

9. POGLAVLJE

ADO.NET

U ovom poglavlju:

- Osnove jezika SQL
- naredbe SELECT, INSERT, UPDATE i DELETE
- Arhitektura ADO.NET-a
- Osnovne komponente ADO.NET-a
- Razlika između *DataReader*a i *DataSet*a
- Praktičan rad s ADO.NET-om
- Prikaz podataka u kontroli *DataGrid*
- Korištenje *DataReader*a za dohvat podataka
- Stvaranje i izvršavanje objekata *Command*

U prethodnom poglavlju naučili ste izrađivati jednostavne aplikacije, a sad idemo korak dalje. U većini slučajeva, bilo da radite prozorske ili web-aplikacije (pa čak i aplikacije za mobilne uređaje ili web-servise), koristit ćete neke podatke. Ti će podaci ponekad biti spremljeni u zasebnim datotekama, no najčešće ćete koristiti neku bazu podataka.

III. DIO: DIJELOVI .NET-A

Za dohvaćanje podataka i komuniciranje s bazama podataka pod .NET-om postoji posebna tehnologija, nazvana ADO.NET. Ona ima svoju impozantnu prošlost, a prelaskom pod okrilje .NET-a doživjela je tolika poboljšanja da možemo govoriti o potpuno novoj tehnologiji. Microsoft pozicionira ADO.NET (od ActiveX Data Objects) kao primarnu tehnologiju za pristup podacima iz .NET Frameworka, te je stoga njeno poznavanje nužno za bilo kakav rad s podacima. Na svu sreću, iako se radi o moćnoj tehnologiji, relativno ju je lako naučiti.

Da biste se mogli upustiti u rad s bazama podataka, trebat ćete upoznati osnove jezika SQL, standarda za komunikaciju i rad s bazama. Na sljedećih nekoliko stranica dat ćemo vam pregled mogućnosti koje će vam trebati u svakodnevnom radu s bazama.

Osnove jezika SQL

SQL, *Structured Query Language*, standardni je jezik za komunikaciju s bazom podataka kroz tzv. *Database Management System* (DBMS) – sustav koji manipulira svim podacima u bazi. SQL koriste Oracle, Microsoft SQL Server, Microsoft Access, IBM DB2, Sybase i gotovo svi drugi sustavi na tržištu. Poznavanje SQL-a postaje nužno svakom programeru, bilo da se bavi izradom standardnih aplikacija ili je više okrenut Webu.



Ukoliko ste već upoznati s jezikom SQL i iza sebe imate iskustvo u radu s bazama podataka, možete preskočiti narednih nekoliko stranica, jer će na njima biti objašnjene osnovne SQL naredbe.

SQL jezik sastoji se od naredbi za dodavanje, čitanje, izmjenu i brisanje podataka. Pravila jezika odredio je ANSI (*American National Standards Institute*), što znači da je SQL jezik otvoren i ne kontrolira ga niti jedna tvrtka.



SQL nije program ili razvojna okolina poput Visual Studija .NET. SQL je isključivo opisni jezik (opisni jer opisuje akciju koju treba napraviti) i koristi se za komunikaciju s bazom podataka (njenim DBMS-om).

Kako je SQL samo jezik kojim se zadaju naredbe, on nema nikakvo sučelje kojim biste mogli isprobavati njegov učinak na bazu. Za to je zadužen sam program u kojem razvijate bazu podataka. Radite li sa SQL Serverom, otvorite Query Analyzer. Svi primjeri će biti pokazani na bazi Northwind

koja dolazi sa SQL Serverom i drugim Microsoftovim bazama podataka i idealna je za prikazivanje općenitih mogućnosti.

Ukoliko na raspolaganju nemate SQL Server, s Microsoftovih stranica možete besplatno preuzeti MSDE, bazu podataka koja u sebi ima istu arhitekturu kao i SQL Server, no kako je besplatna, nema razvojno okruženje u kojem biste mogli isprobati SQL upite. U tom slučaju možete upite isprobavati pomoću nekog besplatnog alata za rad s MSDE-om ili izvršavati upite iz Query Buildera, koji je standardni dio okoline Visual Studija i bit će objašnjen kasnije u poglavlju.



Iako je SQL složen jezik s mnogim detaljima koje je potrebno naučiti za optimiziranje upita, ova knjiga će vam pokazati četiri glavne SQL naredbe za dohvat, dodavanje, izmjenu i brisanje zapisa iz tablice.

SELECT naredba

SELECT je najsloženija naredba s ogromnim brojem opcija, a glavna joj je namjena dohvaćanje zapisa. Da ne bismo zaglibili previše u teoriju, mogućnosti SELECT naredbe bit će vam prikazane na primjerima. Pogledajte, za početak, najosnovniji oblik te naredbe:

```
SELECT * FROM Products
```

Znate li imalo engleskog jezika, trebali biste znati što ova naredba radi. Ona vraća sve zapise svih polja iz tablice *Products*.

Zvjezdica u gornjoj naredbi ne znači da želite sve zapise iz tablice, već sva polja. Umjesto zvjezdice mogli ste napisati:

```
SELECT ProductName, UnitPrice FROM Products
```

Takva naredba vratila bi sadržaj polja *ProductName* i *UnitPrice* svih zapisa u tablici. Polja koja želite vratiti odvajaju se zarezom (","), a ako želite vratiti sva polja iz neke tablice, koristit ćete zvjezdicu.

Dohvaćene zapise možete i poredati po određenom kriteriju. Ako biste, recimo, proizvode željeli poredati po cijeni tako da je najskuplji na vrhu, napisali biste:

```
SELECT * FROM Products ORDER BY UnitPrice DESC
```

III. DIO: DIJELOVI .NET-A

Ključan je ORDER BY dio poslije kojeg navodite polje po kojem želite sortirati i način sortiranja. Ako napišete DESC, tablica će biti sortirana po tom polju silazno odnosno najveće vrijednosti će biti na vrhu. Suprotno od DESC je ASC, a koristi vam za uzlazno sortiranje, da dobijete najmanje vrijednosti na vrhu.

```
SELECT * FROM Products ORDER BY ProductName ASC
```

Tako ste dobili sve zapise poredane abecedno po imenu proizvoda jer je ime tekstualno polje. Prethodni primjer zapise je poredao *brojčano* jer je polje *UnitPrice* decimalnog tipa.



Zapise možete sortirati i po više polja odvojite li uvjete zarezom. Recimo da imate neku tablicu koja sadržava polja Ime i Prezime. Ako želite poredati sve zapise prvo po prezimenu, a onda po imenu, napisat ćete:

```
SELECT * FROM Tablica ORDER BY Prezime ASC, Ime ASC
```

Ako imate više osoba s istim prezimenom, one će tada biti poredane abecedno po imenu.

I još jedan savjet: ASC je *defaultni* način sortiranja te je opcionalan, pa ga ne morate navoditi. Prethodni biste upit mogli napisati i ovako:

```
SELECT * FROM Tablica ORDER BY Prezime, Ime
```

Naravno, ako želite nešto silazno poredati, morat ćete napisati DESC.

Ponekad nećete htjeti vratiti sve zapise neke tablice, već samo određen broj. Za to će vam poslužiti ključna riječ TOP. Ona je posebno korisna, ako je koristite uz ORDER BY i sortiranje zapisa. Pogledajte kako bi izgledao SQL upit kojim biste vratili 10 najskupljih proizvoda:

```
SELECT TOP 10 * FROM Products ORDER BY UnitPrice DESC
```

Ovaj upit zapravo *selektira* sva polja prvih 10 zapisa iz tablice *Products*, i to kad su poredani s najskupljim proizvodom na vrhu.

Možete vratiti i određeni postotak zapisa iz tablice. Recimo, ako biste htjeli vratiti 50% zapisa iz neke tablice, što je korisno kada ne znate ukupan broj zapisa u tablici, napisali biste:

```
SELECT TOP 50 PERCENT * FROM Products
```

Nemojte da vas zbuni ova zvjezdica uz ključnu riječ TOP. Ako ne želite u prethodnom upitu vratiti sva polja iz tablice, već samo neke od njih, sjetite se primjera s početka i navedite željena polja:

```
SELECT TOP 50 PERCENT ProductName, UnitPrice FROM Products
```

Želite li izbrojati koliko zapisa ima u nekoj tablici, iskoristit ćete COUNT(*):

```
SELECT COUNT(*) FROM Products
```

Prethodni SQL upit vratit će vam broj zapisa u tablici *Products*, no naziv polja u tom slučaju neće imati ime. To je dobra prilika da naučite preimenovati polja koja dohvaćate. Samo poslije polja napišite "AS" i njegov novi naziv. U tom slučaju, naravno, ne možete koristiti zvjezdicu za odabir svih polja.

```
SELECT COUNT(*) AS UkupanBroj FROM Products
```

Isto možete primijeniti i na polja koja već imaju imena:

```
SELECT ProductName AS Ime, UnitPrice AS Cijena FROM Products
```

I još samo jedna sitnica: ako biste, recimo, željeli vratiti ID-eve svih dostavljača proizvoda, prvo bi vam pala na pamet naredba:

```
SELECT SupplierID FROM Products
```

No takvom naredbom vraćate vrijednost polja *SupplierID* iz svih zapisa. Što ako ste tim upitom samo htjeli vidjeti koji dostavljači dostavljaju proizvode? Dobili biste ogroman broj zapisa i morali biste ručno kroz sve njih proći i vidjeti čiji se ID-evi ponavljaju. Stoga je vrlo korisna ključna riječ DISTINCT. Ona će vam vratiti samo *različite* vrijednosti nekog polja.

```
SELECT DISTINCT SupplierID FROM Products
```

Tako biste vratili samo ID-eve dostavljača proizvoda bez ponavljanja tih ID-eva.

Postavljanje uvjeta

Kao što ste možda primijetili, gornji primjeri su veoma zgodni za različita vraćanja zapisa. No oni su dostatni samo za neke osnovne manipulacije i ne pružaju vam dovoljne mogućnosti. WHERE ključna riječ služi vam za postavljanje uvjeta koje dohvaćeni zapisi moraju ispunjavati. Ona se koristi i u SELECT naredbi, no i u UPDATE i DELETE naredbama, kao što će biti pokazano kasnije.

Osnovna sintaksa korištenja WHERE uvjeta izgleda ovako:

III. DIO: DIJELOVI .NET-A

```
SELECT * FROM Tablica WHERE uvjeti ORDER BY Polje
```

Dijelovi gornjeg upita “SELECT * FROM Tablica” i “ORDER BY Polje” nisu previše važni i služe jedino da vam pokažu u kojem dijelu SQL naredbe se pojavljuje WHERE – poslije FROM u kojem određujete tablicu iz koje vraćate rezultate i prije ORDER BY kojim sortirate rezultate.

Kao uvjet koji zapisi moraju ispunjavati možete postaviti neograničeno mnogo provjera. Kao i u nekim programskim jezicima s Basic sintaksom, možete koristiti operatore manje (“<”), veće (“>”), jednako (“=”), manje ili jednako (“<=”), veće ili jednako (“>=”) ili pak različito (“<>”). Provjere također možete stavljati u zagrade i odvajati ih logičkim operatorima poput AND, OR, NOT i XOR. Pogledajte primjer:

```
SELECT * FROM Products WHERE SupplierID = 1 AND ProductName = 'Chang'
```

Možete pokušati napraviti i malo složenije uvjete:

```
SELECT * FROM Products WHERE (SupplierID = 1 OR SupplierID = 2) AND NOT ProductName = 'Chang'
```

Ako radite s datumima, postoji niz predefiniranih funkcija koje vam pritom mogu pomoći. Tako, recimo, trenutni datum i vrijeme vraća funkcija *GetDate()* koju možete iskoristiti u upitu:

```
SELECT * FROM Orders WHERE RequiredDate <= GetDate()
```

Gornji upit vratit će sve narudžbe čija se dostava već trebala obaviti, znači prije sadašnjeg trenutka. Funkcija *Now()* vraća i sekunde pa je preciznost upita veća.

Veoma su korisne i funkcije koje iz nekog datuma vraćaju godinu, mjesec ili dan – *Year()*, *Month()* i *Day()*. Ako biste željeli napisati upit koji će vratiti sve narudžbe načinjene prije 5 godina na današnji datum (ako je danas 7.9.2004, tražite vijesti od 7.9.1999), napisali biste:

```
SELECT * FROM Orders WHERE (Year(OrderDate) = Year(GetDate()) - 5) AND (Month(OrderDate) = Month(GetDate())) AND (Day(OrderDate) = Day(GetDate()))
```

Gornji upit vraća sve narudžbe kojima je godina naručivanja jednaka trenutnoj godini minus 5 (dakle, prije 5 godina), mjesec im je jednak trenutnom mjesecu, a dan u mjesecu isti kao i danas.

Ako želite uspoređivati tekstualne nizove, poput polja *ProductName*, korištenje operatora za usporedbu uspoređivat će nizove abecedno i pazeći na velika i mala slova. Na svu sreću, na raspolaganju vam stoji LIKE operator. Njega možete smatrati usporedbom jednakosti, no s malo većim mogućnostima.

Korištenjem LIKE operatora ne trebate paziti na velika i mala slova, a možete koristiti i *wildcard* operatore, poput “%” ili “_”. Kao što i sama riječ znači, uspoređuje se vrijednost koja *nalikuje* na neku drugu. Pogledajte upit kojim tražite sve proizvode koje imaju “cho” u imenu:

```
SELECT * FROM Products WHERE ProductName LIKE '%cho%'
```

Tako ćete pronaći sve zapise koji bilo gdje u imenu imaju riječ “cho”. Pritom ta riječ može biti jedina u polju, a može biti nešto i prije ili poslije nje, zbog upotrebe *wildcarda* “%” koji označava *bilo koje vrijednosti*. Isto tako, može se spominjati i “CHO” ili “cHo”, a vaš će upit svejedno vratiti i te zapise, zbog načina rada LIKE operatora.

Drugi *wildcard* operator je “_”, i zamjenjuje jedan znak u nizu, za razliku od “%” koji zamjenjuje neodređeni broj znakova (od 0 do neograničeno mnogo).

```
SELECT * FROM Products WHERE ProductName LIKE '%c_o%'
```

Gornji upit vratio bi sve proizvode koji u naslovu imaju riječi poput “cpo”, “cho”, “cdo” i slično.

Iako postavljanje uvjeta može biti veoma složeno, ovdje prikazani upiti trebali bi vam biti dovoljni za osnovni rad s bazom podataka i njeno korištenje u aplikacijama. Naravno, ukoliko vam na pamet padne neka složenija ideja, nemojte se ustručavati pogledati u *Books Online* koji dolaze uz SQL Server ili potražiti na Webu.

INSERT naredba

Osnovna namjena INSERT naredbe je dodavanje novih zapisa u neku tablicu. Pogledajte kako izgleda njena osnovna sintaksa:

```
INSERT INTO Tablica (popis_polja) VALUES (popis_vrijednosti)
```

U gornjem upitu *popis_polja* označuje polja u koje želite ubaciti neke vrijednosti odvojene zarezom, a *popis_vrijednosti* vrijednosti za koje želite da poprime ta polja. Ako biste željeli ubaciti novu vijest u tablicu *Vijesti* napisali biste:

```
INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit,
UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued) VALUES ('Moj
novi proizvod', 1, 3, '10 komada', 15.0000, 30, 0, 10, 0)
```

Za razliku od SELECT naredbe koja vraća zapise koje ste zatražili, INSERT naredba ne vraća ništa, već samo izvršava dodavanje zapisa. Primijetite da u gornjem upitu niste specificirali vrijednost polja ProductID, jer je ono tipa *identity*, pa se za njega brine sama baza i automatski ga dodaje.

III. DIO: DIJELOVI .NET-A

I još samo kratka napomena: sintaksa INSERT naredbe veoma je jednostavna. Ubacujete li tekstualne nizove, stavite ih u polunavodnike. Ubacujete li pak brojčane ili logičke vrijednosti, ne trebate koristiti polunavodnike. Kod korištenja INSERT naredbe jedino morate paziti da vrijednosti koje ste napisali unutar VALUES dijela prate poredak polja – tj. u prvo polje ubacit će se prva vrijednost, u drugo polje druga vrijednost itd.

UPDATE naredba

UPDATE naredba izmjenjuje vrijednosti već postojećih zapisa u tablici. Njena osnovna sintaksa je sljedeća:

```
UPDATE Tablice SET Polje1 = Vrijednost1, Polje2 = Vrijednost2, ... WHERE uvjet
```

Primijetite da obavezno morate navesti uvjet koji zapisi moraju ispuniti da bi njihova vrijednost bila promijenjena. U slučaju da ne napišete uvjet i izostavite cijeli WHERE dio, izmijenit će se svi zapisi u tablici, što će vam vjerojatno donijeti više štete nego koristi.

Pri postavljanju uvjeta u UPDATE naredbi koristit će vam ID polje. Evo kako biste promijenili ime i cijenu proizvoda s ID-em 1 (vrijednosti ostalih polja ostale bi neizmijenjene):

```
UPDATE Products SET ProductName = 'Moj proizvod', UnitPrice = 10.5000 WHERE  
ProductID = 1
```

Isto kao INSERT naredba, UPDATE ne vraća nikakvu vrijednost, već samo izmjenjuje vrijednosti zapisa. U slučaju da nije pronađen niti jedan zapis koji odgovara uvjetima, ništa se neće izmijeniti, no vi nećete dobiti poruku o tome. SQL upit će se izvršiti, a na vama je da provjerite što je on zapravo napravio.

DELETE naredba

Vjerovali ili ne, DELETE naredba briše zapise iz neke tablice. Njena osnovna sintaksa je:

```
DELETE FROM Tablica WHERE uvjet
```

Primijetite da ovdje ne morate navesti koja polja brišete jer ona briše cijele zapise. Zato je bitan *uvjet* koji ima isti oblik kao i kod SELECT i UPDATE naredbi. Evo kako biste izbrisali proizvod s ID-em 79:

```
DELETE FROM Products WHERE ProductID = 79
```

No prije nego što se upustite u brisanje zapisa, imajte na umu da je to nepovratna akcija. Ne postoji nikakav *Undo* ili nešto slično čime biste mogli vratiti izbrisane podatke.

Spajanje više tablica

SQL vam omogućava spajanje povezanih tablica i dohvaćanje njihovih vrijednosti jednim upitom. Naprimjer, ako biste željeli spojiti tablice *Products* i *Suppliers*, koje su i logički povezane jer uz svaki proizvod postoji informacija o njegovu dostavljaču, napisali biste:

```
SELECT * FROM Products INNER JOIN Suppliers ON Products.SupplierID =  
Suppliers.SupplierID
```

Gornji upit dohvaća tablicu *Products* i dodaje joj tablicu *Suppliers* tako da ih spaja po poljima *SupplierID*. Primjerice, ako se u nekom retku tablica *Products* pojavljuje *SupplierID* 1, tom retku će biti dodan redak iz tablice *Suppliers* u kojem je *SupplierID* jednak 1. Tako uz svaki proizvod imate informacije o dostavljaču koji ju je dostavio. Rezultat spajanja tih tablica bit će nova tablica, koja će u sebi imati sljedeća polja:

```
ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice,  
UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued, SupplierID, CompanyName,  
ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax,  
HomePage
```

Kao što vidite, radi se o zbroju svih polja iz obje tablice. Prvo su dohvaćena polja iz prve tablice, tj. iz tablice *Products*, a njima su dodana sva polja iz tablice *Suppliers*. Zbog toga su i neki podaci redundantni odnosno ponavljaju se. Tako se ponavlja ID dostavljača u poljima *SupplierID*, jer su ona trebala biti jednaka za spajanje tablica.

Zato je mnogo prihvatljivije točno odrediti koja polja želite iz spojene tablice. Evo kako biste to napravili, uzevši u obzir samo osnovne informacije o proizvodima i njihovim dostavljačima:

```
SELECT Products.ProductID, Products.ProductName, Products.UnitPrice,  
Suppliers.CompanyName FROM Products INNER JOIN Suppliers ON Products.SupplierID =  
Suppliers.SupplierID
```

Rezultat toga upita prikazan je u tablici 9-1. Dohvatili ste samo polja koja će vam trebati u programu i tako ubrzali cijeli postupak izbacivši ona nepotrebna. Također, samo ćete prikazivati ime i cijenu proizvoda te ime tvrtke dostavljača, a ostalo vam nije potrebno.

Pri navođenju polja iz spojene tablice koje želite u konačnom rezultatu, morali ste navesti i tablicu u kojoj se nalaze. Isto možete koristiti i u jednostavnijim upitima. Naravno, ne morate, jer ako je u igri samo jedna tablica, podrazumijeva se otkud dolaze polja, ali samo da uočite sintaksu:

```
SELECT Products.ProductName, Products.UnitPrice FROM Products
```

III. DIO: DIJELOVI .NET-A

Tablica 9-1:

Kompliciranijim upitom dohvaćena su samo neka polja iz spojenih tablica (u tablici je prikazan samo izvadak dohvaćene tablice, i to zapisi s ID-evima 7, 8 i 9).

ProductID	ProductName	UnitPrice	CompanyName
7	Uncle Bob's Organic Dried Pears	30.0000	Grandma Kelly's Homestead
8	Northwoods Cranberry Sauce	40.0000	Grandma Kelly's Homestead
9	Mishi Kobe Niku	97.0000	Tokyo Traders

Spajanje tablica bit će vam veoma korisno tamo gdje je potrebno u što manje upita dohvatiti što više podataka. Ne možete si dopustiti komfor i upućivati gomile SQL upita – što više korisnika koristi vašu aplikaciju, baza će biti sve opterećenija i može doći do zagušenja. Spajanje relacijskih tablica zapravo je jedino rješenje tog problema, a omogućava vam pisanje mnogo urednijeg kôda.

Nakon što ste se upoznali s osnovama jezika SQL, slijedi prava stvar. U nastavku ćete se upoznati s osnovnim komponentama ADO.NET-a i naučiti kako iz .NET aplikacija raditi sa stvarnim podacima iz baza.

Komponente ADO.NET-a

Ukoliko već imate iskustva s programiranjem aplikacija koje pristupaju podacima, zasigurno ste upoznali preteču ADO.NET-a, naziva ADO. Središnji dio ADO arhitekture bio je *recordset* – radilo se o reprezentaciji podataka sličnoj nekoj tablici po kojoj ste se mogli kretati u svojoj aplikaciji, dohvaćati određene retke, stupce i slično. ADO.NET donosi niz novih mogućnosti, kao što su komponente *DataReader* i *DataSet* koje u potpunosti zamjenjuju *recordset* te on više ne postoji – to je prva prepreka koju morate svladati želite li kvalitetno iskoristiti mogućnosti ADO.NET-a.

Na *DataSet* se može gledati kao na složeniju verziju *recordseta* jer može u sebi sadržavati kompletnu reprezentaciju baze podataka; primjerice, više tablica, njihove relacije, primarne ključeve, *pogled* (*view*), omogućava sortiranje, filtriranje itd.

No najveća razlika između *DataSeta* i *recordseta* je u načinu njihova rada – dok je *recordset* za vrijeme svog korištenja stalno bio spojen na bazu podataka i tako konstantno trošio resurse (pogotovo ako bi ga programer zaboravio zatvoriti pa bi ostao u memoriji i nakon završetka programa), *DataSet* nije spojen na bazu i u sebi sadržava samo podatke koji su u njega stavljani. Podatke u *DataSetu* možete u svojoj aplikaciji mijenjati te, kad ste završili, sve spremi natrag u bazu. Više nema potrebe da veza s bazom bude stalno otvorena, što uvelike poboljšava performanse sustava.

Sve opisane komponente bit će kasnije detaljnije razrađene, a zasad opisi služe samo da si stvorite predodžbu i dobijete sliku mogućnosti ADO.NET-a.



Uz *DataSet* usko je vezana i *DataAdapter* komponenta. Ona predstavlja vezu na bazu podataka i zadužena je prvenstveno za popunjavanje *DataSeta*. Ona će i sve promjene načinjene u *DataSetu* spremi natrag u bazu podataka.

Na *DataReader* može se gledati kao na inačicu *recordseta* koja je sposobna isključivo prikazivati podatke iz baze podataka (tzv. *read-only* odnosno podaci se ne mogu mijenjati u *DataReaderu* te potom spremi natrag u bazu) i kretati se kroz dohvaćene zapise samo unaprijed (tzv. *forward-only* odnosno omogućava slijedno čitanje dohvaćenih zapisa bez mogućnosti vraćanja unatrag za jedan zapis – želite li se vratiti unatrag, jedino što vam se nudi je kretanje ispočetka). Unatoč svim navedenim nedostacima, *DataReader* može biti izvrstan izbor u većini slučajeva, pogotovo kad budete razvijali ASP.NET aplikacije za koje je važan brz pristup podacima.

Da biste dobili dojam odnosa, zgodno je napomenuti da *DataAdapter* za popunjavanje *DataSet* objekata interno koristi *DataReader* objekte.



Arhitektura

ADO.NET je, ponovimo, najnovija verzija Microsoftove tehnologije za univerzalan pristup podacima. To znači da je ADO.NET potpuno neovisan o izvoru podataka. On se temelji na XML-u, te je stoga potpuno svejedno odakle se podaci dovlače jer se oni interno spremaju u XML-u. Nakon što napišete programsku podršku i ostvarite funkcionalnost povezivanja i rada s bazom podataka, uz samo male promjene parametara možete raditi i sa sasvim drugim izvorom podataka.

To mogu biti baze smještene na SQL Serveru, poslužitelju Oracle, obične Access baze ili jednostavno XML datoteke – s tim podacima radite na gotovo identičan način, što predstavlja izuzetan dobitak, jer jednom kad svladate osnovne metode ADO.NET-a, svi ti izvori podataka vam stoje na raspolaganju.

Zahvaljujući nasljeđivanju, koje je osnova biblioteke .NET klasa, i ADO.NET klase nasljeđuju bazu funkcionalnost iz osnovnih klasa, što bitno pojednostavljuje programiranje. U tablici 9-1 navedeni su *namespaceovi* u kojima je pohranjena ADO.NET funkcionalnost.

III. DIO: DIJELOVI .NET-A

Tablica 9-2:
Namespaceovi koje koristi ADO.NET

Namespace	Opis
System.Data	sadržava osnovne objekte, kao što su <i>DataTable</i> , <i>DataColumn</i> , <i>DataRowView</i> ; to je temeljni namespace iz kojeg su izvedeni drugi
System.Data.Common	definira generičke objekte koje dijele svi <i>provideri</i> za pristup podacima, kao što su <i>DataAdapter</i> , <i>DataColumnMapping</i> ili <i>DataTableMapping</i> ; u većini slučajeva ovaj <i>namespace</i> nećete sami koristiti (jer se on koristi interno), već samo ako stvarate svoj <i>provider</i> za pristup podacima
System.Data.OleDb	<i>provider</i> za spajanje na sve OLE DB izvore podataka; postiže mnogo bolje performanse nego ODBC <i>provider</i>
System.Data.SqlClient	<i>provider</i> za spajanje isključivo na SQL Server verzija 7.0 ili novijih; ukoliko, dakle, koristite SQL Server, korištenjem ovog <i>provider</i> a postići ćete daleko bolje performanse nego korištenjem OLE DB <i>provider</i> a, jer on interno direktno koristi SQL Server API
System.Data.SqlTypes	klase za rad sa SQL Serverima tipovima podataka
System.Data.Odbc	<i>provider</i> za spajanje na sve ODBC izvore podataka – dostupan je kao besplatan dodatak, a može se preuzeti s Microsoftovih web-stranica



Želite li se s .NET-om spajati na Oracle bazu podataka, na raspolaganju vam stoji “.NET Managed Provider for Oracle”, besplatan dodatak koji se može preuzeti s Microsoftovih web-stranica. Njegovom instalacijom dobivate *System.Data.OracleClient* namespace u kojem se nalaze svi potrebni objekti za rad s Oracle bazama.

U nastavku ćemo objasniti najčešće korištene objekte ADO.NET-a – *Command*, *Connection*, *DataReader* i *DataAdapter*. Oni su definirani u svim klasama za pristup podacima, od *System.Data.OleDb* i *System.Data.Odbc* do *System.Data.SqlClient*.

Kao što vidite, za pristup podacima iz .NET-a ključno je odabrati *provider*. On, dakako, direktno ovisi o izvoru podataka – ukoliko koristite SQL Server, upotrijebit ćete *System.Data.SqlClient*. Ukoliko pak koristite neku drugu bazu podataka, upotrijebit ćete *System.Data.OleDb*, ako za tu bazu podataka imate dostupne *drivere* OLE DB, ili *System.Data.Odbc* – ako imate dostupne *drivere* ODBC.



Objekt *Connection*

Prvi korak u izradi programa temeljenog na nekom izvoru podataka je povezivanje s tim izvorom. Za to služi objekt *Connection*. Spajanje na bazu njegovim korištenjem uistinu je jednostavno i svodi se na definiranje tzv. *connection stringa* odnosno upute u kojoj su definirani izvor podataka, način spajanja i druge postavke.

U nastavku ćemo objasniti osnovne dijelove *connection stringa*. U većini slučajeva nećete se morati zamarati njihovim postavljanjem, već ćete *connection string* stvoriti direktno iz Visual Studio .NET okoline, kao što će biti pokazano kasnije u poglavlju.

Tablica 9-3:

Dijelovi *connection stringa* za spajanje na SQL Server bazu podataka (nije ih potrebno sve uvijek navoditi)

Naziv	Opis
Connection Timeout	vrijeme tijekom kojeg se pokušava spojiti na bazu podataka – ukoliko vrijeme istekne i veza se ne ostvari, podiže se iznimka; po <i>defaultu</i> je to vrijeme postavljeno na 15 sekundi
Data Source	ime ili IP adresa poslužitelja na kojem je smještena baza podataka
Initial Catalog	ime baze podataka koja se otvara
Integrated Security	određuje da li će se pri spajanju na server koristiti postavke korisničkog računa koji otvara konekciju (<i>true</i>) ili će se definirati korisničko ime i zaporka pod kojim će se ostvariti veza (<i>false</i>)
Password	zaporka za spajanje na bazu podataka; ukoliko se koristi <i>integrated security</i> , ne treba je navoditi
Persist Security Info	kad se postavi na <i>false</i> , svi kritični podaci za sigurnost (npr. zaporka) ne vraćaju se zajedno s otvorenom konekcijom
User ID	korisničko ime za spajanje na bazu podataka; ukoliko se koristi <i>integrated security</i> , ne treba ga navoditi

III. DIO: DIJELOVI .NET-A



U tablici 8-2 navedeni su svi dijelovi *connection stringa* za spajanje na SQL Server bazu podataka. No najčešći dio *connection stringa* je *Provider* koji određuje na koji se tip baze podataka spaja. Njega ćete koristiti uvijek, osim kad se spajate na SQL Server, jer se pri tom podrazumijeva da se radi o SQL Server bazi podataka.

Kombinirajući sve dijelove *connection stringa* vrlo lako možemo napraviti izraz pomoću kojeg biste se spojili na neku bazu podataka:

```
Data Source=imeServera;Initial Catalog=northwind;Integrated Security=true;
```

Prethodnim biste se *connection stringom* spojili na *northwind* bazu podataka smještenu na serveru imena *imeServera*. Za spajanje na bazu podataka koristili bi se podaci korisnika koji pokreće aplikaciju.

Objekt *Command*

Pojednostavljeno rečeno, objekt *Command* služi za izvršavanje različitih upita direktno na bazi podataka. To mogu biti obični SQL upiti ili pozivi spremljenih procedura, a unutar samih poziva mogu se definirati i različiti parametri koji se koriste za prsljeđivanje vrijednosti upitima.



Svi objekti koji će biti objašnjeni u ovom dijelu postoje i za OLE DB *provider*, i za ODBC *provider* i za SQL *provider*. Primjerice, objekt *Command* će se u svojoj OLE DB verziji zvati *OleDbCommand*, u ODBC verziji će se zvati *OdbcCommand*, a u SQL Server verziji *SqlCommand*. S njima se radi na potpuno isti način, a razlika je samo u njihovu internom načinu rada, što vas kao programera aplikacija u .NET-u ne treba zamarati.

Pogledajmo na primjeru kako bi izgledalo otvaranje veze prema bazi podataka te stvaranje *Command* objekta.

```
string connStr = "Data Source=mojServer;Initial Catalog=northwind;Integrated
    Security=true;";
string sqlUpit = "SELECT * FROM Orders";

SqlConnection conn = new SqlConnection(connStr);
```

9. POGLAVLJE: ADO.NET

```
conn.Open();
SqlCommand cmd = new SqlCommand(sqlUpit, conn);
```

U našem primjeru radimo sa SQL Serverom i, kao što vidite, za stvaranje veze prema bazi podataka koristi se objekt *SqlConnection*. Pri njegovu stvaranju za parametar se proslijeđuje *connection string* na temelju kojeg ADO.NET zna točno koju bazu treba otvoriti.

Da bi prethodni primjer radio, potrebno je dodati nazive korištenih *namespaceova* na početak programa. Kako koristimo objekte za rad sa SQL Serverom, na početku programa trebali bismo dodati:

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



Prethodni primjer za naredbu bazi podataka koristi običan SQL upit. Objekt *Command* pak može puno pomoći pri izvršavanju spremljenih procedura na SQL Serveru. Recimo da imamo proceduru sljedećeg sadržaja:

```
CREATE PROCEDURE getOrder
    @OID int
AS
    SELECT * FROM Orders WHERE OrderID = @OID
GO
```

Spremljena procedura ili *stored procedure* je SQL naredba (ili više njih) spremljena na SQL Serveru. Kako je interno pohranjena, njeno izvršavanje je optimizirano i izvršava se mnogo brže od običnih SQL upita koji se postavljaju na bazu jer je poslužitelj ne treba kompajlirati i već mu je poznat plan njena izvršavanja.



Radi se o jednostavnoj spremljenoj proceduri koja prima jedan parametar ID narudžbe i na temelju njega vraća odgovarajući zapis. Sad trebamo pozivu procedure dodati i jedan parametar, a to ćemo obaviti zahvaljujući objektu *SqlParameter*. No prije toga moramo dodati još jednu liniju u kojoj ćemo obavijestiti ADO.NET da ne izvršavamo direktno neki SQL upit, već da pozivamo spremljenu proceduru. Pretpostavimo da već postoji otvorena veza s bazom podataka u objektu *conn*.

III. DIO: DIJELOVI .NET-A



U primjerima nećemo i pozivati objekt *Command* i dohvaćati njegove rezultate jer će nam za to biti potrebno razumijevanje *DataReadera* i drugih objekata, što ćemo obraditi kasnije. Zasad je jedino bitno njihovo stvaranje i namještanje parametara.

```
string sqlUpit = "getOrder";

SqlCommand cmd = new SqlCommand(sqlUpit, conn);
cmd.CommandType = CommandType.StoredProcedure;
```

Koristili smo *CommandType* svojstvo u koje smo postavili konstantu i istoimenog objekta. Tako ADO.NET zna da se radi o pozivu spremljene procedure jer sam SQL upit ne bi bio ispravan – radi se samo o nazivu spremljene procedure.



Kod pozivanja spremljenih procedura postoje ulazni i izlazni parametri. Ulazni parametri su oni koji se prosleđuju spremljenoj proceduri i koje ona koristi za obavljanje određenih akcija, dok su izlazni parametri oni koje spremljena procedura vraća programu kao rezultat njihovih akcija. Izlazni parametri su najčešće obične vrijednosti varijabli; izlaznim parametrima ne smatraju se zapisi iz tablica.

Prethodni primjer ne bi bio ispravan jer spremljena procedura *getOrder* koristi jedan ulazni parametar. U programu trebamo zato stvoriti parametar i proslijediti ga spremljenoj proceduri. Za to ćemo iskoristiti spomenuti objekt *SqlParameter*.

```
SqlParameter parametar = cmd.Parameters.Add(new SqlParameter("@OID", SqlDbType.Int, 4));
```

Primijetite da objektu *cmd* koji sadrži komandu bazi podataka dodajemo parametar u kolekciji *Parameters*. Radi se o novom objektu koji ima ime *@OID* i tipa je *Int*. Kao što ćete vidjeti kasnije, najčešće nećete trebati uopće paziti kojeg je tipa parametar koji stvarate, jer ćete sve obaviti iz radne okoline Visual Studija, koji će ga sam namjestiti na odgovarajuću vrijednost.

Kao što smo rekli, parametri mogu biti ulazni i izlazni. Smjer samog parametra možete namjestiti na način prikazan u sljedećem primjeru, a u tablici 9-3 možete vidjeti sve moguće *smjerove* parametara.


```
parametar.Direction = ParameterDirection.Input;
```

Smjer	Opis
Input	ulazni parametar čija se vrijednost proslijeđuje spremljenoj proceduri ili SQL upitu
Output	izlazni parametar koji je u spremljenoj proceduri definiran kao "OUTPUT" i čija se vrijednost vraća natrag u program
InputOutput	parametar koji može služiti i kao ulazni i kao izlazni parametar, što znači da se njegova vrijednost šalje spremljenoj proceduri, ali je njegova nova vrijednost dostupna nakon izvršavanja procedure
ReturnValue	izlazni parametar koji vraća proceduru; može se imati samo jedan za spremljenu proceduru

Tablica 9-4:
Moguća svojstva
ParameterDirection kolekcije
koja određuju smjer
parametra

I na samom kraju, ukoliko koristite ulazni parametar, morate postaviti njegovu vrijednost. To radi-
te tako da mu namjestite svojstvo *Value*:

```
parametar.Value = 50;
```

Ukoliko pak imate izlazni parametar, nakon pozivanja spremljene procedure ili SQL upita, njego-
vu vrijednost možete iščitati:

```
string rezultat = cmd.Parameters("@IzlazniParametar").Value;
```

Objekt *DataReader*

DataReader je, kao što smo spomenuli na početku teksta, jednostavan optimiziran objekt, prila-
gođen isključivo čitanju zapisa prema naprijed (što znači da možete ići samo zapis po zapis una-
prijed, bez mogućnosti vraćanja unatrag). To bi vam u većini slučajeva trebalo biti dovoljno, ako
želite samo dohvatiti određene zapise iz baze, nad njima napraviti određene akcije i ispisati u
svojoj aplikaciji.

III. DIO: DIJELOVI .NET-A

Povratne vrijednosti, gdje ste?

Spremljene procedure u SQL Serveru mogu vraćati vrijednosti s naredbom **RETURN**. Pogledajte jednostavnu proceduru u sljedećem primjeru (Napomena: radi se o izmišljenim podacima iz izmišljene tablice, a procedura dohvaća identifikator zapisa zadnje narudžbe.):

```
CREATE PROCEDURE GetLastOrderID
AS
    DECLARE @LastID INT
    SET @LastID = (SELECT TOP 1
        OrderID FROM Orders ORDER BY
        OrderDate DESC)
    RETURN @LastID
GO
```

Da biste dohvatili vrijednost koju vraća spremljena procedura, trebate stvoriti parametar imena "RETURN VALUE" i namjestiti mu smjer na *ReturnValue*:

```
SqlParameter param =
    cmd.Parameters.Add(new
        SqlParameter("RETURN VALUE",
            SqlDbType.Int, 4));
param.Direction =
    ParameterDirection.ReturnValue;
```

Prethodni primjeri, dakako, rade samo na SQL Serveru.

Evo kako biste napunili *DataReader* objekt i pročitali sve dohvaćene zapise. Pretpostavimo da već postoji objekt *Command* koji dohvaća određene zapise (recimo, najobičnija "SELECT * FROM tablica" naredba).

```
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
    // čitanje pojedinih zapisa
}
```

Ovdje smo prvi put pokrenuli objekt *Command* – izvršili smo naredbu *ExecuteReader()* koja dohvaća podatke prilagođene *DataReader* objektu i koje možemo direktno u njega učitati.

Način rada je veoma sličan *recordset* objektu iz starog ADO-a: u *while* petlji čitamo sve zapise iz *DataReadera*. Ona će se izvršavati sve dok ima zapisa u *DataReaderu* jer se naredbom *Read()* automatski pozicioniramo na naredni zapis.

Želite li dohvatiti vrijednost nekog stupca pojedinog retka, možete koristiti indeks ili ime tog stupca. Primjerice, ukoliko ste izvršili naredbu “SELECT ime, prezime FROM osobe”, na raspolaganju imate dva stupca. Njih možete dohvatiti indeksima 0 i 1 ili imenima “ime” i “prezime”.

```
string vrijednost;
while (dr.Read())
{
    vrijednost = dr[0].ToString();
    vrijednost = dr["ime"].ToString();
}
```

Objekt *DataSet*

Na *DataSet* možete gledati kao na kopiju podataka iz baze podataka spremljenu u memoriji radi bržeg pristupa. To može biti samo jedna tablica ili više tablica, a one čak mogu biti dohvaćene iz različitih baza podataka. Radi se, dakle, o kompletnoj kopiji podataka koja je u potpunosti odvojena od svog izvora i potpuno neovisna o njemu.

Radeći s *DataSet* objektom ne komunicirate direktno s bazom podataka. Tek nakon što obavite sve operacije nad *DataSetom* i njegovim podacima, možete te podatke poslati natrag na njihov izvor na obradu, što može uključivati mijenjanje originalnih podataka u bazi, brisanje, ili dodavanje novih.

Objekt *DataSet* nije ograničen samo na rad s bazama podataka, već može učitavati i spremiti podatke i iz XML datoteka, s kojima se radi jednako kao s bazama podataka.



Objekt *DataTable*

Svaki objekt *DataSet* sadržava kolekciju objekata *DataTable*. Kao što i samo ime kaže, radi se o kolekciji tablica spremljenih u *DataSetu*, a to može biti jedna ili više njih. Na objekte *DataTable* može se gledati kao na tablice u nekoj bazi podataka ili u drugom izvoru podataka.

Kako je cilj razvojnog tima ADO.NET-a bio učiniti objekt *DataSet* ključnim za rad s podacima u .NET-u, dodane su mu mnoge korisne mogućnosti. Tako tablicama (objektima *DataTable*) možete dodavati primarne ključeve (*PrimaryKeys*) ili ih međusobno povezivati korištenjem *DataRelations* kolekcije. S tim mogućnostima objekt *DataSet* doista postaje punopravna kopija sadržaja baze, spremljena u memoriji.



III. DIO: DIJELOVI .NET-A

Objekte *DataTable* možete stvarati programski – stvorite novu tablicu, dodate joj stupce, dodate joj retke sadržaja i slično – ili ih puniti iz baze podataka, što je ipak češći način rada. U nastavku slijedi primjer punjenja *DataSeta* odnosno jedne njegove tablice sadržajem iz baze podataka.

Za to ćemo iskoristiti *DataAdapter* – primjer će dohvatiti sve podatke iz neke tablice i proslijediti ih nekoj kontroli koja može prikazivati podatke, primjerice *Repeater* kontroli u ASP.NET aplikacijama (sljedeći primjer će biti detaljnije objašnjen u narednom poglavlju).

```
SqlConnection sqlConn = new
    SqlConnection("server=(local);database=Northwind;Integrated Security=true");

SqlDataAdapter sqlComm = new SqlDataAdapter("SELECT * FROM Orders", sqlConn);

DataSet ds = new DataSet();
sqlComm.Fill(ds);

Kontrola.DataSource = ds;
Kontrola.DataBind();
```

Ključna naredba je *sqlComm.Fill(ds)* kojom se puni *DataSet* podacima iz baze koji se dohvaćaju preko *DataAdaptera*. Nakon toga, nekoj kontroli za izvor podataka postavljamo stvoreni *DataSet*.

Svakoj tablici u *DataSet* objektu pristupate preko kolekcije *Tables*. Pritom možete koristiti indeks tablice ili njeno ime, primjerice:

```
ds.Tables[0]
ds.Tables["Orders"]
```

Objekt *DataColumn*

Kao što i svaka tablica u bazi podataka ima svoje stupce s odgovarajućim svojstvima, poput tipa podataka koji sadržava, duljine i slično, tako i objekt *DataSet* odnosno svaki njegov objekt *DataTable* sadržava niz stupaca odnosno objekata *DataColumn*.

Stupci nekog objekta *DataTable* predstavljaju njegovu strukturu, dakle bez podataka. Svaki stupac tako ima niz atributa koji utječu na način njegova rada. Primjerice, ukoliko je svojstvo *AllowDBNull* postavljeno na *true*, stupac će moći primati i *null* vrijednosti. Tip podataka nekog stupca sprema se u atribut *DataType*, a ukoliko želite onemogućiti promjene sadržaja tog stupca, postaviti ćete svojstvo *ReadOnly* na *true*.

Kao što je već rečeno, objekte *DataTable* možete puniti iz baza podataka ili ih direktno stvarati. U narednom primjeru bit će prikazano stvaranje tablice s dva stupca i njeno dodavanje u objekt *DataSet*.

```

DataColumn Stupac1 = new DataColumn();
DataColumn Stupac2 = new DataColumn();

Stupac1.DataType = System.Type.GetType("System.Int32");
Stupac1.ColumnName = "OsobaID";
Stupac1.AutoIncrement = true;
Stupac1.ReadOnly = true;

Stupac2.DataType = System.Type.GetType("System.String");
Stupac2.ColumnName = "OsobaImePrezima";

DataTable Tablica = new DataTable("Osobe");
Tablica.Columns.Add(Stupac1);
Tablica.Columns.Add(Stupac2);

DataSet ds = new DataSet();
ds.Tables.Add(Tablica);

```

Dakle, cilj nam je bio stvoriti tablicu koja će služiti pohranjivanju informacija o osobama te će imati dva stupca – prvi će služiti spremanju jedinstvenog identifikatora osobe, a drugi je za njezino ime i prezime.

Zato smo na početku stvorili dva stupca – prvi je bio tipa *Int32*, ime smo mu postavili na *OsobaID*, postavili smo mu i svojstvo *AutoIncrement* na *true*, što znači da će se identifikator osoba automatski povećavati kako budemo dodavali nove osobe i mi ga nećemo morati mijenjati, te smo stoga i zabranili njegovo mijenjanje postavljenjem svojstva *ReadOnly* na *true*.

Na sličan način smo namjestili i svojstva drugog stupca – tip podataka koji će sadržavati je *string*, jer je predviđeno da u njega upisujemo ime i prezime osobe, a shodno tome smo mu postavili i ime.

Sljedeći korak bio je stvaranje nove tablice, tj. objekta *DataTable*. Novu tablicu smo nazvali *Osobe*, a dodali smo joj dva već prije stvorena stupca. Na samom smo kraju stvorili objekt *DataSet* kojem smo dodali netom stvorenu tablicu.

Objekt DataRow

Drugi glavni dio objekta *DataTable*, uz spomenuti *DataColumn*, jest objekt *DataRow*. Radi se pojedinim zapisima u tablici podataka, a oni služe za mijenjanje, dodavanje i brisanje podataka.

III. DIO: DIJELOVI .NET-A

Verzije i greške u redcima

Svaki *DataRow*, osim podataka, sadržava i informacije o greškama u retku i njegovim verzijama. Verzije sadržaja nekog retka mogu se pratiti pomoću *RowState* svojstva i *GetChanges* i *HasChanges* metoda pa tako redak može biti u stanju *Added* (redak je dodan u kolekciju), *Deleted* (pozvana je metoda za brisanje), *Detached* (redak nije dio kolekcije), *Modified* (njegov sadržaj je promijenjen) ili *Unchanged* (sadržaj je nepromijenjen). Želite li spremati sve promjene načinjene u retku, pozvat ćete metodu *AcceptChanges*. Imajte na umu da se pozivom te metode ne mijenjaju podaci originala odnosno u samoj bazi podataka ili ko-

ji ste već izvor koristili – tek se pozivom *Update* metode *DataAdapters* te promjene spremaju, i to samo one u redcima nad kojima je pozvana metoda *AcceptChanges*.

Želite li pak saznati više o greškama koje su se pojavile u retku odnosno njegovu sadržaju, provjerit ćete svojstvo *HasErrors*. Ukoliko ono ima vrijednost *true*, možete pozvati metodu *GetColumnError* da dobijete jedan stupac s greškom ili *GetColumnsInError* da dobijete kolekciju stupaca s greškom.

DataTable, dakle, sadržava kolekciju redaka odnosno kolekciju objekata *DataRow*. S tom kolekcijom se radi kao i sa svakom drugom – želite li dodati novi redak, iskoristit ćete metodu *Add*, a želite li pristupiti pojedinom retku iz kolekcije, iskoristit ćete svojstvo *Item*.

Razlike između *DataReader*a i *DataSeta*

Kao što je već više puta naglašeno, svi podaci koji se koriste kroz ADO.NET nisu konstantno spojeni na svoj izvor, tj. nazivamo ih *disconnected*. Pojednostavljeno rečeno, pristup podacima može se svesti na dva modela – korištenjem *DataReader*a i korištenjem *DataSeta*.

Pri korištenju *DataSeta* podaci se učitavaju u lokalni spremnik, zatim čitaju i mijenjaju te na kraju sinkroniziraju s izvorom podataka. Veza s bazom podataka otvara se na početku, *DataSet* se puni (tj. njegov *DataTable*) i aplikacija može s tom kopijom podataka raditi što želi, a veza s bazom se zatvara. Resursi baze podataka se tako ne troše i ona je slobodna obavljati druge poslove.

DataReader pak ne dopušta mijenjanje tih podataka ili višekratno korištenje – podaci se čitaju samo jednom i već se pri čitanju narednog retka svi prošli podaci brišu iz memorije. No za razliku od *DataSeta*, pri korištenju *DataReader*a je stalno otvorena veza s bazom podataka. Ukoliko bi vaša aplikacija dopuštala korisniku korištenje *DataReader* objekta, to bi moglo predstavljati potencijalan problem – baza podataka vraća redak po redak te čeka na sljedeću naredbu *DataReader*a. Sto-

ga je *DataReader* najbolje koristiti u situacijama kad vam je potreban brz pristup nekim podacima i kad ih sve iščitavate u nekoj petlji, bez interakcije korisnika, te kad ih ne planirate više puta upotrebljavati.

Objekt *DataView*

I na samom kraju, ostao nam je još jedino objekt *DataView*. Njegova namjena je da povezuje podatke s formularima i kontrolama u aplikaciji, a može se koristiti i za pretraživanje, filtriranje, sortiranje i uređivanje podataka.

DataView se temelji na objektu *DataTable* i služi kao poveznica između njegovih podataka i polja predviđenih podatke na formularu, bilo u web-aplikaciji ili prozorskoj aplikaciji. Kao što i samo ime objekta kaže, on nudi stvaranje drugačijih *pogleda* na podatke i njihovo prilagođavanje. Primjerice, možete stvoriti jedan *DataView* koji će prikazivati sve zapise iz objekta *DataTable* koji imaju cijenu veću od 10 i drugi *DataView* koji će prikazivati sve s manjom cijenom zahvaljujući mogućnosti filtriranja redaka.

Primijetite da se pri korištenju objekta *DataView* podaci dohvaćaju samo jednom, i to u *DataTable* objekt, a kasnije se u memoriji filtriraju i obrađuju u drugim *DataView* objektima, što uvelike olakšava posao bazi podataka koja podatke dohvaća samo jednom.



Rad s ADO.NET-om

U nastavku ćemo objasniti kako lako napraviti aplikaciju koja radi s podacima. S nekoliko klikova mišem stvorit ćemo vezu s bazom podataka i prikazivati podatke u *DataGrid* kontroli te uz malo kôda te podatke i uređivati.

Za početak, stvorite običnu prozorsku aplikaciju, kao što ste naučili u prethodnom poglavlju. Ona će nam biti temelj za sve buduće akcije.

U narednim primjerima bit će pokazan praktičan rad ADO.NET-a sa SQL Serverom te će stoga biti korištene komponente *SqlConnection*, *SqlDataAdapter* i *SqlCommand*, no imajte na umu da ćete na isti način raditi i s ostalim komponentama, ako se odlučite za rad ODBC, OLE DB ili Oracle bazama podataka.

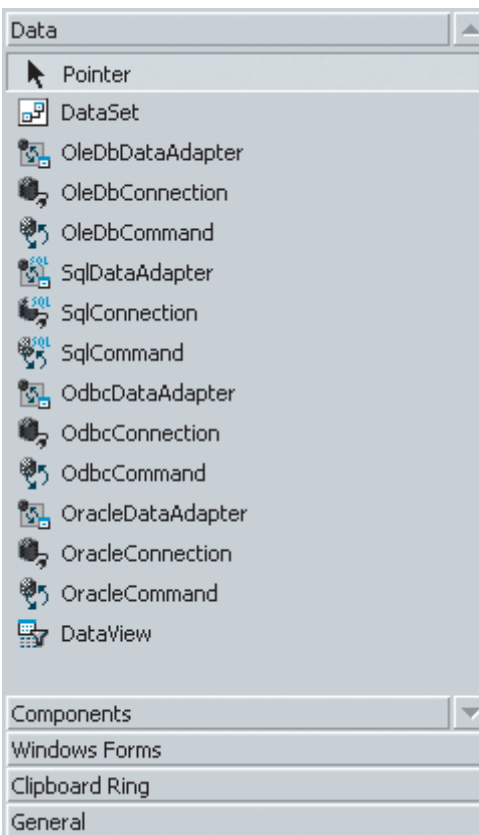


III. DIO: DIJELOVI .NET-A

Stvaranje veze s bazom

Da biste radili s bazom podataka, trebat ćemo stvoriti vezu na nju. U lijevom dijelu sučelja Visual Studija .NET nalazi se *Toolbox* prozor i u njemu *Data* okno, prikazano na slici 9-1.

Slika 9-1:
Data dio Toolbox prozora sadrži komponente za rad s bazama podataka.



Iz *Data* dijela povucite na radnu površinu aplikacije *SqlConnection* objekt, čime ga dodajete u aplikaciju. Kako se ne radi o kontroli, već o komponenti, ona će biti prikazana u posebnoj površini ispod aplikacije, kao što je prikazano na slici 9-2, i bit će joj dodijeljeno početno ime *sqlConnection1*. Jednostavnosti radi, svim ćemo u aplikaciju dodanim komponentama ostaviti početna imena.

Sljedeći korak je namještanje postavki veze s bazom. Označite dodanu komponentu *sqlConnection1* i prebacite se u *Properties* prozor. U njemu ćete pronaći *ConnectionString* svojstvo – kliknite na njega i u padajućem izborniku odaberite opciju *New Connection*. Otvorit će vam se prozor poput onog prikazanog na slici 9-3.

9. POGLAVLJE: ADO.NET



Slika 9-2:
SqlConnection kontrola pojavit će se ispod radne površine aplikacije.



Slika 9-3:
Prozor "Data Link Properties" služi za namještanje postavki veze s bazom podataka.

III. DIO: DIJELOVI .NET-A



Želite li vidjeti na koje se sve tipove baza podataka možete spajati odnosno koji su vam sve *provideri* dostupni, kliknite na karticu *Provider* prozora “Data Link Properties” (na slici 9-3).

U našem primjeru spajat ćemo se na lokalni SQL Server i njegovu Northwind bazu podataka. Stoga u prvom koraku upišite ili odaberite ime servera iz padajućeg izbornika. U drugom koraku možete upisati korisničko ime i zaporku za spajanje na bazu ili odabrati korištenje *Integrated Securityja*. Treći korak služi za odabir baze na koju ćete se spajati.

Slika 9-4:

Postavljene sve opcije za spajanje na bazu podataka



Nakon što ste postavili sve opcije, obavezno provjerite njihovu ispravnost – kliknite na gumb *Test Connection*. Ukoliko vam se pojavi poruka “Test connection succeeded”, sve je u redu i možete nastaviti s programiranjem.

Jednom načinjenu vezu s bazom podataka možete vrlo lako isprobati u *Server Explorer* prozoru koji se nalazi odmah uz *Toolbox* prozor s komponentama za aplikaciju. U njemu možete vidjeti strukturu baze, pogledati koje spremljene procedure i tablice sadržava, saznati njihova polja, tj. stupce, te dohvatiti njihov sadržaj – samo dvaput kliknite na ime tablice u *Server Exploreru*. Iskušajte sami i uvjerit ćete se da vam razvojna okolina Visual Studija pruža sve potrebno za rad, a više ne trebate u drugim programima petljati po bazi i informirati se o njihovom sadržaju ili proučavati koje stupce sadrže da ih uključite u svoje SQL upite. Ukoliko pak radite sa SQL Serverom, iz *Server Explorera* možete stvarati nove spremljene procedure, poglede ili funkcije – samo kliknite desnim gumbom miša i odaberite odgovarajuću opciju.



Zatvorite prozor za namještanje postavki veze klikom na OK. U *ConnectionString* svojstvu u *Properties* prozoru moći ćete pronaći nešto poput sljedeće generirane postavke za spajanje na bazu podataka:

```
workstation id=IME_RACUNALA;packet size=4096;integrated security=SSPI;data
source="IME_POSLUZITELJA";persist security info=False;initial catalog=Northwind
```

Time ste stvorili vezu s bazom podataka. U sljedećim koracima možete nastaviti programiranje i prikazivanje podataka iz baze u aplikaciji.

Upravo smo iskoristili razvojnu okolinu Visual Studija .NET za obavljanje jednostavnih akcija poput stvaranja veze s bazom, a on je u pozadini sam generirao programski kôd. Stoga je preporučljivo nakon svake ovdje objašnjene akcije prebaciti se u kôd aplikacije te potražiti i pročitati generirani kôd koji se u ovom slučaju skriva unutar regije “Windows Form Designer generated code”.



Prikaz podataka u *DataGrid* kontroli

Da biste iskoristili načinjenu vezu s bazom za dohvat podataka, trebat će nam jedna komponenta *SqlDataAdapter*. Baš kao što ste učinili i s komponentom *SqlConnection* povucite i nju na radnu površinu, što će stvoriti novi objekt *sqlDataAdapter1*.

III. DIO: DIJELOVI .NET-A

No pritom će vas dočekati čarobnjak koji će vam pomoći u dohvaćanju podataka. Otvorit će se prozor, prikazan na slici 9-5, koji predstavlja prvi korak definiranja *DataAdaptora*.

Slika 9-5:
Prvi korak
čarobnjaka za
definiranje
DataAdaptora



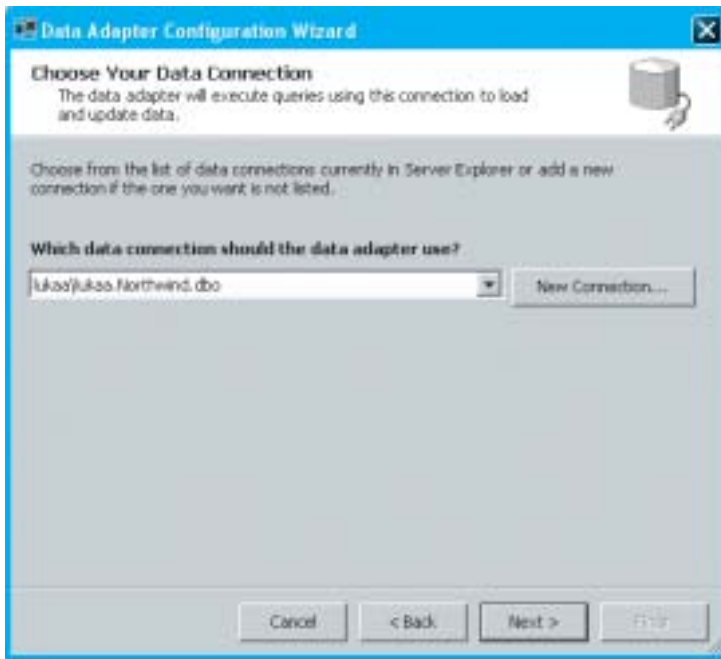
Kliknite na *Next* i doći ćete na sljedeći korak stvaranja *DataAdaptora*. U njemu odabirete vezu na bazu podataka. Imate mogućnost stvaranja nove veze klikom na *New Connection* gumb ili odabira postojeće iz padajućeg izbornika. Kako smo mi netom stvorili novu vezu, iskoristit ćemo nju – odaberite je iz padajućeg izbornika kao na slici 9-6.

Sljedeći korak predstavlja izbor komunikacije s bazom podataka – on se može vršiti preko običnih SQL naredbi, preko spremljenih procedura koje ćete tek stvoriti ili onih već stvorenih. Jednostavnosti radi, napisat ćemo svoju SQL naredbu za dohvat podataka, a vi možete eksperimentirati i s drugim izborima.

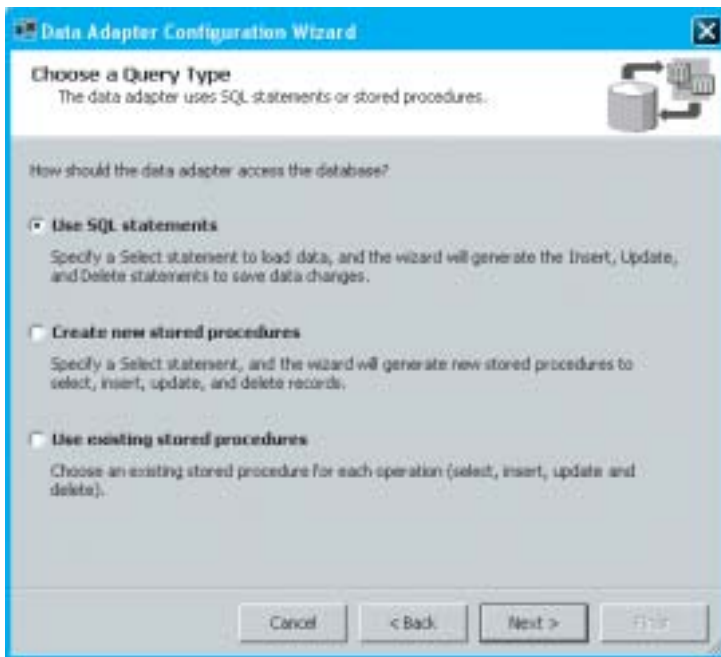
U tekstualno polje upišite SQL upit pomoću kojeg želite dohvaćati podatke iz baze:

```
SELECT ProductID, ProductName, UnitPrice FROM Products
```

9. POGLAVLJE: ADO.NET



Slika 9-6:
Odabir veze s bazom
podataka



Slika 9-7:
Odabir načina komu-
nikacije s bazom
podataka

III. DIO: DIJELOVI .NET-A

Zidam kuću trokatnicu...

U prozoru prikazanom na slici 9-9 možete kliknuti na Query Builder i pojaviti će vam se prozor za složeniju izradu upita. Takav prozor je standardan za Visual Studio .NET pa do njega možete doći iz različitih dijelova aplikacije i pri obavljanju različitih poslova s bazom podataka. Isti takav prozor možete koristiti i u Microsoft Accessu za stvaranje upita ili čak Reporting Services alatu za izradu izvještaja.

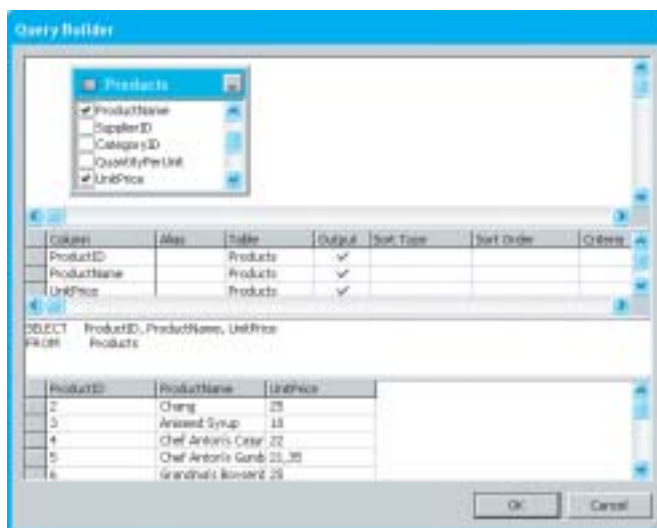
No nije to sve toliko važno – najvažnije je da vam on može uvelike pomoći pri izradi upita. Podijeljen je na 4 dijela. U prvom gornjem dijelu nalaze se tablice koje su uključene u vaš upit. Kliknete li taj dio desnim gumbom miša, možete odabrati opciju *Add Table* i dobit ćete popis svih tablica u bazi koje možete iskoristiti pri izgradnji upita. Nakon što dodate koju

tablicu, polja koja želite uključiti među dohvaćenim podacima jednostavno uključite klikom na *checkbox* u prozorčiću s tablicom.

U drugom dijelu Query Buildera možete postaviti dodatne kriterije nad stupcima. No, ukoliko ste vični ručnom pisanju SQL upita, najkorisniji će vam biti treći dio u kojem možete direktno upisati naredbu.

Kliknete li bilo gdje unutar Query Buildera desnim gumbom miša možete odabrati opciju *Run*, što će rezultirati izvršavanjem vašeg upita nad bazom podataka i dohvaćanjem podataka u četvrti dio Query Buildera. Tako možete odmah provjeriti svoj upit. Kad završite s izradom SQL upita u Query Builderu kliknite na OK i vratit ćete se u prozor na slici 9-9.

Slika 9-8:
Query Builder
može uvelike
pomoći pri
izgradnji SQL
upita

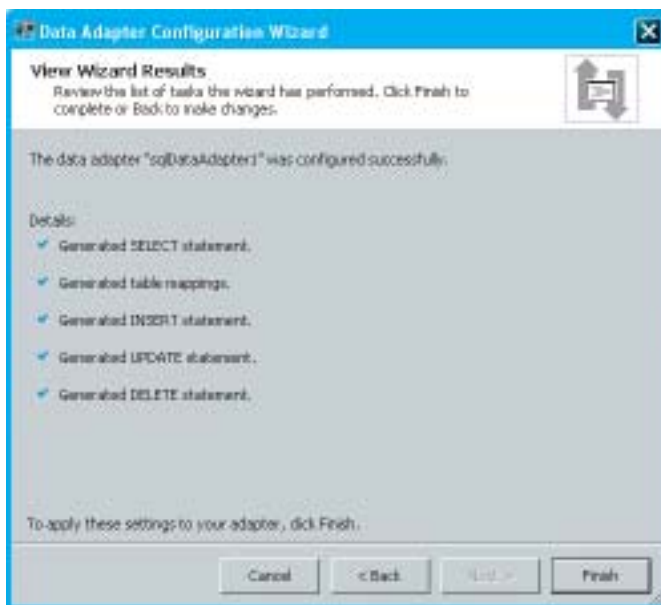


9. POGLAVLJE: ADO.NET



Slika 9-9:
*SQL upit izrađen ručno
ili uz pomoć Query
Buildera*

Nakon što ste izgradili upit u prozoru na slici 9-9, možete kliknuti na *Next*. Time dolazite do zadnjeg koraka i generiranja kompletnog kôda automatski. Za vas će se, ako je sve u redu, stvoriti komande za izvršavanje SELECT, INSERT, UPDATE i DELETE naredbi nad podacima u bazi. Kasnije



Slika 9-10:
*Zadnji korak defini-
ranja DataAdaptera*

III. DIO: DIJELOVI .NET-A

ćete vidjeti kako te naredbe iskoristiti – čak nećete ni morati znati da one u pozadini postoje, jer će se automatski pozivati izmjenom podataka u aplikaciji. Kliknite na *Finish*.



Nakon definiranja *DataAdaptera*, uvijek je korisno provjeriti ispravnost njegova rada. U Visual Studiju kliknite na opciju *Data – Preview Data*. Otvorit će vam se prozor u kojem možete prikazati podatke tako da kliknete na gumb *Fill Dataset* i doista se uvjerite radi li sve ispravno.

Da biste dohvaćene podatke iz baze podataka prezentirali u svojoj aplikaciji, trebat ćete ih uobličiti u *DataSet* objekt. Naravno, ni to nećete morati raditi sami, već ćete prepustiti Visual Studiju da za vas generira *DataSet* objekt.

Odaberite opciju *Data – Generate DataSet* i otvorit će vam se prozor kao na slici 9-11. U njemu morate odabrati koje ćete podatke dodati u novi *DataSet*. Dosad smo stvorili samo jedan *DataAdapter* pa ćemo njegove podatke prebaciti u *DataSet*. Kao što ste mogli vidjeti ranije, radi se o tablici *Products* iz baze *Northwind*.

Slika 9-11:
Automatsko generiranje objekta DataSet iz Visual Studija



Klikom na OK, među komponentama aplikacije ćete sad imati objekte *sqlConnection1*, *sqlDataAdapter1* i *dataSet11*.

9. POGLAVLJE: ADO.NET

Pri generiranju objekta *DataSet*, među datotekama u vašem projektu stvorit će se i istoimena datoteka s ekstenzijom XSD (u našem slučaju, "DataSet1.xsd") koja sadržava shemu podataka iz *DataSeta*.



Sad možemo podatke dohvaćene u objekt *DataSet* prikazati u aplikaciji. Za to će nam poslužiti *DataGrid* komponenta. Nju možete pronaći u dijelu *Windows Forms* prozora *Toolbox*. Povucite je na radnu površinu i po volji promijenite veličinu.

Označite zatim netom dodanu komponentu *DataGrid* i prebacite se u prozor *Properties*. U njemu pronađite svojstvo *DataSource* (nalazi se pod *Data*) i iz padajuće liste odaberite podatke koje želite da taj *DataGrid* prikazuje. Ako ste sve radili kao i mi, u izborniku će vam biti ponuđen *dataSet1.Products*.

Nakon što odaberete *DataSource*, stupci *DataGrida* automatski će se promijeniti i poprimiti vrijednosti prema shemi podataka iz *DataSeta*. No pokušate li sada pokrenuti aplikaciju, vidjet ćete da se podaci ipak još ne pokazuju u *DataGridu*. To je i logično – dosad ste tek povezali *DataGrid* s izvorom podataka, no te podatke još niste dohvatili.

Kliknite dvaput na vašu aplikaciju ili se prebacite unutar metode *Form_Load* u kodu. Tamo je potrebno upisati sljedeći kôd:

```
sqlDataAdapter1.Fill(dataSet1);
```

Dakle, *dataSet1* s kojim je i povezan *DataGrid* u aplikaciji je inicijalno prazan. Po učitavanju aplikacije trebamo ga napuniti, a to činimo metodom *Fill* našeg *DataAdaptora* u kojem smo definirali kako dohvatiti podatke. Pokrenete li sada aplikaciju, dobit ćete nešto kao na slici 9-12.

ProductID	ProductName	UnitPrice
2	Chang	25.0000
3	Aniseed Syru	10.0000
4	Chef Anton's	22.0000
5	Chef Anton's	21.3500
6	Grandma's B	25.0000
7	Uncle Bob's	30.0000
8	Northwoods	40.0000
9	Mishi Kobe Ni	97.0000
10	Ruta	31.0000
11	Queso Cabral	21.0000
12	Queso Manch	30.0000
13	Konbu	6.0000
14	Toki	73.0000

Slika 9-12:
Tablica prikazana u DataGridu

III. DIO: DIJELOVI .NET-A



Tablica prikazana u *DataGridu* može imati i svoj naslov umjesto prazne trake prikazane na slici 9-12. Među svojstvima *DataGrida* pronađite svojstvo *CaptionText* i postavite ga na neku vrijednost, primjerice "Proizvodi".

No podatke dohvaćene ovakvim *DataSetom* možete i uređivati. Još jednom označite *DataGrid* i provjerite je li mu svojstvo *ReadOnly* u *Properties* izborniku postavljeno na *False*. Ako nije, postavite ga, jer želimo *DataGrid* iskoristiti za uređivanje podataka u bazi, što uključuje dodavanje novih podataka i brisanje postojećih.

Dodajte u aplikaciju pored *DataGrida* i jedan gumb (ostavite mu *defaultno* ime "button1"). Promijenite mu tekst u "Spremi promjene", jer će on, kao što mu i ime kaže, služiti za spremanje promjena načinjenih u *DataGridu*. Kliknite na njega dva puta i upišite naredni kôd u metodu *button1_Click*:

```
int num = sqlDataAdapter1.Update(dataSet11);
MessageBox.Show("Promijenjeno je " + num.ToString() + " zapisa.", "Operacija
    završena");
```

Pozivamo metodu *Update* koja koristi načinjeni *DataAdapter* za izmjenu svih podataka u *dataSet11* objektu. Prisjetimo se, taj je objekt povezan s *DataGridom* i u njemu se nalaze sve promjene koje je korisnik načinio nad podacima u aplikaciji.

Metoda *Update* vraća cijeli broj koji je jednak broju promijenjenih redaka. Njega spremamo u varijablu *num* i koristimo pri ispisu poruke o broju promijenjenih zapisa.

No pri izmjeni podataka u bazi može ponekad doći i do greške – primjerice, možete pokušati izbrišati podatak čiji se ID referencira u nekoj drugoj tablici, čime biste narušili integritet podataka. To baza neće dopustiti, a vama će se u aplikaciji javiti greška.

Stoga je prikladno napisati kôd koji će provjeravati greške i shodno tome se ponašati. No bit ćemo jednostavni – u slučaju greške ispisat ćemo poruku koju nam vrati baza podataka, a ona će sadržavati razlog neuspjeha metode *Update*.

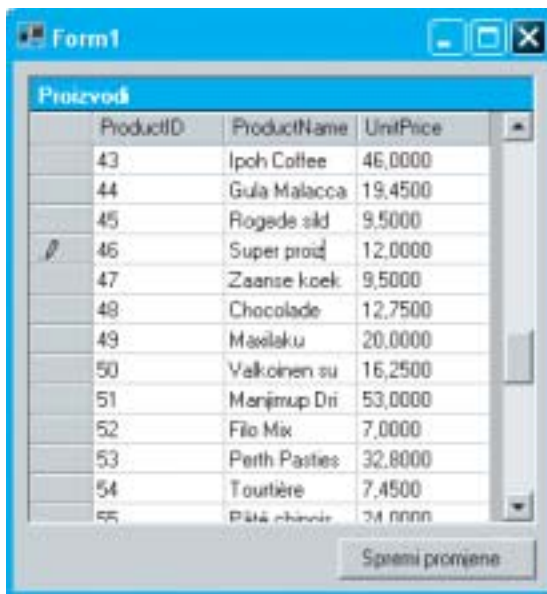
```
int num;
try
{
    num = sqlDataAdapter1.Update(dataSet11);
}
catch (SqlException se)
{
    MessageBox.Show(se.Message, "Greška!");
    num = 0;
}
```

9. POGLAVLJE: ADO.NET

```
MessageBox.Show("Promijenjeno je " + num.ToString() + " zapisa.", "Operacija  
završena");
```

Dakle, u bloku *try* pokušavamo izvršiti metodu *Update*, a u bloku *catch* hvatamo iznimku *SqlException* (svakako provjerite imate li na početku programa "using System.Data.SqlClient"). Ukoliko se ona pojavi, ispisujemo njen tekst, jer se radi o grešci. Na samom kraju ispisujemo broj izmijenjenih podataka.

Kako se uopće mogu mijenjati podaci *DataGridom*? Jednostavno. Pozicionirajte se u bilo koji redak, kliknite mišem i on će se pretvoriti u tekstualno polje čiji ćete sadržaj moći mijenjati. Tako možete mijenjati sadržaj bilo kojeg retka, no ne možete mu promijeniti primarni ključ, u našem slučaju *ProductID*.



Slika 9-13:
Naša gotova aplikacija u procesu mijenjanja sadržaja nekog retka

Retke možete i brisati. Označite cijeli redak i pritisnite DEL gumb na tastaturi. Želite li pak dodati novi redak, pozicionirajte se na dno *DataGrida* i učit ćete prazan redak označen s "*". U njega možete upisati nove podatke. Kad jednom pozovete metodu *Update*, ti će se podaci automatski sinkronizirati s bazom podataka odnosno snimit će se sve promjene.

DataGridu možete mijenjati standardni izgled. Označite ga i na dnu prozora *Properties*, ispod tablice sa svojstvima, pronađite link *Auto Format*. Kliknete li na njega, otvorit će vam se prozor s mogućim temama izgleda *DataGrida*. Pronađite onu koja vam se najviše sviđa, kliknite na OK i vaš će *DataGrid* odmah promijeniti izgled.



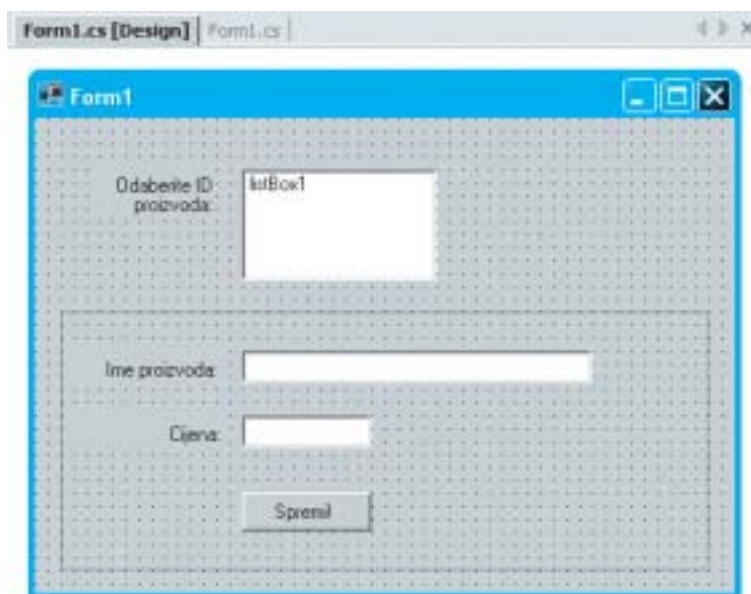
III. DIO: DIJELOVI .NET-A

Uređivanje podataka u bazi pomoću formulara

Naš sljedeći zadatak bit će uređivanje podataka prikazanih iz prošlog primjera pomoću formulara. Nećemo naglasak staviti na samu aplikaciju, već na pristup podacima – cilj nam je pokazati praktičnu uporabu *DataReadera* te objekata *Command* s parametrima i bez njih.

Izradite stoga formular izgleda poput onog na slici 9-14. U gornji dio stavite kontrolu *ListBox* u kojoj će stajati popis zapisa iz baze. Odabirom nekog zapisa iz baze, njegovo ime i cijena će se učitati u dva tekstualna polja spremljena niže u formularu. Klikom na gumb *Spremi*, novi će se podaci spremiti u bazu podataka. Svim kontrolama smo ostavili *defaultna* imena, pa se *ListBox* kontrola i dalje zove *listBox1*, tekstualno polje za ime proizvoda *textBox1*, za cijenu *textBox2*, a gumb za spremanje *button1*.

Slika 9-14:
Kontrole u formularu za uređivanje podataka iz baze



Dohvaćanje popisa zapisa

Krenut ćemo standardnim načinom. Na radnu površinu povucite objekt *SqlConnection* iz dijela *Data* prozora *Toolbox*. Izgradite novi *connection string* ili odaberite neki već postojeći iz padajućeg izbornika u svojstvu *ConnectionString* u prozoru *Properties*. Kao što smo rekli, dohvaćat ćemo i mijenjati podatke kao u prošlom primjeru, dakle iz tablice *Products* iz baze *Northwind*.

9. POGLAVLJE: ADO.NET

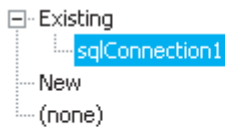
U svojoj aplikaciji možete imati više veza na različite baze podataka, tj. *SqlConnection* objekata. Tako smo podatke u našem formularu mogli dohvaćati iz više različitih baza ili izvora podataka, samo nam trebaju dodatni *Connection* objekti, a postupak i dalje ostaje isti.



No, za razliku od prethodnog primjera u kojem smo koristili *DataAdapter*, ovaj put ćemo iskoristiti objekte *Command* za komunikaciju s bazom. Ukupno će nam trebati čak tri objekta *Command* – jedan će dohvaćati popis svih zapisa u bazi koji će se puniti u izbornik u gornjem dijelu formulara, drugi će dohvaćati podatke nekog zapisa (ime proizvoda i cijenu) u trenutku kad korisnik odabere drugi proizvod u izborniku, a treći će vršiti naredbu UPDATE i mijenjati podatke u bazi.

Stoga napravimo prvi objekt *Command* – povucite iz *Data* dijela prozora *Toolbox* na radnu površinu objekt *SqlCommand*. Dodati će se na dno radne površine među ostale objekte za rad s podacima.

Da bi objekt *Command* ispravno radio, treba mu namjestiti vezu s bazom podataka, tj. odabrati koja se veza koristi. Kako smo mi već napravili vezu s bazom podataka, iskoristit ćemo postojeću. U svojstvu *Connection* u padajućem izborniku odaberite *Existing – sqlConnection1*, kao što je i prikazano na slici 9-15.



Slika 9-15:
Odabir veze s bazom podataka za objekt Command

Objekt *Command* vezan uz SQL Server može pozivati spremljene procedure, a može i izvršavati obične SQL upite. Zbog jednostavnosti, u primjerima ćemo sami izgrađivati SQL upite. Stoga provjerite je li u svojstvu *CommandType* odabran "Text", a zatim u svojstvu *CommandText* upišite novi SQL upit ili ga načinite pomoću već opisanog Query Buildera.

Kako namještamo prvi objekt *Command* koji će služiti za dohvaćanje podataka u *ListBox*, napisat ćemo sljedeći upit:

```
SELECT ProductID FROM Products ORDER BY ProductID
```

Još nam je samo ostalo zbog jednostavnosti promijeniti ime objekta *Command*. Promijenite mu stoga svojstvo (*Name*) i u njega upišite *cmdProizvodi*. Podsjetimo, imat ćemo tri objekta *Command*, pa ih je nužno tako nazvati kako bismo se lakše snalazili.

III. DIO: DIJELOVI .NET-A

Sljedeći korak je pozvati načinjeni objekt *Command* i upisati dohvaćene podatke u kontrolu *List-Box*. To ćemo obaviti odmah po učitavanju formulara – kliknite dvaput bilo gdje na površinu formulara i prebacit ćete se u metodu *Form_Load* koja se izvršava po učitavanju, dakle točno ono što nama treba.

U nju upišite sljedeći kôd:

```
sqlConnection1.Open();
SqlDataReader dr = cmdProducts.ExecuteReader();

while (dr.Read())
{
    listBox1.Items.Add(dr["ProductID"].ToString());
}

dr.Close();
sqlConnection1.Close();

listBox1.SelectedIndex = 0;
```

Krenimo redom. Na početku otvaramo vezu s bazom jer nam je ona potrebna za dohvaćanje podataka, a zatim stvaramo objekt tipa *SqlDataReader* i u njega spremamo rezultate izvršavanje načinjenog objekta *Command*. I tu je ključ – pozivom metode *ExecuteReader()* nad objektom *Command* dohvaćamo podatke u *DataReader* i njih možemo čitati i koristiti u aplikaciji.

Slika 9-16:
Prva verzija aplikacije koja
podacima iz baze popunjava
kontrolu ListBox



U petlji *while* prolazimo kroz sve dohvaćene zapise – naredbom *Read()* nad *DataReaderom* čitamo naredni zapis i sve dok je ona *true*, zapisi postoje. U trenutku kad ona vrati *false*, došli smo do kraja učitanih zapisa i vrijeme je da petlja prestane s izvršavanjem.

U samoj petlji dodajemo nove elemente u kontrolu *ListBox* pozivanjem metode *Add*. Kao elemente dodajemo vrijednosti *ProductID* svakog zapisa u bazi. Njih dohvaćamo već poznatim načinom – korištenjem *DataReader*a i navođenjem imena stupca unutar navodnika u uglatim zagradama. Sve naravno pretvaramo u znakovni niz da bi se moglo prikazati u aplikaciji.

Nakon petlje zatvaramo otvoreni objekt *DataReader* i vezu s bazom podataka. Također, označavamo u kontroli *ListBox* prvi zapis, što će nam trebati kasnije. Indeksi svih zapisa, kao i u drugim kolekcijama, kreću od nule, pa prvi zapis ima indeks 0.

Pokušajte pokrenuti aplikaciju i uvjerit ćete se sami – rezultat je prikazan na slici 9-16.

Dohvaćanje informacija o jednom zapisu

Krenimo s nadogradnjom naše jednostavne aplikacije. U sljedećem koraku ćemo pokazati kako se pozivaju objekti *Command* s parametrima jer želimo dohvatiti informacije o jednom zapisu iz baze, ovisno o tome koji se proizvod odabere u *ListBoxu*.

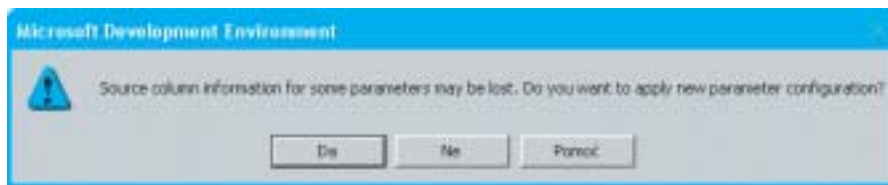
Stoga dodajte još jedan objekt *Command* u aplikaciju, a ime mu promijenite u *cmdGetProduct* (primijetite da svuda za objekte *Command* koristimo prefiks “cmd” kako bismo se lakše snalazili među raznim objektima koje koristimo). Kao i u prethodnom primjeru, postavite mu *Connection* svojstvo na već postojeću vezu na bazu podataka, *sqlConnection1*.

Ključan je upis SQL upita. Otvorite Query Builder iz svojstva *CommandText* i u njemu upišite sljedeći upit:

```
SELECT ProductName, UnitPrice FROM Products WHERE (ProductID = @PID)
```

Kao što vidite, dohvaćamo ime i cijenu za onaj proizvod kojem vrijednost *ProductID* odgovara vrijednosti *@PID*, što predstavlja parametar imena “PID” (nazvali smo ga tako, jer sadržava vrijednost

Slika 9-17:
Automatsko stvaranje kolekcije parametara za objekt Command



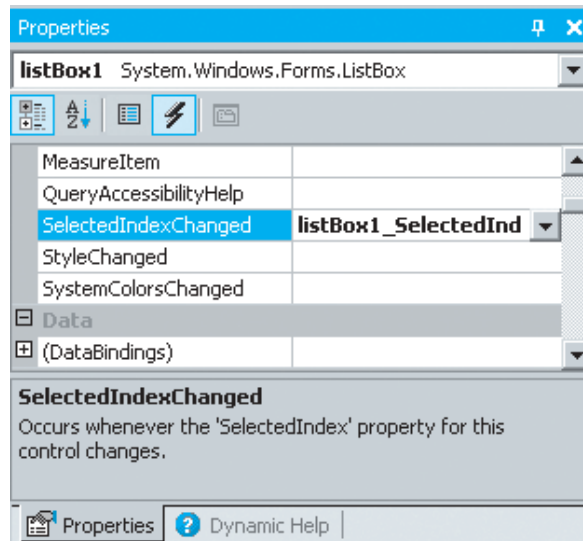
III. DIO: DIJELOVI .NET-A

ProductID-a). Parametri se, dakle, označavaju znakom “@”. U našoj aplikaciji ćemo u različitim situacijama davati drugu vrijednost parametru @PID, što će rezultirati drugačijim SQL upitom i, samim time, drugačijim rezultatima.

Svaki objekt *Command* ima svoju kolekciju parametara. U prethodnom primjeru ih nismo koristili, no sada to činimo. Nakon što upišete SQL upit koji koristi parametre, Visual Studio će za vas automatski generirati kolekciju parametara. Prikazat će vam se prozor na slici 9-17, a vi potvrdno odgovorite jer želite koristiti parametre definirane u SQL upitu.

Sad trebamo na promjenu odabranog proizvoda u *ListBoxu* promijeniti i sadržaj tekstualnih polja. Za to ćemo se pouzdati u događaj u aplikaciji odnosno nad samim *ListBoxom*. Označite ga te se prebacite u prozor *Properties*. U njemu, u dijelu *Events*, kao što je prikazano na slici 9-18, dva puta kliknite u svojstvo *SelectedIndexChanged*. Automatski će se stvoriti nova metoda koja će se pozivati na svaku promjenu odabranog elementa u *ListBoxu*.

Slika 9-18:
Događaj *SelectedIndexChanged*
poziva se pri svakoj promjeni
odabranog elementa u
***ListBoxu*.**



U novoj metodi koja se, dakle, poziva na svaku promjenu odabranog elementa u *ListBoxu* trebat ćemo dohvatiti informacije iz baze za proizvod s odabranim ID-em. Evo i kompletnog kôda:

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
```



```

string pid = listBox1.SelectedItem.ToString();

sqlConnection1.Open();
cmdGetProizvod.Parameters["@PID"].Value = pid;
SqlDataReader dr = cmdGetProizvod.ExecuteReader(CommandBehavior.SingleRow);

if (dr.Read())
{
    textBox1.Text = dr["ProductName"].ToString();
    textBox2.Text = dr["UnitPrice"].ToString();
}

dr.Close();
sqlConnection1.Close();
}

```

Na samom početku u varijablu *pid* učitavamo vrijednost trenutno odabranog elementa u *ListBoxu*. To je ID proizvoda čije ime i cijenu želimo učitati u tekstualna polja.

Zatim otvaramo vezu s bazom podataka. I slijedi ključni dio – objektu *cmdGetProizvod* postavljamo vrijednost parametra *@PID* na vrijednost trenutno odabranog elementa iz *ListBoxa*. To činimo pomoću kolekcije *Parameters*, navodeći ime parametra unutar navodnika u uglatim zagradama (nemojte zaboraviti znak “@” u imenu parametra!).

Sad tu naredbu možemo izvršiti. Kao i u prethodnom primjeru, izvršavamo naredbu *ExecuteReader* koja vraća podatke iz baze. No ovaj put naredbi prosljeđujemo jedan parametar. Radi se o pobrojanom polju imena *CommandBehavior* i njegovoj vrijednosti *SingleRow*. Naime, kako planiramo dohvatiti samo jedan zapis iz baze, navodeći pri izvršavanju naredbe *CommandBehavior.SingleRow* optimiziramo njeno izvršavanje i time poboljšavamo rad aplikacije. U prethodnom primjeru smo očekivali više vraćenih redaka, pa nismo navodili ovaj parametar.

Opet koristimo metodu *Read()* za čitanje – ukoliko ona vrati vrijednost *true*, znači da postoje podaci za čitanje, pa ih možemo upisati u tekstualna polja. U slučaju da je vratila *false*, to bi značilo da naš upit nije vratio niti jedan rezultat, što je očita posljedica neke greške, pa ne bi ni imalo smisla pokušavati pročitati podatke i zapisati ih u tekstualna polja.

Sjetite se kako smo na kraju metode *Form_Load* označili prvi element u *ListBoxu*. Time se automatski poziva maloprije napisana metoda *listBox1_SelectedIndexChanged* pa imamo sljedeći rezultat – po pokretanju aplikacije označava se prvi element, a samim time se i učitavaju vrijednosti tog zapisa iz baze i u odgovarajuća polja u formularu upisuju ime i cijena proizvoda.



III. DIO: DIJELOVI .NET-A

Nakon zapisivanja podataka u formular, zatvaramo *DataReader* i, naravno, vezu s bazom podataka. Iskušajte sada aplikaciju – pokrenite je i mijenjajte odabrane elemente i vidjet ćete kako se imena i cijene proizvoda upisuju u polja formulara.

Slika 9-19:
Poboljšana verzija aplikacije. S promjenom odabira u ListBoxu mijenjaju se vrijednosti u odgovarajućim tekstualnim poljima.

Spremanje promjena

Ostalo nam je još samo spremanje promjena. Za to će nam trebati treći objekt *Command*. Promijenite mu ime u “cmdSaveProizvod” i upišite naredni SQL upit:

```
UPDATE Products SET ProductName = @PN, UnitPrice = @UP WHERE (ProductID = @PID)
```

Dakle, izvršit ćemo naredbu UPDATE koja će spremiti novo ime proizvoda (parametar @PN) i novu cijenu proizvoda (parametar @UP) za određeni zapis. Koji točno zapis treba promijeniti odredit ćemo pogledavši koji je element odabran u *ListBoxu*.

Kliknite dvaput na gumb na kontroli koji je predviđen za spremanje podataka u bazu i prebacit ćete se u metodu *button1_Click* koja će se pozvati na svaki klik na gumb.

```
private void button1_Click(object sender, System.EventArgs e)
{
    cmdSaveProizvod.Parameters["@PN"].Value = textBox1.Text;
```

9. POGLAVLJE: ADO.NET

```

cmdSaveProizvod.Parameters["@UP"].Value = textBox2.Text;
cmdSaveProizvod.Parameters["@PID"].Value = listBox1.SelectedItem.ToString();

sqlConnection1.Open();
int num = cmdSaveProizvod.ExecuteNonQuery();
sqlConnection1.Close();

if (num == 1)
{
    MessageBox.Show("Promjene su uspješno snimljene.", "Uspjeh");
}
}

```

Najprije postavljamo parametre naredbe *cmdSaveProizvod*. Parametar @PN služi za spremanje vrijednosti *ProductName*, a nju čitamo iz prvog tekstualnog polja. S druge strane, parametar @UP služi za spremanje vrijednosti *UnitPrice*, a nju čitamo iz drugog tekstualnog polja. Treći parametar određuje ID proizvoda čije ime i cijenu trebamo promijeniti, a njega čitamo iz *ListBoxa* i on odgovara trenutačno označenom ID-u.

Dalje je sve poznato – otvaramo bazu podataka te nad objektom *Command* izvršavamo metodu *ExecuteNonQuery()* koja služi za izvršavanje naredbi koje ne vraćaju neke podatke (dakle, nećete je upotrijebiti uz naredbu SELECT), već vraća broj zapisa koji su promijenjeni (što može podrazumijevati dodane zapise, ukoliko izvršavate naredbu INSERT, izbrisane ukoliko izvršavate DELETE ili izmijenjene ukoliko pak izvršavate naredbu UPDATE).

Dosad smo objasnili pozivanje *ExecuteReader()* i *ExecuteNonQuery()* metoda nad *Command* objektom, no još jedna vam može biti veoma korisna. Primjerice, želite li dohvatiti samo jednu vrijednost odnosno vrijednost prvog stupca u prvom zapisu dohvaćenih podataka, korisna će vam biti *ExecuteScalar()* naredba. Primjerice, imate li SQL naredbu "SELECT COUNT(*) FROM Products" kojom brojite zapise u tablici *Products*, nju biste izvršili na sljedeći način i odmah u varijabli *broj* imali traženi podatak jer je on dohvaćen kao vrijednost prvog stupca u prvom zapisu podataka. Ne treba ni govoriti koliko je to brži način dohvaćanja jednog podatka iz baze od, primjerice, korištenja *DataReader* objekta za takve stvari.

```
int broj = cmdBrojProizvoda.ExecuteScalar();
```

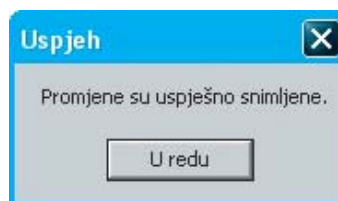


Podatak o broju izmijenjenih zapisa spremamo u varijablu *num*. Ukoliko je njena vrijednost jednaka 1, znači da smo uspješno izmijenili samo jedan zapis, kao što je i planirano. Na kraju ispisujemo poruku o uspjehu.

III. DIO: DIJELOVI .NET-A



Slika 9-20:
Uspješno obavljena operacija
spremanja



Korištenje ADO.NET-a i rad s podacima iz baza podataka ide ruku pod ruku s *error handlingom* i hvatanjem iznimki. Preporučuje se svaku operaciju s bazom podataka staviti u blok *try catch* i, naravno, hvatati samo odgovarajuće iznimke, primjerice tipa *SqlException*. Greške pri radu s ADO.NET-om su moguće jer ne ovise o vašem kodu – primjerice, baza podataka može biti nedostupna, SQL Server može biti zaustavljen i ne odgovara na vaše upite, pokušavate izbrisati zapis koji ne možete izbrisati zbog referencijalnog integriteta baze i slično. U svakom slučaju, iako smo ovdje radi jednostavnosti izbjegli pisanje blokova *try catch*, to preporučujemo u stvarnim situacijama.

Na kraju, sve ovdje opisano vrijedi i za ASP.NET web-aplikacije. Iako smo u primjerima koristili prozorske aplikacije, ADO.NET je ključan dio .NET *frameworka* i možete ga iskoristiti u svim tipovima aplikacija, bilo da izrađujete ASP.NET aplikacije, web-servise ili aplikacije za mobilne uređaje.