ne generiše samostalno komande za interakciju sa podacima. U primeru ubacivanja novog reda u bazu podataka, bilo je neophodno definisati vrednost InsertCommand naredbe. Takav slučaj je i sa izvršavanjem ostalih komandi. Ipak, stvari se mogu u izvesnoj meri olakšati korišćenjem SqlCommandBuilder klase. Ova klasa ima sposobnost da na osnovu SELECT upita samostalno generiše ostale SQL upite. Tako je već naveden primer moguće uprostiti na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              try              {                  using (conn)                  {                      using (SqlDataAdapter dataAdapter = new SqlDataAdapter("SELECT \* FROM Products", conn))                      {                          using (DataSet dataSet = new DataSet())                          {                              dataAdapter.Fill(dataSet, "Products");                              Console.WriteLine("There are {0} rows in the table", dataSet.Tables["Products"].Rows.Count);                              DataRow newRow = dataSet.Tables["Products"].NewRow();                              newRow["ProductName"] = "NewProduct";                              newRow["Discontinued"] = false;                              dataSet.Tables["Products"].Rows.Add(newRow);                              SqlCommandBuilder objCommandBuilder = new SqlCommandBuilder(dataAdapter);                              dataAdapter.Update(dataSet, "Products");                          }                          //Just to prove we inserted                          using (DataSet newDataSet = new DataSet())                          {                              dataAdapter.Fill(newDataSet, "Products");                              Console.WriteLine("There are {0} rows in the table", newDataSet.Tables["Products"].Rows.Count);                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

U primeru se sada može primetiti odsustvo svojstava DataAdapter objekta kojima se definišu naredbe za obavljanje osnovnih CRUD naredbi. Jedina komanda koja je definisana jeste ona za obavljanje SELECT operacije, i to unutar konstruktora DataAdapter klase. Ovog puta su selektovane sve kolone *Products* tabele. Tako nešto nije obavezno, ali je neophodno da jedna od selektovanih kolona bude i ona koja predstavlja primarni ključ. Takva informacija će kasnije biti korišćena za obavljanje ažuriranja, brisanja i unosa. Da bi sve ovo bilo moguće, pre pozivanja Update metode DataAdapter klase, instanciran je objekat klase SqlCommandBuilder, kome je prosleđena instanca DataAdapter tipa.

**Ažuriranje podataka (Update)**

U ovom pasusu biće prikazan način na koji je moguće izvršiti ažuriranje podataka korišćenjem DataSeta. Postupak je vrlo sličan unosu novih podataka, jer se i u jednom i u drugom slučaju vrši ažuriranje stanja DataSet objekta.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              try              {                  using (conn)                  {                      using (SqlDataAdapter dataAdapter = new SqlDataAdapter())                      {                          dataAdapter.SelectCommand = new SqlCommand("SELECT ProductID, ProductName, UnitPrice, Discontinued FROM Products", conn);                          dataAdapter.InsertCommand = new SqlCommand("INSERT INTO Products (ProductName, Discontinued) values (@ProductName, @Discontinued)", conn);                          dataAdapter.InsertCommand.Parameters.Add(new SqlParameter("ProductName", SqlDbType.NVarChar, 40, "ProductName"));                          dataAdapter.InsertCommand.Parameters.Add(new SqlParameter("Discontinued", SqlDbType.Bit, 1, "Discontinued"));                            dataAdapter.UpdateCommand = new SqlCommand("UPDATE Products SET ProductName = @ProductName WHERE ProductID = @ProductID", conn);                          dataAdapter.UpdateCommand.Parameters.Add(new SqlParameter("ProductName", SqlDbType.NVarChar, 40, "ProductName"));                          dataAdapter.UpdateCommand.Parameters.Add(new SqlParameter("ProductID", SqlDbType.Int));                          using (DataSet dataSet = new DataSet())                          {                  dataAdapter.FillSchema(dataSet, SchemaType.Source, "Products");                              dataAdapter.Fill(dataSet, "Products");                              Console.WriteLine("There are {0} rows in the table", dataSet.Tables["Products"].Rows.Count);                              DataRow existingRow = dataSet.Tables["Products"].Rows.Find(80);                              existingRow["ProductName"] = "UpdatedName";                           dataAdapter.UpdateCommand.Parameters["ProductID"].Value = existingRow["ProductID"];                              dataAdapter.Update(dataSet, "Products");                          }                          //Just to prove we updated                          using (DataSet newDataSet = new DataSet())                          {                              dataAdapter.Fill(newDataSet, "Products");                              foreach (DataRow row in newDataSet.Tables["Products"].Rows)                              {                                  Console.WriteLine(row["ProductName"] + ", " + row["UnitPrice"]);                              }                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Za pronalazak konkretnog podatka za brisanje, u primeru se koristi metoda Find. Ovo je metoda DataRowCollection klase koja predstavlja kolekciju redova jedne tabele. Potpis metode je sledeći:

|  |  |
| --- | --- |
| 1 | public DataRow Find(object key) |

Metoda Find prihvata jedan parametar koji se odnosi na ID vrednost reda koji je potreno pronaći. Stoga je neophodno da tabela kojom se operiše poseduje kolonu primarnog ključa. Ukoliko red sa definisanom Id vrednošću ne bude pronađen, metoda Find vraća vrednost Null.

Prikazani primer podrazumeva scenario po kome se komande za interakciju sa podacima definišu ručno. Ekvivalentan efekat je mogao biti postignut i na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              try              {                  using (conn)                  {                      using (SqlDataAdapter dataAdapter = new SqlDataAdapter("SELECT \* FROM Products", conn))                      {                          using (DataSet dataSet = new DataSet())                          {                              dataAdapter.FillSchema(dataSet, SchemaType.Source, "Products");                              dataAdapter.Fill(dataSet, "Products");                                Console.WriteLine("There are {0} rows in the table", dataSet.Tables["Products"].Rows.Count);                                DataRow existingRow = dataSet.Tables["Products"].Rows.Find(80);                              existingRow["ProductName"] = "UpdatedName";                              SqlCommandBuilder objCommandBuilder = new SqlCommandBuilder(dataAdapter);                              dataAdapter.Update(dataSet, "Products");                          }                          //Just to prove we updated                          using (DataSet newDataSet = new DataSet())                          {                              dataAdapter.Fill(newDataSet, "Products");                              foreach (DataRow row in newDataSet.Tables["Products"].Rows)                              {                                  Console.WriteLine(row["ProductName"] + ", " + row["UnitPrice"]);                              }                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Ovoga puta, u primeru ažuriranja, komanda za ažuriranje nije eksplicitno navedena, već je njeno generisanje prepušteno SqlCommandBuilder objektu.

U primerima ažuriranja novost je i metoda FillSchema. Ova metoda dobavlja informacije šeme iz izbora podataka, i smešta objekat tipa DataTable unutar DataSet instance. Pritom, ova metoda konfiguriše razna svojstva pročitana iz skladišta, kao što su: null vrednosti, AutoIncrement svojstvo, maksimalna dužina podataka, ali i kolone primarnih ključeva. S obzirom na to da su informacije o primarnom ključu tabele neophodne za obavljanje ažuriranja, korišćenje ove metode u prikazanom primeru je obavezno.

**Brisanje podataka (Delete)**

Na kraju će biti demonstrirano i brisanje podataka korišćenjem nepovezanog pristupa. Brisanje podataka se obavlja ne veoma jednostavan način, sličan ažuriranju. Sledi primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | SqlConnection conn = new SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              try              {                  using (conn)                  {                      using (SqlDataAdapter dataAdapter = new SqlDataAdapter("SELECT \* FROM Products", conn))                      {                          using (DataSet dataSet = new DataSet())                          {                              dataAdapter.FillSchema(dataSet, SchemaType.Source, "Products");                              dataAdapter.Fill(dataSet, "Products");                              Console.WriteLine("There are {0} rows in the table", dataSet.Tables["Products"].Rows.Count);                              DataRow existingRow = dataSet.Tables["Products"].Rows.Find(84);                              existingRow.Delete();                              SqlCommandBuilder objCommandBuilder = new SqlCommandBuilder(dataAdapter);                              dataAdapter.Update(dataSet, "Products");                          }                          //Just to prove we deleted                          using (DataSet newDataSet = new DataSet())                          {                              dataAdapter.Fill(newDataSet, "Products");                              foreach (DataRow row in newDataSet.Tables["Products"].Rows)                              {                                  Console.WriteLine(row["ProductName"] + ", " + row["UnitPrice"]);                              }                          }                      }                  }              }              catch (Exception err)              {                  // Handle an error by displaying the information.                  Console.WriteLine("Error reading the database. " + err.Message);              } |

Nakon dolaska do instance DataRow klase, koja predstavlja red koji je potrebno obrisati, brisanje se izvršava pozivanjem metode Delete nad instancom klase DataRow. Kao i u prethodnim primerima, komada za brisanje generisana je od strane objekta klase SqlCommandBuilder.