KONZORCIJ ŠOLSKIH CENTROV

REPUBLIKA SLOVENIJA
MINISTRSTVO ZA ŠOLSTVO IN ŠPORT

Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

Načrtovanje programskih aplikacij

NPA 3.del

Srečo Uranič

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KONZORCIJ ŠOLSKIH CENTROV
REPUBLIKA SLOVENIJA
MINISTRSTVO ZA ŠOLSTVO IN ŠPORT
Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski socialni sklad

SPLOŠNE INFORMACIJE O GRADIVU

Izobraževalni program

Tehnik računalništva

Ime modula

Načrtovanje programskih aplikacij – NPA 4

Naslov učnih tem ali kompetenc, ki jih obravnava učno gradivo

Vizuelno programiranje, dedovanje, knjižnice, večokenske aplikacije, delo z bazami podatkov, testiranje in dokumentiranje, nameščanje aplikacij.

Avtor: Srečo Uranič
Recenzent: Matija Lokar
Lektor: v lekturi
Datum: Februar 2011
CIP:

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

POVZETEK/PREDGOVOR

Gradivo ***Načrtovanje programskih aplikacij*** je namenjeno dijakom 4. letnika izobraževalnega programa SSI – Tehnik računalništva in dijakom 5 letnika PTI - Tehnik računalništva. Pokriva vsebinski del modula Načrtovanje in razvoj programskih aplikacij, kot je zapisan v katalogu znanja za ta program v poklicnem tehniškem izobraževanju. Kot izbrani programski jezik sem izbral programski jezik C# v razvojnem okolju Microsoft Visual C# 2010 Express. Gradivo ne vsebuje poglavij, ki so zajeta v predhodnem izobraževanju. Dostopna so v mojih mapah http://uranic.tsckr.si/ na šolskem strežniku TŠCKR (to so poglavja osnove za izdelavo konzolnih aplikacij, metode, varovalni bloki, razhroščevanje, tabele, zbirke, strukture, naštevanje, razredi in objekti, rekurzije, metoda main in datoteke). V gradivu bomo večkrat uporabljali kratico *OOP* – Objektno Orireniriranо Programiranje.

V besedilu je veliko primerov programov in zgledov, od najenostavnejših do bolj kompleksnih. Bralce, ki bi o posamezni tematiki radi izvedeli več, vabim, da si ogledajo tudi gradivo, ki je navedeno v literaturi.

Gradivo je nastalo na osnovi številnih zapiskov avtorja, črpal pa sem tudi iz že objavljenih gradiv, pri katerih sem bil "vpleten". V gradivu je koda, napisana v programskem jeziku, nekoliko osenčena, saj sem jo na ta način želel tudi vizualno ločiti od teksta gradiva.

Literatura poleg teoretičnih osnov vsebuje številne primere in vaje. Vse vaje, pa tudi številne dodatne vaje in primeri so v stisnjeni obliki na voljo na strežniku TŠC Kranj http://uranic.tsckr.si/VISUAL%20C%23/, zaradi prevelikega obsega pa jih v literaturo nisem vključil še več.

Programiranja se ne da naučiti s prepisovanjem tujih programov, zato pričakujem, da bodo dijaki poleg skrbnega študija zgledov in rešitev pisali programe tudi sami. Dijakom tudi svetujem, da si zastavijo in rešijo svoje naloge, ter posegajo po številnih, na spletu dosegljivih primerih in zbirkah nalog.

Gradivo ***Načrtovanje programskih aplikacij*** vsebuje teme: izdelava okenskih aplikacij, lastnosti in dogodki okenskih gradnikov, dogodki tipkovnice, miške in ure, kontrola in validacija uporabnikovih vnosov, sporočilna okna, dedovanje, virtualne in prekrivne metode, polimorfizem, knjižnice, pogovorna okna, delo s tiskalnikom, nadrejeni in podrejeni obrazci, dostop in ažuriranje podatkov v bazah podatkov, transakcije, testiranje in dokumentiranje aplikacije, izdelava namestitvenega programa.

Ključne besede: gradniki, lastnosti, metode, dogodki, validacija (preverjanje pravilnosti), sporočilna okna, pogovorna okna, razred, dedovanje, polimorfizem, virtual, override, using,

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

knjižnica, tiskalnik, baza, transakcija, MDI, DataSet, SQL, testiranje, dokumentiranje, namestitev.

Legenda: Zaradi boljše preglednosti je gradivo opremljeno z motivirajočo slikovno podporo, katere pomen je naslednji:

-  informacije o gradivu;
-  povzetek oz. predgovor;
-  kazala;
-  učni cilji;
-  napoved učne situacije;
-  začetek novega poglavja;
-  posebni poudarki;
-  nova vaja, oziroma nov primer;
-  rešitev učne situacije;
-  literatura in viri.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO SLIK:

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO TABEL

UČNI CILJI

Učenje programiranja je privajanje na algoritmični način razmišljanja. Poznavanje osnov programiranja in znanje algoritmičnega razmišljanja je tudi nujna sestavina sodobne funkcionalne pismenosti, saj ga danes potrebujemo praktično na vsakem koraku. Uporabljamo oz. potrebujemo ga:

- pri vsakršnem delu z računalnikom;
- pri branju navodil, postopkov (pogosto so v obliki "kvazi" programov, diagramov poteka);
- za umno naročanje ali izbiranje programske opreme;
- za pisanje makro ukazov v uporabniških orodjih;
- da znamo pravilno predstaviti (opisati, zastaviti, ...) problem, ki ga potem programira nekdo drug;
- ko sledimo postopku za pridobitev denarja z bankomata;
- ko se odločamo za podaljšanje registracije osebnega avtomobila;
- za potrebe administracije – delo z več uporabniki;
- za ustvarjanje dinamičnih spletnih strani;
- za nameščanje, posodabljanje in vzdrževanje aplikacije;
- ob zbiranju, analizi in za dokumentiranje zahtev naročnika, komuniciranje in pomoč naročnikom;
- za popravljanje "tujih" programov

CILJI

- Spoznavanje oz. nadgradnja osnov programiranja s pomočjo programskega jezika C#.
- Poznavanje in uporaba razvojnega okolja Visual Studio za izdelavo programov.
- Načrtovanje in izdelava preprostih in kompleksnejših programov.
- Privajanje na algoritmični način razmišljanja.
- Poznavanje razlike med enostavnimi in kompleksnimi programskimi strukturami.
- Ugotavljanje in odpravljanje napak v procesu izdelave programov.
- Uporaba znanih rešitev na novih primerih.
- Spoznavanje osnov sodobnega objektno orientiranega programiranja.
- Manipuliranje s podatki v bazi podatkov.
- Testiranje programske aplikacije in beleženje rezultatov testiranja.
- Ob nameščanju, posodabljanju in vzdrževanju aplikacije.
- Ko zbiramo, analiziramo in dokumentiramo zahteve naročnika, komuniciramo z njim in

mu pomagamo.

- Izdelava dokumentacije in priprava navodil za uporabo aplikacije.
- Uporaba več modulov v programu in izdelava lastne knjižnice razredov in metod.
- Delo z dinamičnimi podatkovnimi strukturami.

DRŽAVE

Potrebujemo bazo podatkov z imenom *DrzaveSQL*, v njej pa dve tabeli: *Kontinenti* in *Drzave*. Pripravili bomo aplikacijo, v kateri bomo obe tabeli tudi poljubno ažurirali in dodajali nove postavke, ustvariti pa želimo tudi nekaj osnovnih poročil. Naučili se bomo izdelati SQL bazo podatkov kar znotraj razvojnega okolja, spoznali gradnike za delo z bazami podatkov, ter razrede in metode za prikaz podatkovnih tabel na obrazcih. Pojasnili bomo tudi pojem transakcije.

Podatkovna skladišča (baze) in upravljanje s podatki

V naslednjem poglavju bo razložena in prikazana manipulacija s podatkovnimi bazami. Naučili se bomo izdelati podatkovno bazo kar znotraj okolja *Visual C# Express Edition*, kasneje pa bomo uporabljali že izdelane primere podatkovnih baz, ki že vsebujejo podatkovne tabele s testnimi podatki. Za izdelavo podatkovnih baz obstajajo seveda tudi druga, bolj specializirana orodja, npr. *Microsoftov SQL Server Management Studio Express*. Za razumevanje poglavja je potrebno vsaj osnovno poznavanje relacijskih podatkovnih baz.

S prihodom *.NET* orodij, se je *Microsoft* odločil tudi za nadgradnjo svojega modela dostopa do podatkovnih baz (ActiveX Data Objects – ADO) in tako je nastal *ADO.NET*. S pomočjo mehanizmov *ADO.NET*, je lahko povezava s podatkovno bazo uporabljena v različnih aplikacijah, pri čemer obstaja možnost začasne prekinitve povezave in ponovne vzpostavitve kar znotraj aplikacije. Tak način dela mnogokrat predstavlja občuten prihranek časa.

Izdelava baze podatkov

Razvojno *Visual C# Express Edition* omogoča enostavno izdelavo podatkovne baze in ustreznih tabel, ključev in vsega ostalega znotraj baze. Seveda nam morajo biti prej znani ključni pojmi glede baz podatkov: kaj pravzaprav baza je, čemu služi, kaj so to podatkovne tabele znotraj baze, kaj je to polje, tipi polj, pojem zapisa, ključa, …

Pri izdelavi baze kar znotraj razvojnega okolja *Visual C# Express Edition*, imamo na izbiro dve vrsti baze podatkov:

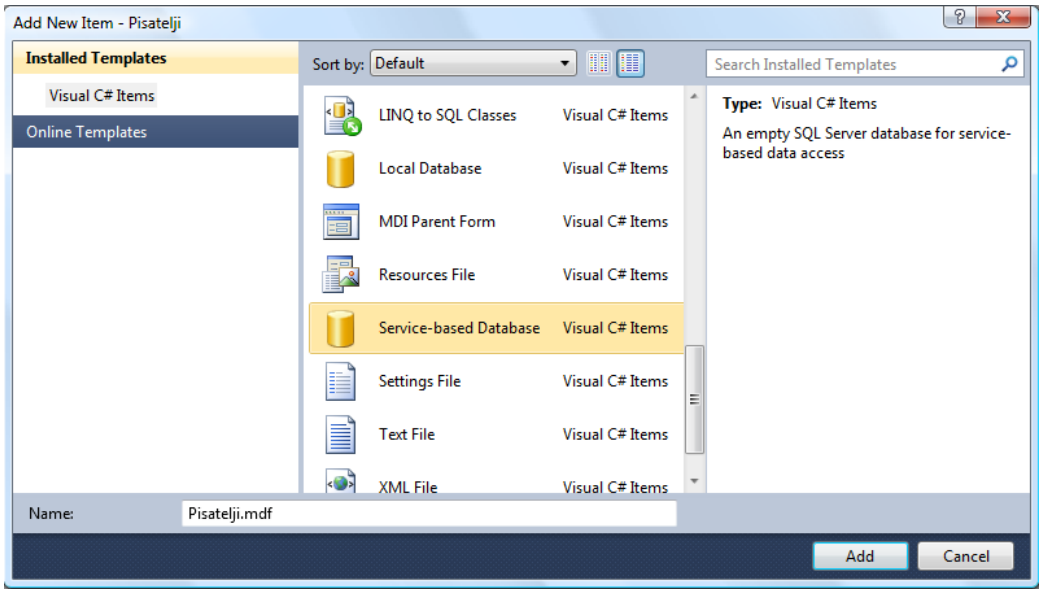
- *Local Database* in
- *Service Based Database*.

Local Database je t.i. *compact edition* baza, ki temelji na datotekah. Za neposreden dostop do podatkov v taki bazi potrebujemo le ustrezen gonilnik (*Driver*). Taka baza ne podpira t.i. *stored* procedur, ki predstavljajo zelo močan mehanizem, ki skrbi za varnost podatkov. Datoteka, v kateri je baza shranjena, ima končnico *SDF* (*SQL Server Compact Edition format*). Za dostop do take baze na lokalnem računalniku ni potrebna instalacija lokalnega strežnika.

Service Based Database pa je baza podatkov, ki je dostopna le s pomočjo *SQL* strežnika. Datoteka, v kateri je baza shranjena, ima končnico *MDF*, ki predstavlja *SQL Server format*. Za priklop na *SQL Server* bazo podatkov je na lokalnem računalniku potreben zagon *SQL Server* servisa, saj le preko tega servisa lahko dostopamo do podatkovnih tabel v bazi.

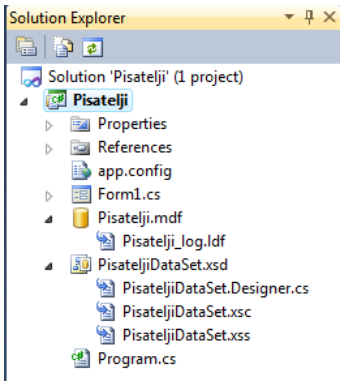
Pri izdelavi aplikacij, ki delajo s podatkovnimi skladišči, se bomo morali s pomočjo *Database Explorerja* priključiti na ustrezno bazo. V primeru, da so naše nastavitve v *SQL Server Configuration Managerju* napačne, bomo pri poskusu uspešnosti povezave na to bazo dobili obvestilo *Failed to generate a user instance of SQL Server due to failure in starting the process for the user instance*. Rešitev je naslednja: odprite *SQL Server Configuration Manager*, nato dvokliknite na *SQLServer (SQLEXPRESS)* in v spustnem seznamu *Buil-in-account* izberite *Local System*. Pobrišite (ali pa bolje le preimenujte) mapo C:\Users\[***user***]\AppData\Local\Microsoft\Microsoft SQL Server Data**SQLEXPRESS**

Za začetno vajo izdelajmo nov projekt in ga poimenujmo *Pisatelji*. Novo bazo, ki jo bomo v tem projektu uporabili, ustvarimo tako, da v meniju *Project izberemo Add New Item…* V oknu izberimo *Service-based Database*, bazo poimenujmo *Pisatelji.mdf* in kliknimo *Add*.



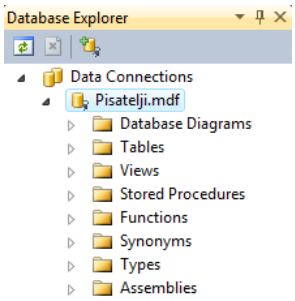
Slika 1: Izdelava nove baze podatkov.

V oknu *Solution Explorer* se pojavi nova postavka *Pisatelji.mdf*, na ekranu pa se čez nekaj časa pojavi okno *Data Source Configuration Wizard*. Razvojno okolje nam ponudi dva podatkovna modela (*POZOR*: izbira dveh modelov je možna šele pri verziji 2010, pri prejšnjih verzijah pa je model *Dataset* privzet!). Izberimo *Dataset* in kliknimo gumb *Next*. Ker želimo ustvariti novo bazo, se čez nekaj časa v oknu pojavi obvestilo, da ustvarjamo novo bazo, oz. da baza, ki jo izdelujemo še ne vsebuje objektov. Razvojno okolje bo zato za nas ustvarilo objekt tipa *DataSet*, to je objekt, ki predstavlja nekakšno kopijo naše baze v pomnilniku. V tem objektu bodo začasno shranjeni vsi podatki iz tabel, ki pripadajo bazi podatkov. Ime objekta (v našem primeru *PisateljiDataSet*) se pojavi v spodnjem levem delu okna, s klikom na gumb *Finish* pa ime potrdimo. Ustvarili smo prazen *DataSet*, ki se pojavi tudi v oknu *Solution Explorer*.



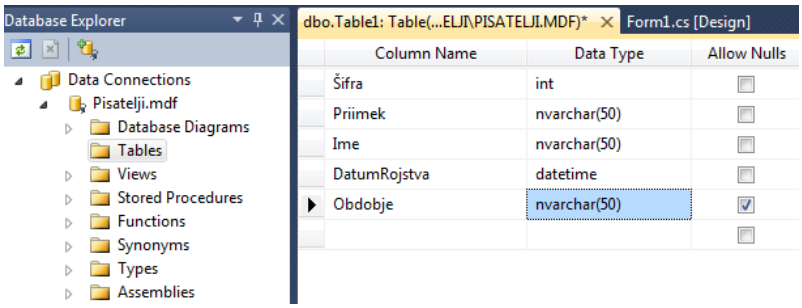
Slika 2: Okno *Solution Explorer* in v njem objekt tipa *DataSet* z imenom *PisateljiDataSet*

V naslednjem koraku bomo v bazi *Pisatelji* ustvarili še tabelo z imenom *Pisatelji*. Dvokliknimo na datoteko *Pisatelji.mdf* v *Solution Explorerju* in (običajno na levi strani) se nam odpre novo okno *Database Explorer*. Namenjeno je ustvarjanju novih tabel in ažuriranju že obstoječih.



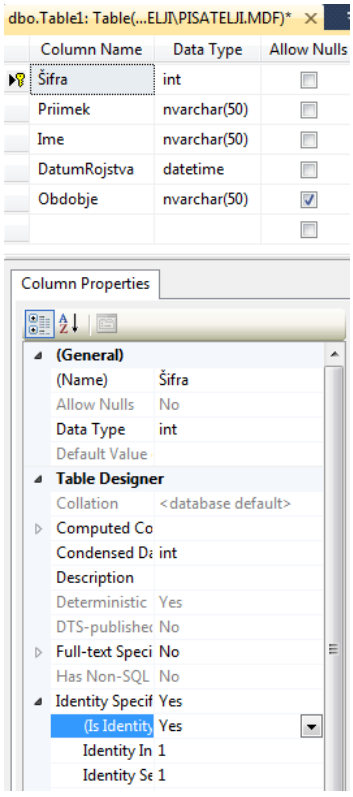
Slika 3: Okno *DataBase Explorer*.

V oknu *Database Explorer* izberimo vrstico *Tables*, kliknimo desni miškin gumb in nato *Add New Table*. Odpre se urejevalnik polj za novo tabelo. Pisatelja bomo opisali s polji, ki so prikazni na naslednji sliki. Dodajmo še, da je razlika med poljema tipa *nvarchar* in *varchar* v tem, da je prvi tip namenjen tudi za shranjevanje neangleških znakov.



Slika 4: Ustvarjanje polj v tabeli *Pisatelji*.

Vsem poljem (razen polju *Obdobje*) smo odstranili kljukico v stolpcu *Allow Nulls*. S tem smo prepovedali prazne vnose posameznih polj v tabelo. Z miško se postavimo še v polje *Šifra* in kliknimo desni gumb. V oknu, ki se prikaže, izberimo *Set Primary Key*. Nastavili smo t.i. primarni ključ, kat v našem primeru pomeni, da se vrednost polja *Šifra* v tabeli ne more nikoli ponoviti. Polju *Šifra* določimo še lastnost *Auto Increment*. Za polja, ki imajo to lastnost je značilno, da bo *SQL Server* njihovo vrednost določal avtomatično. Postopek je naslednji: izberimo polje *Šifra*, nato pa v oknu *Column Properties*, ki se nahaja pod oknom za urejanje polj, nastavimo lastnost *Identity Specification*. Podlastnost *is Identity* nastavimo na *Yes*, podlastnosti *Identity Increment* (vrednost, za katero se bo polje avtomatično povečevalo) in *Identity Seed* (izhodiščna vrednost) pa nastavimo po želji. Privzeti vrednosti sta enaki 1. V našem primeru smo pustili vrednosti privzeti, kar pomeni, da bo šifra prvega pisatelja v tabeli enaka 1, vsem ostalim pa se bo vrednost avtomatično povečevala za 1.



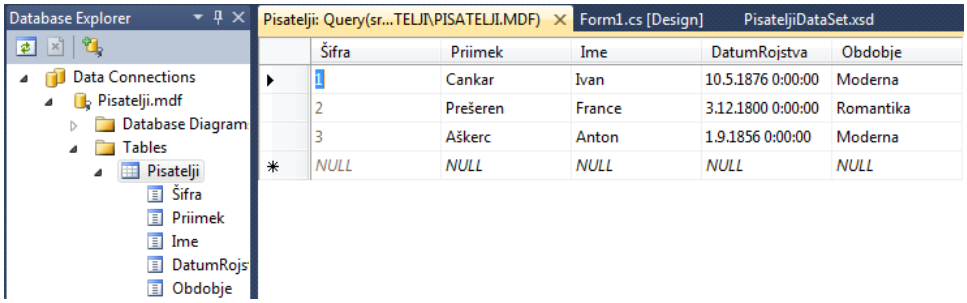
Slika 5: Okno *Column Properties*.

Končno se ponovno postavimo z miško na zavihek nad urejevalnikom polj. Kliknimo desni miškin gumb, izberimo opcijo *SaveTable1* in za ime tabele vnesemo *Pisatelji*. Ustvarjanje prve tabele znotraj naše baze podatkov *Pisatelji* je tako končano.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

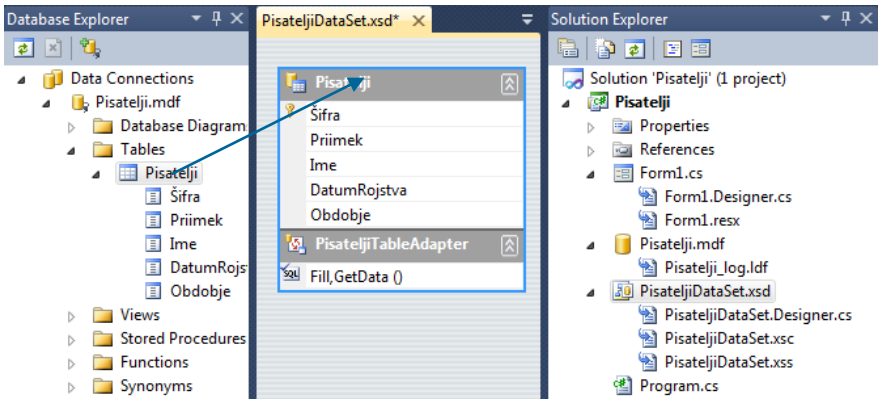
Kasnejše ažuriranje polja neke tabele znotraj razvojnega okolja *Express Edition* NI možno. Če torej hočemo v shranjeni tabeli kakorkoli spremeniti poljubno obstoječe polje (npr povečati število znakov), bomo pri ponovnem shranjevanju dobili obvestilo, da to ni možno. Seveda pa lahko poljubno polje v tabelo kadarkoli dodamo ali pa ga pobrišemo in nato tabelo ponovno shranimo.

Sedaj lahko pričnemo z vpisovanjem testnih podatkov v to tabelo. V *DataBase Explorerju* razširimo vrstico *Tables* pod bazo *Pisatelji.*mdf, napravimo desni klik na tabelo *Pisatelji* in izberimo opcijo *Show Table Data*. V tabelo, ki se prikaže, lahko že kar na tem mestu vnesemo nekaj testnih podatkov. Šifer seveda ne vnašamo, ker smo polju nastavili lastnost *Autoincrement*, se vrednosti temu polju dodeljujejo avtomatično, od 1 naprej.



Slika 6: Vnos podatkov v tabelo *Pisatelji*.

Pri ustvarjanju zgornje baze *Pisatelji* in v njej tabele *Pisatelji*, nam je Visual C# ustvaril tudi prazen *DataSet* in ga poimenoval *PisateljiDataSet*. Ime smo pustili kar privzeto. V ta *DataSet* bomo sedaj dodali našo tabelo *Pisatelji* in jo tako naredili dostopno gradnikom na obrazcu. V oknu *Solution Explorer* dvokliknimo na *PisateljiDataSet*, nato pa v okno, ki se odpre, povlecimo našo tabelo *Pisatelji*.

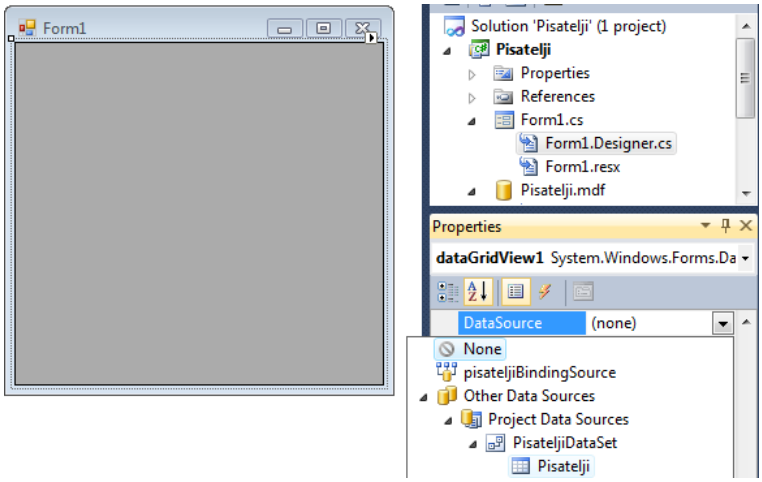


Slika 7: Tabela *Pisatelji* v gradniku *DataSet*.

Tabela je sedaj pripravljena in do nje lahko dostopamo tudi preko lastnosti gradnikov. Ogledamo si lahko tudi predogled vsebine podatkovne tabele *Pisatelji*. V oknu *PisateljiDataSet*

desno kliknemo na tabelo *Pisatelji*. Odpre se meni, v katerem izberemo možnost *Preview Data*. Ko se okno *Preview Data* odpre, v njem le še kliknemo gumb *Preview*.

Vsebino tabele *Pisatelji* bomo sedaj prikazali npr. v gradniku *DataGridView*. Na zaenkrat še prazen obrazec našega projekta, ki smo ga na začetku poimenovali *Pisatelji*, dodajmo gradnik *DataGridView*. Njegovo ime naj bo kar privzeto, to je *dataGridView1*. Lastnost *Dock* mu natavimo na *Fill*, da se razširi čez celoten obrazec. V tem gradniku bi sedaj radi prikazali vsebino tabele *Pisatelji* naše baze *Pisatelji*. Postopek je sledeč: izberimo gradnik *dataGridView1* in v oknu *Properties* lastnost *DataSource*. V spustnem seznamu izberemo *Other Data Sources*. Kliknemo na znak *+*, da se seznam razširi, nato razširimo še *ProjectDataSources*, *PisateljiDataSet* in končno izberemo *Pisatelji*.



Slika 8: Povezava gradnika *DataGridView* s podatkovnim izvorom – tabelo v bazi podatkov.

V gradniku *dataGridView1* so nastali stolpci, ki so dobili enaka imena kot so polja v tabeli *Pisatelji*. Napise na vrhu stolpcev lahko kadarkoli poljubno preimenujemo. To storimo na enak način, kot smo to spoznali v poglavju *Gradnik DataGridView*.

Na obrazcu je vsebina tabele *Pisatelji* zaenkrat še nevidna, a če projekt zaženemo, je gradnik *dataGridView1* napolnjen z ustreznimi podatki tabele *Pisatelji*. Razvojno okolje je ob tem ustvarilo odzivno metodo *Form1_Load* dogodka *Load* našega obrazca in vanj zapisalo stavek.

```
this.pisateljiTableAdapter.Fill(this.pisateljiDataSet.Pisatelji);
```

S pomočjo tega stavka podatke iz tabele *Pisatelji* prenesemo v ustrezen objekt v pomnilniku. Ta objekt smo nato povezali z gradnikom *DataGridView*, ki podatke prikaže na obrazcu.

Ko smo gradili projekt je Visual C# dodal nekaj nevizuelnih gradnikov. Ti so vidni v polju pod obrazcem, njihov pomen in uporabo bomo spoznali v nadaljevanju.

Seveda nas zanima še to, v kateri mapi je razvojno okolje ustvarilo novo bazo, pa seveda tudi to, kako pravkar izdelano bazo prestaviti v kako drugo mapo. Lokacija baze je v fazi načrtovanja

projekta zapisana v datoteki *App.config*, ki se nahaja v mapi z našim projektom. Običajno jo najdemo tudi v oknu *Solution Explorer*. V tej datoteki je tudi stavek *connectionString* z vsemi potrebnimi podatki o lokaciji in dostopu do baze podatkov.

```
connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Pisatelji.mdf;Integrated
Security=True;User Instance=True"
```

Baza se torej na začetku, potem, ko smo jo ustvarili, nahaja v ***|DataDirectory|\Pisatelji.mdf***. V V tej mapi je tudi datotka *app.config* in ostalime glavne datoteke našega projekta. Ko pa projekt z vključeno bazo prvič prevedemo, razvojno okolje celotno bazo prekopira v mapo z izvršilno datoteko našega projekta, to je v mapo ..\bin\Debug. Razvojno okolje v isti mapi ustvari tudi konfiguracijsko datoteko, ki ima enako ime kot izvršilna datoteka in še dodatno končnico *.config*. V zgornjem primeru je ime datoteke *Pisatelji.exe.config*. Bazo sestavljata dve datoteki, *Baza.mdf* in *Baza_log.ldf*, v zgornjem primeru torej *Pisatelji.mdf* in *Pisatelji_log.ldf.* Če želimo bazo prestaviti v drugo mapo, moramo ti dve datoteki najprej premakniti v želeno mapo, nato pa spremeniti *connectinString* v konfiguracijski datoteki, npr. takole:

```
connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=c:\BAZE\Pisatelji.mdf;Integrated
Security=True;User Instance=True"
```

V fazi načrtovanja našega projekta bo ta še vedno uporabljal lokalno bazo v mapi z glavnimi datotekami našega projekta. Pri prevajanju se bo baza še vedno prekopirala v mapo *bin\Debug*, aplikacija pa bo ob zagonu dejansko uporabljala bazo, katere pot je napisana v konfiguracijski datoteki. To seveda pomeni, da moramo pri nameščanju aplikacije, ki dela z bazo podatkov, na ciljni računalnik, namestiti tako izvršilno datoteko, kot tudi konfiguracijsko datoteko, pa seveda tudi ustrezno bazo podatkov. Dobra stran tega dejstva pa je, da je lokacija baze neodvisna od izvršilnega programa, ki ga zaradi tega ni potrebno spreminjati ne glede na to v kateri mapi ali na katerem strežniku je baza.

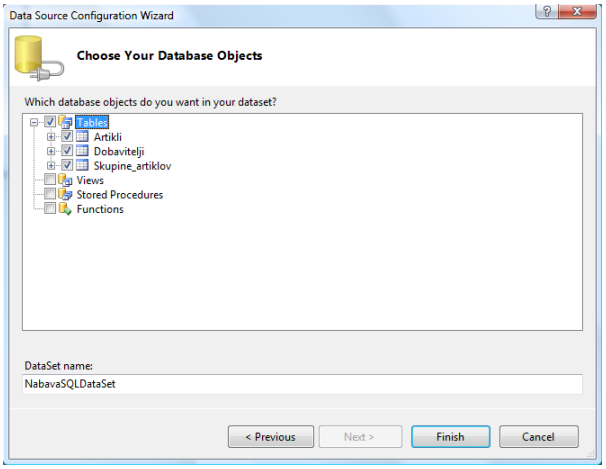
Izdelava novega projekta, ki dela nad obstoječo bazo podatkov – uporaba čarovnika

Pri ustvarjanju novega projekta navadno uporabljamo že obstoječo bazo podatkov, bodisi smo jo naredili že kdaj prej, ali pa jo je za nas naredil že kdo drug. V naslednjem primeru bomo uporabili že obstoječo bazo *NabavaSQL* s tremi tabelami, ki je dostopna na strežniku *http://uranic.tsckr.si/Lokalne%20baze/*. Bazo si najprej prekopirajmo v mapo na svojem računalniku, nato pa ustvarimo nov projekt z imenom *Nabava*. Bazo *NabavaSQL* lahko v naš projekt s pomočjo čarovnika vključimo na dva načina:

- V *Solution Explorerju* označimo naš projekt, kliknemo desni gumb miške, izberemo opcijo *Add* in nato *Add Existing Item*. Odpre se okno za dodajanje obstoječe datoteke v projekt. V spustnem seznamu na dnu tega okna določimo vrsto datoteke, ki jo bomo

dodali v projekt. Izberemo *Data Files* - datoteke tipa *.mdf* in izberemo ustrezno datoteko. Če smo le-to prej skopirali v mapo z našim projektom jo le izberemo, sicer pa se premaknemo v ustrezno mapo na disku, ki vsebuje to bazo podatkov in jo izberemo. S klikom na gumb *Add* jo vključimo v naš projekt. V našem primeru v projekt vključimo bazo *NabavaSQL*. Bazo smo na ta način prekopirali v mapo našega projekta, tako da v fazi načrtovanja oz. izdelave projekta delamo s kopijo prave baze!

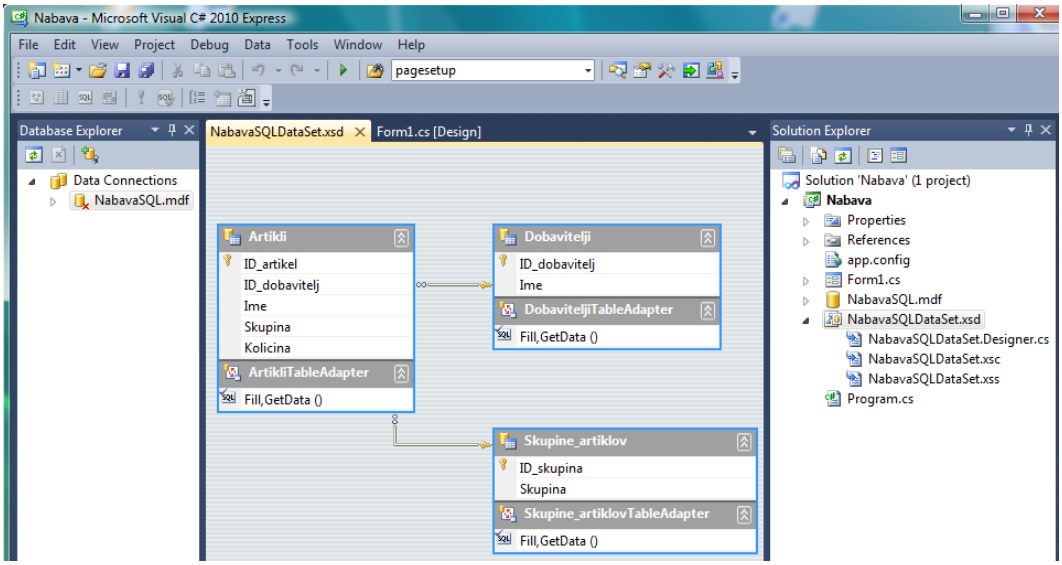
Čez nekaj časa se prikaže okno *Data Source Configuration Wizzard*, kjer izberemo *Dataset* in kliknemo gumb *Next*. Prikaže se seznam tabel, ki pripadajo izbrani bazi podatkov. Odkljukajmo tiste, ali pa vse, s katerimi bomo v projektu delali in kliknimo gumb *Finish*.



Slika 9: V oknu *Data Source Configuration Wizard* izberemo tabele, ki jih bomo potrebovali.

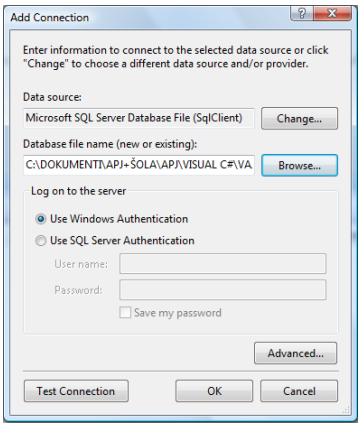
Vsebina stavka connectionString je specifična za posamezno bazo podatkov. Na strani http://www.connectionstrings.com/ so prikazani najpomembnejši povezovalni nizi za posamezne baze.

Razvojno orodje bo ob tem za nas ustvarilo gradnik *Dataset* z imenom *NabavaSQLDataSet*, ki se prikaže v oknu *Solution Explorer*. Če nanj dvokliknemo, dobimo grafični prikaz, obenem pa še povezave med tabelami. Te so odvisne od primarnih ključev v tabelah. V *Solution Explorerju* se pojavi tudi datoteka *app.config*, ki vsebuje podatke o povezavi do baze podatkov (*connectionString*) v fazi načrtovanja projekta. Pri prvem prevajanju projekta bo v mapi *…Bin/Debug* nastala konfiguracijska tekstovna datoteka s končnico *.exe.config*. V njej bo zapisana pot, ki jo bo uporabljal izvršilni program. Pri prenosu na drug računalnik je potrebno le v tej datoteki spremeniti pot do baze na novem računalniku.



Slika 10: *Dataset* z vključenimi tabelami iz baze *NabavaSQL*.

- **Drugi način**: V meniju *Project* izberemo *ADD New Item*, zatem *Dataset*, ki ga poimenujemo npr. *NabavaDataSet* in kliknemo gumb *Add*. V urejevalniškem delu se prikaže okno *Dataset Designer*. V oknu kliknemo na označen tekst *Database Explorer*, da se na levi strani prikaže okno *Database Explorer*. Z miško se postavimo v okno *Database Explorer* in desno kliknimo. V primeru, da z bazo delamo prvič, se nam prikaže okno, v katerem izberemo opcijo *Add Connection...* Odpre se okno *Add Connection*.



Slika 11: Okno *Add Connection*.

Za podatkovni izvor (*DataSource*) izberemo *Microsoft SQL Server Database File*. S klikom na gumb *Browse* poiščemo bazo podatkov, v našem primeru *NabavaSQL.mdb*. Po želji še preverimo, če povezava z bazo deluje (klik na gumb *Test Connection*), s klikom na gumb *OK* pa okno zapremo.

V oknu *DataBase Explorer* kliknemo na *NabavaSql.mdf→Tables* in v okno *NabavaSQLDataSet* povlečemo vse tri tabele iz baze.

Če pa smo z bazo že delali (npr. v prejšnjem projektu), se prikaže okno *TableAdapter Configuration Wizard*. V njem izberemo obstoječo povezavo z bazo, ali pa s klikom na gumb *New Connection* odpremo okno *Add Connection.* Izbiro potrdimo s klikom na gumb *OK*, nato pa le še sledimo navodilom, ki nam jih ponuja čarovnik. Tudi pri tem, drugem načinu, se v oknu *Solution Explorerj* pojavi datoteka *app.config*, ki vsebuje podatke o povezavi do baze podatkov v času načrtovanja projekta (*connectionString*).

Ko smo na enega od načinov bazo vključili v projekt, lahko podatke izbrane tabele baze podatkov prikažemo npr. v gradniku *DataGridView*. Gradnik postavimo na obrazec, kliknemo na ikono v desnem zgornjem kotu tega gradnika in odpremo spustni seznam *Choose Data Source*. Do lastnosti Choose Data Source pridemo tudi tako, da izberemo gradnik *DataGridView1* in v oknu *Properties* odpremo spustni seznam *DataSource*. Razširimo *Other Data Source→Project Data Sources→NabavaSQL* in končno izberemo npr. tabelo *Artikli*! Posebnost podatkov, prikazanih v gradniku *DataGridView* je tudi v tem, da jih lahko urejamo po poljubnem stolpcu, potreben je le klik v celico na vrhu stolpca!

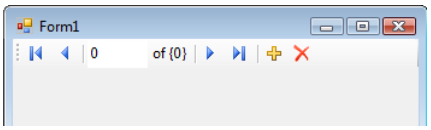
OPOZORILO: v fazi razvoja projekta so v gradniku *DataGridView* prikazana le imena stolpcev tabele iz baze podatkov, vsebina pa se pokaže šele ko projekt zaženemo!

Gradnik BindingNavigator in gradnik TableAdapterManager

Kadar smo obstoječo bazo v ključili v projekt s pomočjo čarovnika, lahko ažuriranje tabel in vstavljanje novih zapisov realiziramo s pomočjo gradnikov *BindingNavigator* in *TableAdapterManager*. Gradnika omogočata enostavno premikanje po podatkovni tabeli, dodajanje in brisanje zapisov.

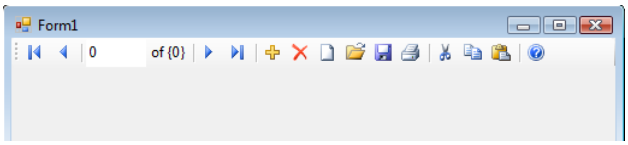
Odprimo projekt *Pisatelji*, ki smo ga ustvarili na začetku poglavja o bazah. Na obrazcu izberimo gradnik *dataGridView1* in ga začasno odstranimo (pobrišimo) iz obrazca. To storimo zato, ker bomo na obrazec najprej postavili gradnik *BindingNavigator* in šele nato zopet *DataGridView*, da bo le-ta zavzemal celotno površino pod gradnikom *BindingNavigator*.

Gradnik *BindingNavigator* je orodjarna z gumbi za premikanje po podatkovni tabeli in standardnimi urejevalniškimi gumbi. Nahaja se na paleti *Data*. Ko ga postavimo na obrazec, se prilepi na vrh obrazca, v njegovi orodjarni pa so le osnovni, standardni gumbi.



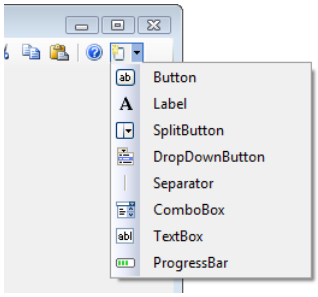
Slika 12: Gradnik *BindingNavigator*.

Dodatne gumbe vključimo takole: izberemo gradnik (*BindingNavigator*), kliknemo na trikotnik v desnem zgornjem kotu in izberemo opcijo *Insert Standard Items*. V orodjarni se prikažejo vsi standardni urejevalniški gumbi.



Slika 13: *BindingNavigator* z vsemi standardnimi urejevalniškimi gumbi.

Poljuben gumb lahko iz orodjarne odstranimo, lahko ga izrežemo, ali pa kopiramo. Najprej ga izberemo, kliknemo desni miškin gumb in iz nato izberemo eno od možnosti *Delete*, *Cut* ali pa *Copy*. Dodajamo lahko tudi nove gumbe, labele in ostale gradnike, ki so na voljo. Potrebno je le odpreti spustni seznam *AddToolStripButton*, ki se nahaja v zgornjem desnem kotu gradnika.

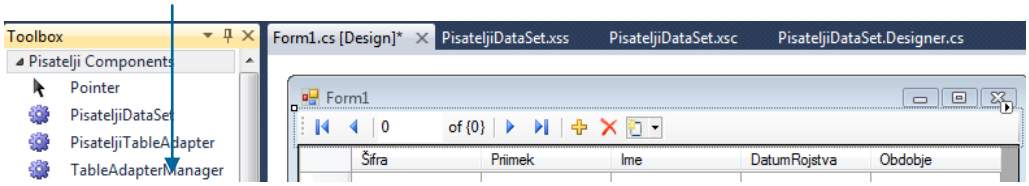


Slika 14: Dodajanje novih gumbov v *BindingNavigator*.

Nadaljujmo sedaj z gradnjo našega projekta. Na obrazec postavimo nazaj gradnik *DataGridView* in mu lastnost *Dock* nastavimo na *Fill*. Povežimo ga s podatkovnim izvorom Pisatelji tako, da za *DataSourc* izberemo *Pisatelji*. Urejevalniške gumbe gradnika *BindingNavigator* aktiviramo tako, da tudi tega povežemo z istim podatkovnim izvorom. Izberemo gradnik *BindingNavigator*, ki je že na obrazcu, v oknu *Properties* izberemo lastnost *BindingSource* in izberemo isti podatkovni izvor kot smo ga prej pri gradniku *DataGridView* (v našem primeru *pisateljiBindingSource*). Gumbi za premik po tabeli, dodajanje in brisanje zapisov postanejo aktivni. Vendar pa moramo biti pri spreminjanju, dodajanju ali brisanju podatkov v gradniku *DataGridView* pozorni. Če hočemo, da bodo spremenjeni podatki ažurirani tudi v bazi podatkov, moramo obvezno ažurirati še bazo. To storimo z uporabo gradnika *TableAdapterManager*. Ta se pojavi v oknu *Toolbox* šele potem, ko smo z uporabo čarovnika ustvarili povezavo z neko tabelo v bazi podatkov. Na obrazec ga postavimo tako kot ostale gradnike, le da je ta gradnik neviden in z njim le dobimo dostop do pomembnih razredov in metod za delo s podatkovnimi tabelami.

Kadar imamo na obrazcu več gradnikov, ki so povezani z nekim podatkovnim izvorom (tako kot v zgornjem primeru gradnik *BindingNavigator* in *DataGridView*), morata imeti oba gradnika isti podatkovni izvor (*BindingSoure*). Le tako bo med njima obstajala povezava.

Gradniku *TableAdpterManager*, ki smo ga postavili na obrazec, pustimo ime kar privzeto.



Slika 15: Gradnik *TableAdapterManager*.

Najpomembnejša metoda gradnika je metoda *UpdateAll*, s katero ažuriramo tabelo v bazi podatkov. V *BindingNavigator* najprej vstavimo dodatni gumb. Gumbu napišemo odzivno metodo dogodka *Click*, v njej pa s pomočjo metode *Update* podatke ažuriramo tudi v bazi podatkov. Pred tem z metodo *Validate* preverimo veljavnost zadnjega vnosa, z metodo *EndEdit* pa spremembe zapišemo v pripadajoči podatkovni izvor.

```
//odzivna metoda dogodka Click gumba Shrani v gradniku BindingNavigator
private void saveToolStripButton_Click(object sender, EventArgs e)
{
    /*Metoda Validate preveri veljavnost polja, ki je trenutno aktivno (ima
      focus) in v primeru napake izvrže izjemo!*/
    Validate();
    pisateljiBindingSource.EndEdit();/*Zaključek urejanja, zadnje spremembe
                                       se zapišejo v ustrezen DataSource*/
     /*Za shranjevanje lahko uporabimo razred TableAdapter in metodo Update
       Ta je v tem primeru bolj smiselna,saj se shranijo podatki le v točno
       določeno tabelo*/
     pisateljiTableAdapter.Update(pisateljiDataSet.Pisatelji);
}
```

Kadar naredimo spremembe v večih tabelah, pa le-te še niso shranjene v bazi, lahko s pomočjo razreda *TableAdapterManager* in njegove metode *UpdateAll* shranimo vse spremembe naenkrat.

```
/*Za shranjevanje lahko uporabimo metodo razreda TableAdapterManager in
  njegovo metodo UpdateAll. Shranijo se vsi še ne shranjeni podatki v
  vseh odprtih tabelah*/
tableAdapterManager1.UpdateAll(this.pisateljiDataSet);
```

V orodjarno *BindingNavigator* dodajmo še gumb za preklic sprememb in dodatni gumb za brisanje vrstice. Pred preklicem sprememb in pred dokončnim brisanjem želimo tudi uporabnikovo potrditev. Na desni strani orodjarne *BindingNavigator* odpremo spustni seznam in dodamo dva gumba. Po želji za oba gumba izberemo ustrezni sliki, nato pa dvoklinimo na enega in drugega in zapišemo ustrezni odzivni metodi.

```
//odzivna metoda gumba za brisanje vrstice
private void toolStripButton2_Click(object sender, EventArgs e)
```

```
{
    //za brisanje postavke zahtevamo uporabnikovo potrditev
    if (MessageBox.Show("Brišem postavko", "BRISANJE",
MessageBoxButtons.OKCancel, MessageBoxIcon.Question) == DialogResult.OK)
    {
        int stvrstice = dataGridView1.CurrentCell.RowIndex;
        //Odstranimo vrstico iz gradnika DataGridView
        dataGridView1.Rows.RemoveAt(stvrstice);
        //Sledi še dokončno ažuriranje v bazi podatkov
        pisateljiTableAdapter.Update(pisateljiDataSet.Pisatelji);
    }
}
```

Za shranjevanje novih oz. ažuriranje spremenjenih podatkov lahko uporabljamo razreda *TableAdapter* in *TableAdapterManager*. Razlika med njima je ta, da se prvi nanaša na le na določeno tabelo v bazi podatkov, drugi pa na vse tabele hkrati.

```
//metoda za preklic (še ne ažuriranih oz. neshranjenih ) sprememb v bazi
private void preklicStripButton1_Click(object sender, EventArgs e)
{
    pisateljiDataSet.Pisatelji.RejectChanges();
    /*takole pa bi preklicali spremembe v vseh tabelah baze, seveda le v
      primeru, če je v bazi več tabel:
      pisateljiDataSet.RejectChanges();*/
}
```

DataGridView, prikaz podatkov in testiranje pravilnosti vnosa podatkov

Gradnik *DataGridView*, ki smo ga povezali s tabelo v bazi podatkov, lahko v fazi načrtovanja projekta tudi poljubno oblikujemo.

Ustvarimo nov projekt *PisateljiDataGridView*. V projekt vključimo nov *DataSet* (desni klik na projekt v oknu *SolutionExplorer→Add→New Item→*izberemo *DataSet*) in ga poimenujemo *DSPisatelji*. Na levi strani razvojnega okolja se prikaže okno *DataBase Explorer*, v katerem desno kliknemo in izberemo *Add Connection*. Odpre se okno *Add Connection*, v katerem izberemo *Microsoft SQL Server Database File*. S klikom na gumb *Browse* poiščemo bazo podatkov *Pisatelji.mdb* in s klikom na gumb *OK* okno zapremo. V oknu *DataBase Explorer* kliknemo na *Pisatelji.mdf→Tables* in v okno *DSPisatelji* povlečemo tabelo *Pisatelji*. Na obrazec *Form1* sedaj postavimo gradnik *DataGridView*, mu lastnost *Dock* nastavimo na *Fill* in ga preko lastnosti *DataSource* povežemo s tabelo *Pisatelj*. V gradniku *DataGridView* se pojavijo stolpci, ki jih bomo sedaj oblikovali. Kliknemo na gumb s trikotnikom na zgornjem desnem robu gradnika *DataGridView* in izberemo *Edit Columns*, da se odpre okno *Edit Columns*. V stolpcu *Obdobje* želimo pripraviti spustni seznam, tako da bo uporabnik obdobje lahko le izbral, ne pa pisal nekaj svojega. V ta namen v oknu *Edit Columns* izberemo vrstico obdobje, poiščemo lastnost *ColumnType* in jo spremenimo v *DataGridViewComboBoxColumn*. Nato poiščemo lastnost *Items* in s klikom na tripičje odpremo urejevalnik *String Colllection Editor*. Vnesemo vrstici *Moderna* in

Romantika, ali pa še več vrstic različnih imen obdobij in urejevalnik zapremo. Po želji lahko spremenimo tudi ostale lastnosti gradnika *DataGridView*. Projekt prevedemo in v stolpcu *Obdobje* bomo le-tega lahko le izbirali, ne pa vnašali neposredno.

Če kliknemo v katerokoli celico gradnika *DataGridView* lahko preko tipkovnice v polje vnesemo novo vsebino. V primeru, da v celico, ki sicer vsebuje numerične podatke, vnesemo alfanumerične znake, dobimo pri poskusu premika na drugo celico obvestilo o napaki. Okno z obvestilom v tem primeru zapremo in popravimo vsebino celice (ali pa pritisnimo tipko *Escape* za preklic vnosa). Obvestilo o napaki je privzeto, lahko pa ga nadomestimo s poljubnim lastnim obvestilom o napaki. To storimo tako, da napišemo svoj *DataError* dogodek, ki je sestavni del dogodkov gradnika *DataGridView*, npr.:

```
//odzivna metoda dogodka DataError gradnika tipa DataGridView
private void dataGridView1_DataError(object sender,
DataGridViewDataErrorEventArgs e)
{
    //testiranje napačnega vnosa datuma
    if (dataGridView1.Columns[e.ColumnIndex].DataPropertyName ==
           "DatumRojstva")
        MessageBox.Show("Napaka pri vnosu datuma!");
    //obvestilo o kakršnikoli napaki
    else MessageBox.Show("Napaka pri vnosu datuma!");
}
```

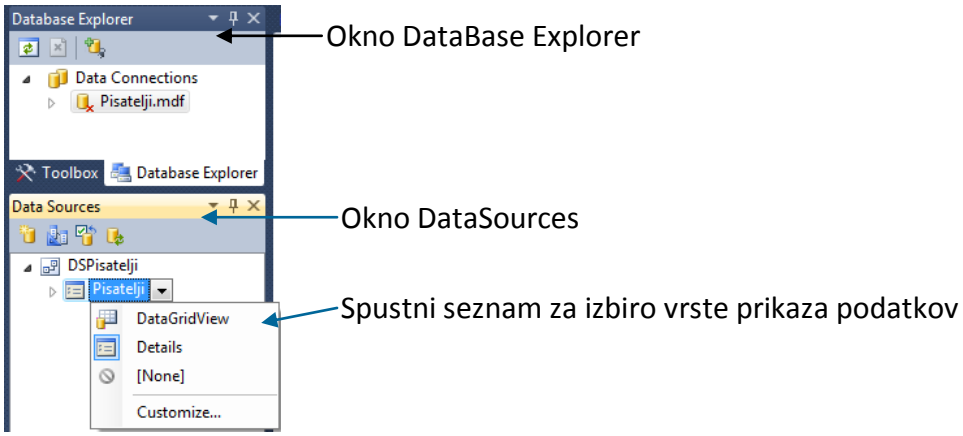
Za shranjevanje in dodajanje novih zapisov moramo seveda projekt opremiti še z gradnikom *BindingNavigator* ali pa shranjevanje spremenjenih podatkov rešiti programsko.

Prikaz podatkov iz baze v načinu *Details* – podrobnostni pregled.

Poljubna tabela baze podatkov je na obrazcu lahko prikazana v dveh oblikah

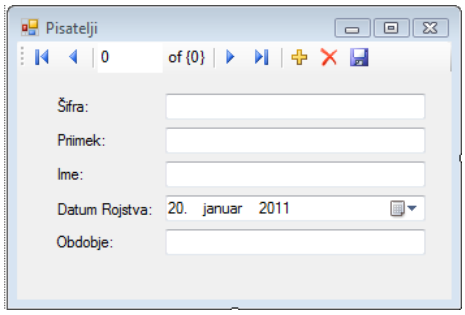
- v obliki *DataGridView*: tabelarični pogled. To je pogled, ki smo ga uporabili v vseh dosedanjih primerih;
- v obliki *Details*: podrobni pogled. To je pogled, pri katerem so na obrazcu le podatki enega samega zapisa iz tabele.

Poglejmo, kako bi namesto tabelaričnega pogleda na tabelo *Pisatelji* naredili podrobnostni pregled. Na obrazec v tem primeru ne postavimo gradnika *DataGridView*, ampak v glavnem meniju izberemo postavko *Data* in nato možnost *ShowDataSources*. Na levem robu se pokaže okno *DataSources*. Razširimo vrstico *DSPisatelji* in kliknemo na tabelo *Pisatelji*, da se prikaže spustni seznam. V seznamu izberemo možnost *Details*.



Slika 16: Okni *DataBase Explorer* in *Data Sources*, ter izbira vrste prikaza podatkov.

V oknu *DataSources* nato izberemo tabelo *Pisatelji* in jo z miško potegnemo na obrazec, da dobimo naslednjo sliko.



Slika 17: Prikaz vsebine tabele iz baze podatkov v podrobnem načinu (*Details*).

Če na obrazcu še ni bilo gradnika *BindingNavigator*, se ta pojavi sedaj. Z njegovo pomočjo se bomo premikali po tabeli. Pod navigatorjem se pojavijo gradniki za prikaz podatkov. Imena polj iz tabele *Pisatelji* so prikazana na oznakah (labelah), poleg njih pa so gradniki, ki ustrezajo tipom podatkov. Večinoma so to gradniki tipa *TextBox*, datum pa je v našem primeru prikazan v gradniku *DateTimePicker*. Ko projekt prevedemo se s pomočjo navigatorja premikamo po tabeli, posamezni podatki tekoče vrstice tabele pa so prikazani na obrazcu. Dodajamo lahko tudi nove zapise, jih brišemo in ažuriramo.

Med ustvarjanjem prejšnjega projekta je razvojno okolje v kodo obrazca dodalo dve metodi: metodo za shranjevanje podatkov v tabelo in metodo za prenos podatkov iz tabele *Pisatelji* v objekt tipa *DataSet*.

```
//Metoda za shranjevanje novih/ažuriranih podatkov v pogledu Details
```

```
private void pisateljiBindingNavigatorSaveItem_Click(object sender, EventArgs
e)
{
    this.Validate();
    this.pisateljiBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.dSPisatelji);
}
/*Metoda za prenos podatkov iz baze v objekt dSPisatelji.Pisatelji ki je tipa
  DSPisatelji*/
private void Form1_Load(object sender, EventArgs e)
{
    this.pisateljiTableAdapter.Fill(this.dSPisatelji.Pisatelji);
}
```

Programska uporaba ADO.NET (brez čarovnika)

Pri izdelavi bolj kompleksnih aplikacij nam uporaba čarovnika v večini primerov ne bo zadoščala. Za dostop do baze podatkov in ustvarjanje ustrezne poizvedbe potrebujemo imenski prostor *System.Data.SqlClient*, ki ga dodamo v naš projekt.

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

Imenski prostor *System.Data.SqlClient* vsebuje specializirane *ADO.NET* razrede, ki se uporabljajo za dostop do *SQL* strežnika. Med njimi je najpomembnejši razred *SQLConnection*, ki je namenjen povezavam z bazami podatkov tipa *SQL Server*.

```
SqlConnection povezava = new SqlConnection();
```

Objekt tipa *SqlConnection* lahko uporabimo na več načinov. Najpogostejši so trije:

- ustvariti želimo samo neko *poizvedbo* (npr. nek *SELECT* stavek) in podatke prikazati v enem od vizuelnih gradnikov na obrazcu;
- izvesti želimo neko obdelavo tabele v bazi in rezultate prikazati npr. v sporočilnem oknu;
- izvesti želimo pravo *SQL transakcijo* v bazi podatkov (npr. *INSERT* ali pa *UPDATE* stavek).

Tako pri poizvedbi, kot tudi pri pravi SQL transakciji je nujna uporaba varovanega bloka, saj pri povezavah lahko pride do številnih problemov in obvestilo o napaki ter vrsti napake je še kako potrebno in dobrodošlo.

Izdelava poizvedbe in prikaz podatkov v gradniku DataGridView

Vsebino tabele *Pisatelji* iz baze podatkov *Pisatelji* bi radi prikazali v gradniku *DataGridView*, ki je na obrazcu. Potrebujemo objekte tipa *SqlConnection*, *SqlDataAdapter* in *DataSet*. Njihov pomen je naslednji:

- Z objektom *SqlConnection* določimo vse potrebne podatke o izvoru podatkov. Konstruktorju posredujemo podatke o izvoru. Uporabimo *ConnectionString*, ki smo ga spoznali v datoteki *App.config*.

- Objekt *SqlDataAdapter* zagotavlja komunikacijo med našo aplikacijo in tabelo v bazi podatkov. Njegova naloga je povezava z bazo, izvršitev ustrezne poizvedbe, shranjevanje pridobljenih podatkov v objekt tipa *DataSet*, ter vračanje ažuriranih podatkov nazaj v bazo podatkov.

- Objekt *DataSet* se uporablja za shranjevanje podatkov, ki smo jih s pomočjo objekta *SQLTableAdapter* pridobili iz neke tabele v bazi podatkov. Podatki so shranjeni v delovnem pomnilniku.

Razlika med razredoma *TableAdapter* in *SQLDataAdapter* je v tem, da je *TableAdapter* vizuelna komponenta, ki jo razvojno okolje uporablja za prikaz vseh podrobnosti povezave z bazo podatkov, SQLDataAdapter pa je razred, preko katerega vzpostavimo povezavo z bazo podatkov s kodiranjem.

Naslednji primer prikazuje, kako bi podatke iz tabele *Pisatelji* prikazali v tabelarični obliki programsko. Objektu tipa *SqlConnection* posredujemo podatke o izvoru, to je niz *ConnectionString*. Komunikacijo z bazo vzpostavimo z objektom *daPisatelji*, ki ga izpeljemo iz razreda *SqlDataAdapter*. Konstruktorju pošljemo želeno poizvedbo in objekt za povezavo z bazo. Podatke iz baze nato pridobimo tako, da ustvarimo objekt tipa *DataSet*, z metodo *Fill* objekta *daPisatelji* pa le-tega napolnimo s podatki iz tabele *Pisatelji*. Gradnik *DataGridView* sedaj s pomočjo lastnosti *DataSource* le še povežemo z objektom *dsPisatelji* in izbrano tabelo. Vsaka tabela iz baze ima v objektu tipa *DataSource* svoj indeks. Tako ima tabela *Pisatelji* v objektu *dsPisatelji* indeks 0 in do nje pridemo preko *dsPisatelji.Tables[0]*.

```
SqlConnection povezava;//Objekt za povezavo z bazo
//Objekt tipa SqlDataAdapter za komunikacijo z tabelo v bazi podatkov
SqlDataAdapter daPisatelji;
//Objekt tipa DataSet za shranjevanje pridobljenih podatkov iz baze
DataSet dsPisatelji;
//Pri delu s poizvedbami navadno uporabimo varovalni blok
try
{
    //Povezovalni niz ConnectionString z bazo podatkov
    string izvor = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\Pisatelji\Pisatelji.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True";
    string SQL = "SELECT * FROM Pisatelji";//Vsebina poizvedbe
    //Konstruktor objekta tipa SqlConnection dobi podatke o izvoru baze
    povezava = new SqlConnection(izvor);
    //Objekt tipa SqlAdapter vzpostavi komunikacijo z bazo
    daPisatelji = new SqlDataAdapter(SQL, povezava);
    //Objekt za shranjevanje pridobljenih podatkov v pomnilniku
```

```
    dsPisatelji = new DataSet();
    //Podatke iz baze z metodo Fill dejansko prenesemo v ustrezen DataSet
    daPisatelji.Fill(dsPisatelji, "Pisatelji");
    //Gradnik DataGridView povežemo s podatki gradnika DataSet v pomnilniku
    //do tabel v bazi dostopamo preko indeksa (Tables[0] je prva tabela)
    dataGridView1.DataSource = dsPisatelji.Tables[0];
}
catch
{
    MessageBox.Show("Napaka pri dostopu do baze podatkov!");
}
finally
{
    povezava.Close();//Na koncu prekinemo povezavo z bazo
}
```

Pogosto želimo v gradniku *DataGridView* prikazati le določene podatke iz neke tabele. Naslednji primer prikazuje, kako lahko iz tabele *Pisatelji* prikažemo le določene pisatelje.

Obdelava tabele v bazi podatkov

Ugotoviti želimo število pisateljev iz tabele *Pisatelji*, ki so rojeni po letu 1900. V ta namen bomo tabelo *Pisatelji* le obdelali. Prebrali bomo vse vrstice iz tabele, eno za drugo. Uporabili bomo objekt razreda *SqlDataReader*, ki ima podobno vlogo kot objekta razreda *StreamReader* pri branju vrstic iz tekstovne datoteke. Razred *SqlDataReader* je bil narejen z namenom, da podatke iz tabel pridobiva enega za drugim in ne pušča zaklenjenih vrstic v tabeli, ko je vsebina vrstice enkrat pridobljena. Tak način pridobivanja podatkov (vrstic) iz tabel seveda pomeni veliko prednost pri izboljšanju konkurenčnosti naše aplikacije. Obenem je uporaba objektov izpeljanih iz razreda *SqlDataReader* tudi najhitrejši način pridobivanja podatkov iz baze.

Za branje celotnega zapisa iz določene tabele bomo zato ustvarili objekt *dataReader* tipa *SqlDataReader* in uporabili njegovo metodo *Read()*. Metoda vrne *False*, če podatki ne obstajajo, ali pa smo na koncu tabele. Do posameznih polj prebranega zapisa pridemo z metodo *GetValue* objekta *dataReader* preko indeksov.

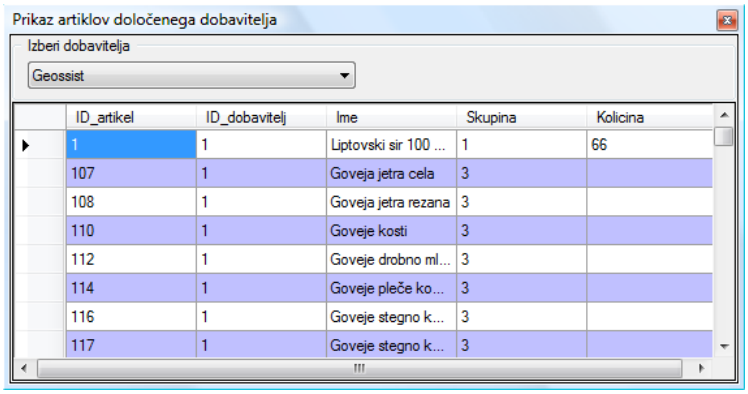
```
//Povezovalni niz z bazo podatkov
string izvor = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\Pisatelji\Pisatelji.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True";

SqlConnection povezava;//Objekt za povezavo
povezava = new SqlConnection(izvor);

//Povezavo z bazo lahko ustvarimo na več načinov. Prikazana sta dva od njih!
//1. NAČIN
//SqlCommand poizvedba = new SqlCommand();
```

```
//poizvedba.CommandText = "SELECT * FROM Pisatelji";
//poizvedba.Connection = povezava;
//2.NAČIN
//Vrsto interakcije z bazo napovemo z objektom tipa SqlCommand
SqlCommand poizvedba = new SqlCommand("SELECT * FROM Pisatelji", povezava);
//Odpremo povezavo z bazo
poizvedba.Connection.Open();

/*Ustvarimo podatkovni tok za branje vrstic iz tabele v bazi. Najhitrejši
  način za pridobivanje podatkov iz SQL Server baze podatkov je z uporabo
  razreda SqlDataReader. Ta razred pridobi vrstice iz baze podatkov
  najhitreje kot omrežje sploh omogoča in podatke preda naši aplikaciji*/
SqlDataReader dataReader = poizvedba.ExecuteReader();
int skupaj = 0; //začetno število pisateljev, rojenih po letu 1900

//metoda Read vrne true, če je poizvedba uspešna, sicer vrne false.
while (dataReader.Read())//dokler obstajajo podatki
{
    /*ker smo v SELECT stavku navedli vse podatke iz tabele Pisatelji ima
      podatek o datumu rojstva znotraj enega stavka indeks enak 3!*/
    if (Convert.ToDateTime(dataReader.GetValue(3)).Year >1900)
            skupaj++;
}
dataReader.Close();//zapremo podatkovni tok
povezava.Close();//Zapremo povezavo z bazo

//Rezultat obdelave prikažemo v sporočilnem oknu
MessageBox.Show("Pisateljev rojenih po letu 1900: "+skupaj);
```

Uporabili smo tudi razred *SqlCommand*. Z objektom tega tipa povemo, kakšno vrsto interakcije z bazo želimo. Osnovne možnosti so *select*, *insert*, modify in delete. V zgornjem primeru smo uporabili le poizvedbo s stavkom *select*. Dejansko pa smo podatke iz tabele nato pridobili s pomočjo objekta tipa *SqlDataReader*.

Povezava med dvema tabelama: prikaz le določenih podatkov iz tabele

Pogosto želimo na obrazcu prikazati le določene podatke (zapise) iz tabele. Naslednja primera prikazujeta, kako lahko to naredimo povsem programsko in kako s pomočjo gradnika *DataSet*. Spustni seznam bomo povezali s tabelo *Dobavitelji* v bazi *NabavaSQL*, vse artikle izbranega dobavitelja iz tabele *Artikli*, pa bomo prikazali v gradniku *DataGridView* na istem obrazcu. V obeh primerih najprej na obrazec postavimo trie gradnike: gradnik *GroupBox* z napisom *Izberi Dobavitelja*, vanj postavimo gradnik *ComboBox*, preostali del obrazca pa zavzema gradnik *DataGridView*.



Slika 18: Povezava gradnika *ComboBox* s tabelo *Dobavitelji* in *DataGridView* s tabelo *Artikli*.

Programska povezava med dvema tabelama

Oba gradnika povežimo s podatkovnim izvorom najprej programsko. Povezavo zapišemo v odzivno metodo dogodka *Load* obrazca. Ustvarimo še objekta *daDobavitelji* tipa *SqlDataAdapter* in *dsNabavaSQL* tipa *DataSet*. Podatke iz tabele *Dobavitelji* lahko sedaj v objekt *dsDobavitelji* prenesemo z metodo *Fill* objekta *daDobavitelji*. Objektu tipa *ComboBox* moramo nato določiti izvor podatkov *DataSource* in še lastnosti *DisplayMember* in *ValueMember*. Pri določanju izvora bomo uporabili stavek

```
comboBox1.DataSource = dsNabavaSQL.Tables[0];
```

V objektu *dsNabavaSQL* je namreč v splošnem lahko več tabel iz baze *NabavaSQL*, mi pa želimo povezavo z eno samo. Vsaka tabela ima svoj indeks in ker smo uvozili eno samo ima ta prav gotovo začetni indeks 0.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    //povezovalni niz za povezavo z bazo NabavaSQL
    string izvor = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\NabavaSQL\NabavaSQL.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True";
    SqlConnection povezava;//Objekt za povezavo
    DataSet dsNabavaSQL; //DataSet za shranjevanje prenešenih tabel
    SqlDataAdapter daDobavitelji;//adapter za prenos dobavitev
    private void Form1_Load(object sender, EventArgs e)
    {
        try
        {
```

```
            povezava = new SqlConnection(izvor);
            SqlCommand poizvedba = new SqlCommand();
            poizvedba.CommandText = "SELECT * FROM Dobavitelji";
            poizvedba.Connection = povezava;
            daDobavitelji = new SqlDataAdapter(poizvedba.CommandText,
povezava);
            dsNabavaSQL = new DataSet();
            //objekt dsNabavaSQL povežemo s tabelo Dobavitelji
            daDobavitelji.Fill(dsNabavaSQL,"Dobavitelji");
            //objekt comboBox1 povežemo s podatkovnim izvorom
            comboBox1.DataSource = dsNabavaSQL.Tables[0];
            //prikazana bodo imena dobaviteljev
            comboBox1.DisplayMember = "Ime";
            /*ob izbiri dobavitelja bo "vrednost" izbire ID številka tega
              Dobavitelja*/
            comboBox1.ValueMember = "ID_Dobavitelj";
            //Odpremo povezavo z bazo, da se izvede poizvedba
            poizvedba.Connection.Open();
            povezava.Close();//Zapremo povezavo z bazo
            //klic metode, ki se sicer zgodi ob zapiranju gradnika ComboBox
            comboBox1_DropDownClosed(this, null);
        }
        catch
        {
            MessageBox.Show("Napaka pri dostopu do baze!\nProjekt se bo
zaprl!");
            Application.Exit();
        }
    }
    //metoda, ki se izvede, ko uporabnik izbere dobavitelja v ComboBox-u
    private void comboBox1_DropDownClosed(object sender, EventArgs e)
    {
        SqlDataAdapter daArtikli;//adapter za prenos podatkov o artiklih
        povezava = new SqlConnection(izvor);
        SqlCommand poizvedba = new SqlCommand();
        //prikazati želimo le artikle, ki pripadajo izbranemu dobavitelju
        poizvedba.CommandText = "SELECT * FROM Artikli WHERE ID_Dobavitelj='"
+ comboBox1.SelectedValue.ToString() + "'";
        poizvedba.Connection = povezava;
        daArtikli = new SqlDataAdapter(poizvedba.CommandText, povezava);
        dsNabavaSQL = new DataSet();
        //objekt dsNabavaSQL povežemo s tabelo Artikli
        daArtikli.Fill(dsNabavaSQL, "Artikli");
        //objekt dataGridView1 povežemo s podatkovnim izvorom
        dataGridView1.DataSource = dsNabavaSQL.Tables[0];
        //Odpremo povezavo z bazo, da se izvede poizvedba
        poizvedba.Connection.Open();
        povezava.Close();//Zapremo povezavo z bazo
    }
}
```

Povezava med dvema tabelama s pomočjo gradnika *DataSet*

Povezavo realizirajmo še s pomočjo gradnika *DataSet*. V projekt dodajmo gradnik *DataSet* (to smo v tem poglavju že počeli), ga poimenujmo *NabavaSQLDataSet* in kliknemo gumb *Add*. Pojavi se prazen *DataSet* v katerem izberemo *DataBase Explorer*, da se na levi strani prikaže okno *DataBase Explorer* in v njem vrstica *DataConnection*. Desno kliknimo na *DataConnection* in izberemo *Add Connection*. Odpre se okno *Add Connection*, v katerem za *Datasource* izberemo *Microsoft SQL Server Database file*, s klikom na gumb *Browse* pa nato piščemo našo bazo *NabavaSQL*. Izbiro potrdimo s klikom na gumb *Open*. V oknu *Database Explorer* se pojavi vrstica *NabavaSQL.mdf*. Razširimo jo, nato razširimo še vrstico *Tables* in v okno *NabavaDataSet* povlecimo tabeli *Artikli* in *Dobavitelji*. Obakrat se na ekranu pojavi sporočilno okno, v katerem izberemo opcijo *Da* – bazo bomo za potrebe razvoja aplikacije prekopirali v naš projekt. Preklopimo na vizuelni pogled obrazca in gradniku *ComboBox* nastavimo tri lastnosti:

- lastnost *DataSource* nastavimo na *Other Data Sources* → *Project Data Sources* → *NabavaDataSet* → *Dobavitelji*;
- lastnost *DisplayMember* nastavimo na *Ime*;
- lastnost *ValueMember* nastavimo na *ID_Dobavitelj*.

Povezava prvega gradnika je gotova. Če projekt poženemo, so v *ComboBox*-u imena vseh dobaviteljev iz tabele *Dobavitelji* baze *NabavaSQL*.

Izbiro dobavitelja v gradniku *ComboBox* moramo sedaj še vezati na prikaz artiklov v gradniku *DataGridView*. Gradnik *DataGridView*, ki je že na obrazcu, preko lastnosti *DataSource* povežemo s tabelo *Artikli* (*Other Data Sources* → *Project Data Sources* → *NabavaDataSet* → *Artikli*). Ustvarimo še odzivno metodo dogodka *Shown* našega obrazca. Vanjo zapišimo *SELECT* stavek, ki bo poskrbel za prenos artiklov le tistega dobavitelja, ki bo izbran v gradniku tipa *ComboBox*. .

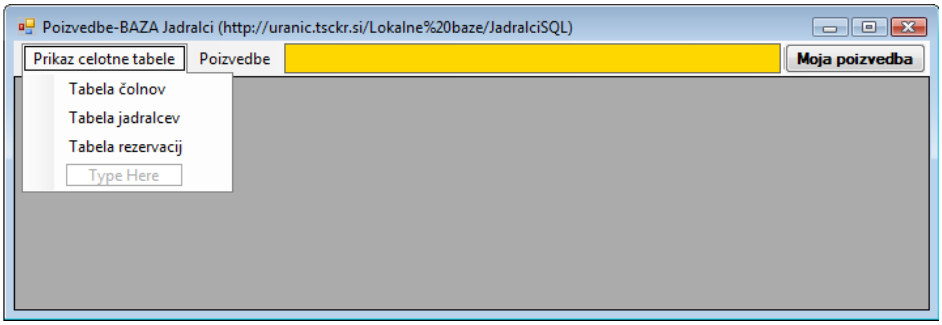
```
private void Form1_Shown(object sender, EventArgs e)
{
    /*s spremembo SELECT stavka dosežemo, da so v gradniku DataGridView
      prikazani le artikli dobavitelja, ki je izbran v gradniku ComboBox.  */
    artikliTableAdapter.Adapter.SelectCommand.CommandText = "SELECT * FROM
Artikli WHERE ID_Dobavitelj='" + comboBox1.SelectedValue.ToString() + "'";
    this.artikliTableAdapter.Fill(this.nabavaSQLDataSet.Artikli);
}
```

Ostane le še odzivna metoda dogodka *DropDownClosed* gradnika *ComboBox*. Izberimo *ComboBox* in v oknu *Properties* odprimo spustni seznam pri tem dogodku. Izberemo že ustvarjeno odzivno metodo *Form1*_Shown. Po vsakem zapiranju gradnika *ComboBox* se izvede nov *SELECT* stavek, ki ima za posledico prikaz artiklov izbranega dobavitelja v gradniku *DataGridVew*.

Poizvedbe

Na strežniku http://uranic.tsckr.si/Lokalne%20baze/JadralciSQL je že pripravljena baza *JadralciSQL* v kateri so tri tabele: *Coln*, *Jadralec* in *Rezervacija*. Iz te baze bi radi naredili nekaj različnih poizvedb in rezultate prikazali v gradniku *DataGridView*.

V projekt najprej vključimo *using* stavek *System.Data.SqlClient*. Na obrazec postavimo gradnik *MenuStrip*, v katerem so tri menijske postavke: dve postavki sta tipa *MenuItem* (*Prikaz celotne tabele* in *Poizvedbe*), tretja pa je tipa *TextBox*, kamor bomo lahko zapisali poljubno poizvedbo. To poizvedbo bomo ustvarili s pomočjo gumba z napisom *Moja poizvedba*, ki ga postavimo ob *TextBox*.



Slika 19: Obrazec za izdelavo poizvedb iz baze *JadralciSQL*.

Postavki glavnega menija naj vsebujeta naslednje poizvedbe:

- Prikaz celotne tabele
  - Tabela čolnov.
  - Tabela jadralcev.
  - Tabela rezervacij.

- Poizvedbe
  - Jadralci starejši od 30 let.
  - Vsi podatki o rdečih čolnih.
  - Seznam rezervacij jadralcev starejših od 50 let in njihovih čolnov.
  - ID čolnov, ki jih je rezerviral Darko.

Za prvo poizvedbo (*Tabela vseh čolnov*) napišimo odzivno metodo, v katero zapišimo vse potrebne stavke za realizacijo te poizvedbe.

```
//Povezovalni niz z bazo podatkov
```

```
string izvor = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\JadralciSQL\Jadralci.mdf;Integrated Security=True;User Instance=True";

//Odzivna metoda menijske vrstice za prikaz tabele vseh čolnov
private void tabelaToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        //besedilo poizvedbe
        string SQL = "select * from Coln";
        //vzpostavimo komunikacijo z bazo in izvedemo poizvedbo
        SqlDataAdapter DAJadralci = new SqlDataAdapter(SQL, izvor);

        //Podatke iz tabele Coln uvozimo v objekt tipa DataTable
        DataTable table = new DataTable();

        /*Nastavitev lokalnih informacij za primerjavo nizov znotraj
          objekta table*/
        table.Locale = System.Globalization.CultureInfo.InvariantCulture;

        //objekt table napolnimo s podatki iz baze
        DAJadralci.Fill(table);
            // ali  tudi takole
            ////DataSet dsJadralci=new DataSet();
            ////DAJadralci.Fill(dsJadralci,"Coln");
            ////dataGridView1.DataSource = dsJadralci.Tables[0];

        /*Dimenzijo stolpcev v DataGridView prilagodimo glede na vsebino*/
dataGridView1.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCellsExcep
tHeader);

        //Podatki v gradniku DataGridView naj bodo ReadOnly
        dataGridView1.ReadOnly = true;
        //za boljši pregled naj bo ozadje sodih vrstic obarvano drugače
        dataGridView1.AlternatingRowsDefaultCellStyle.BackColor =
               Color.LightSteelBlue;
        //Gradnik DataGridView še povežemo s podatkovnim izvorom
        dataGridView1.DataSource = table;

        //Poljuben stolpec lahko skrijemo: skrijemo npr. prvi stolpec
        dataGridView1.Columns[0].Visible = false;
        //Širina stolpcev naj bo npr. taka, da zapolnijo celotno tabelo
        for (int i = 0; i < dataGridView1.Columns.Count; i++)
            dataGridView1.Columns[i].Width = this.Width
               /(dataGridView1.Columns.Count-1);
    }
    catch
    {
        MessageBox.Show("Napaka pri dostopu do baze podatkov!");
```

```
    }
}
```

Napisali smo tudi nekaj stavkov, v katerih smo poskrbeli za lepši prikaz podatkov v celicah gradnika *DataGridView*. Dimenzije stolpcev smo prilagodili glede na vsebino, sode vrstice pa so zaradi boljšega pregleda drugače pobarvane. Širine stolpcev smo določili tako, da bodi vsi stolpci skupaj ravno zapolnili celotno širino tabele.

Za ostale poizvedbe napišimo svojo metodo *Poizvedba*, ki ji bomo za parameter posredovali niz poizvedbe, metoda pa bo rezultat poizvedbe prikazala v gradniku *DataGridView*. V metodo vključimo varovalni blok. Ta bo v primeru napačne poizvedbe, ali pa karšnekoli napake pri komunikaciji z bazo, izvrgel obvestilo o napaki.

```
//Metoda za izdelavo poizvedb - vhodni podatek je ustrezna poizvedba
private void Poizvedba(string SQLPoizvedba)
{
    try
    {
        dataGridView1.DataSource = "";
        //nizu za povezavo z bazo priredimo vrednost parametra metode
        string SQL  = SQLPoizvedba;

        //vzpostavimo komunikacijo z bazo in izvedemo poizvedbo
        SqlDataAdapter dataAdapter = new SqlDataAdapter(SQL, izvor);
        DataTable table = new DataTable();
        dataAdapter.Fill(table);

        //za boljši pregled naj bo ozadje sodih vrstic obarvano drugače
        dataGridView1.AlternatingRowsDefaultCellStyle.BackColor =
Color.LightSteelBlue;

        //Podatki v gradniku DataGridView naj bodo ReadOnly
        dataGridView1.ReadOnly = true;
        //povezava gradnika s podatkovnim izvorom
        dataGridView1.DataSource = table;

        //Širina stolpcev naj bo taka, da zapolnijo celoten DataGridView
        for (int i = 0; i < dataGridView1.Columns.Count; i++)
            dataGridView1.Columns[i].Width = this.Width
/dataGridView1.Columns.Count;
    }
    catch
    {
        MessageBox.Show("Napaka pri dostopu do baze podatkov!");
    }
}
```

Ostale poizvedbe v meniju naredimo tako, da besedilo poizvedbe posredujemo metodi *Poizvedbe*.

```
//Poizvedba za seznam vseh jadralcev iz tabele Jadralec
private void tabelaToolStripMenuItem1_Click(object sender, EventArgs e)
{
    Poizvedba("select * from Jadralec");
}

//Poizvedba za rezervacijami iz tabele Rezervacija
private void tabelaRezervacijToolStripMenuItem_Click(object sender, EventArgs
e)
{
    Poizvedba("select * from Rezervacija");
}

//Poizvedba za jadralci, ki so starejši od 30 let
private void jadralciStarejšiOd30LetToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Poizvedba("select * from Jadralec where starost>30");
}
//Poizvedba o vseh rdečih čolnih
private void vsiPodatkiORdečihČolnihToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Poizvedba("select *from coln where barva='rdeca'");
}

//Poizvedba za rezervacije jadralcev starejših od 50 let in njihovih čolnov
private void
imenaJadralcevInČolnovRezervacijeJadralcevStarejšihOd50LetToolStripMenuItem_C
lick(object sender, EventArgs e)
{
    Poizvedba("select jadralec.ime,Coln.ime from jadralec,coln,rezervacija
where jadralec.jid=rezervacija.jid and rezervacija.cid=coln.cid  and
Jadralec.starost>50");
}

//Poizvedba za vsemi čolni, ki jih je rezerviral Darko
private void iDČolnovKiJihJeRezervToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Poizvedba("Select rezervacija.cid,jadralec.ime from rezervacija JOIN
jadralec ON jadralec.jid=rezervacija.jid where ime='Darko'");
}
```

Ostane nam še odzivna metoda gumba z napisom *Poizvedba*. Uporabnik namreč v tekstovno polje levo od tega gumba lahko zapiše poljubno svojo poizvedbo, ki se bo izvedla ob kliku na ta gumb. Za poizvedbo zopet uporabimo klic metode *Poizvedba*.

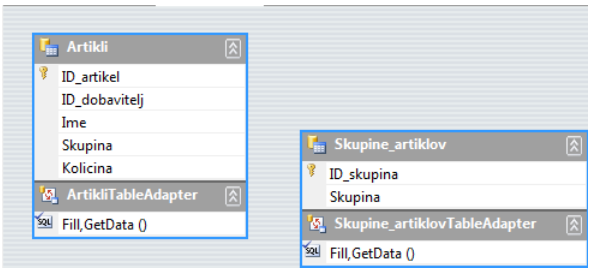
```
//Klic poizvedbe, ki jo vpiše uporabnik v TextBox
private void bPoizvedba_Click(object sender, EventArgs e)
{
    Poizvedba(toolStripTextBox1.Text);
}
```

Transakcije

Kadar je operacija nad podatki še posebno pomembna ali pa sestavljena iz več ukazov, uporabimo transakcijo. To se zgodi pri dodajanju novih zapisov, pri ažuriranju podatkov ali pa pri brisanju podatkov. S transakcijami tudi rešimo težave, ki bi jij sicer lahko imeli v primeru, ko ima brisanje zapisa v eni tabeli za posledico hkratno brisanje določenih zapisov v drugi tabeli. Transakcija običajno (ni pa nujno) povzroči več sprememb v eni ali pa v več tabelah. Predstavlja tudi najboljši način za dodajanje novega zapisa v tabelo ali pa za ažuriranje zapisa v tabeli. Bistvena lastnost transakcij je, da se bodo izvedli vsi *SQL* ukazi znotraj transakcije, ali pa nobeden. Če se torej transakcija ne izvede v celoti, se vse spremembe zavržejo, vzpostavi se prvotno stanje, to je stanje pred transakcijo. Uporabljamo jih torej za zavarovanje podatkov v vsakem primeru, ko obstaja možnost, da med izvajanjem *SQL* stavkov pride do napake in je potrebno vzpostaviti prvotno stanje.

Transakcijo pričnemo z ustvarjanjem novega objekta razreda *SQLTransaction*, končamo pa s stavkom *COMMIT*, ki dokončno potrdi spremembe v bazi, ali pa z ukazom *ROLLBACK* (ki spremembe zavrže).

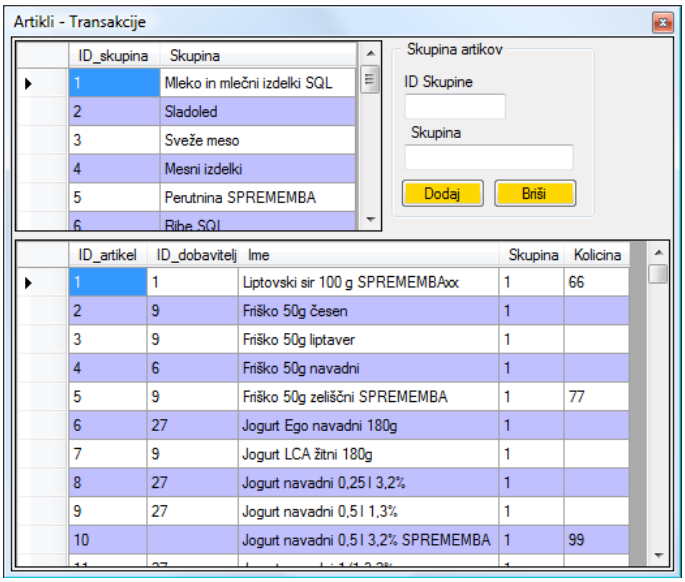
Uporabo transakcij bomo prikazali v naslednjem projektu. V nov projekt dodajmo *DataSet* in ga povežimo z bazo *nabavaSQL*. V *DataSet* dodajmo dva *TableAdapter*-ja za tabeli *Artikli* in *Skupina_artiklov*.



Slika 20: *Dataset* z dvema tabelama.

Na osnovni obrazec dodajmo dva gradnika *DataGridView* in jima določimo lastnost *DataSource* tako, da bomo imeli v zgornjem gradniku (objekt *dataGridView1*) vsebino tabele *Skupine_artiklov*, v spodnjem gradniku (objekt *dataGridView2*) pa vsebino tabele *Artikli*. Na

obrazec postavimo še gradnik *GroupBox* in vanj postavimo dva gradnika *Label*, dva gradnika *TextBox* in dva gumba (*bDodaj* in *bBrisi*).



Slika 21: Obrazec za prikaz transakcij.

Odzivni metodi dogodka *Click* gumba *bDodaj* bomo priredili transakcijo, ki bo v tabelo *Skupina_artiklov* dodala nov zapis (potrebna podatka vnesemo v oba gradnika *TextBox*).

V *Insert* stavek znotraj transakcije bomo vrednosti novih podatkov, ki jih želimo dodati v tabelo, prenesli s pomočjo *Sql* paramatrov. Nov *Sql* parameter določimo takole:

```
SqlCommand dataCommand = new SqlCommand();//Objekt tipa SqlCommand
dataCommand.Parameters.AddWithValue("@imeParametra", imeObjekta);
```

Sql parameter določimo im mu dodelimo vrednost s pomočjo metode *AddWithValue*, ki pripada objektu *dataCommand*, izpeljanemu iz razreda *SqlCommand*. Ime *Sql* parametra je poljubno, oznaka "@" pred imenom pa označuje, da gre za *Sql* parameter. Drugi parameter metode je vrednost, ki jo želimo prirediti *Sql* parametru. Za vstavljanje, ali pa ažuriranje podatkov nekega zapisa potrebujemo metodo *ExecuteNonQuery*. Z metodo *Commit* skušamo dokončno potrditi spremembe va tabeli.

```
private void bDodaj_Click(object sender, EventArgs e)
{
    SqlConnection dataConnection = new SqlConnection();
    dataConnection.ConnectionString = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\NabavaSQL\NabavaSQL.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True";
    dataConnection.Open();
    //Ustvarimo nov objekt za transakcijo
    SqlTransaction transakcija = dataConnection.BeginTransaction();
```

```
    //Ustvarimo nov objekt za SQL ukaz
    SqlCommand dataCommand = new SqlCommand();
    dataCommand.Connection = dataConnection;
    dataCommand.Transaction = transakcija;
    try
    {
        int novID = Convert.ToInt32(textBox1.Text);
        string novaSkupina = textBox2.Text;
        //nove vrednosti dodamo v tabelo preko Sql parametrov @ID in @novaSK
        dataCommand.Parameters.AddWithValue("@ID", novID);
        dataCommand.Parameters.AddWithValue("@novaSk", novaSkupina);
        dataCommand.CommandText = "Insert INTO Skupine_artiklov
(ID_skupina,Skupina) VALUES (@ID,@novaSk)";

        /*lahko pa tudi takole
        dataCommand.Parameters.Add(newSqlParameter("@ID",textBox1.Text));
        dataCommand.Parameters.Add(new SqlParameter("@novaSK",
                                    textBox2.Text));
        dataCommand.CommandText = "Insert INTO Skupine_artiklov
               (ID_skupina,Skupina) VALUES (@ID,@novaSk)";*/

        dataCommand.CommandType = CommandType.Text;
        dataCommand.ExecuteNonQuery();
        //dokončno potrdimo spremembe v bazi
        transakcija.Commit();
        MessageBox.Show("Dodajanje nove skupine uspešno!");
    }
    catch (Exception ep)
    {
        //Če je prišlo do napake, v bazi vzpostavimo prvotno stanje
        transakcija.Rollback();
        MessageBox.Show("Napaka pri shranjevanju podatkov!");
    }
    finally
    {
        dataConnection.Close();
        //Ažuriramo vsebino dataSet1 in s tem tudi gradnika dataGridView1
this.skupine_artiklovTableAdapter.Fill(this.dSNabavaSQL.Skupine_artiklov);
        this.artikliTableAdapter.Fill(this.dSNabavaSQL.Artikli);
    }

}
```

Odzivni metodi dogodka *Click* gumba *Briši* pa bomo priredili transakcijo, ki bo v tabeli *Artikli* pobrisala vse zapise, v katerih nastopa podatek *ID Skupine*, ki ga zapišemo v gradnik *textBox1*.

```
private void bBrisi_Click(object sender, EventArgs e)
{
    SqlConnection dataConnection = new SqlConnection();
```

```
    dataConnection.ConnectionString = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\NabavaSQL\NabavaSQL.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True";

    dataConnection.Open();
    //Ustvarimo nov objekt za SQL transakcijo v SQL bazi podatkov
    SqlTransaction myTrans = dataConnection.BeginTransaction();
    //Ustvarimo nov objekt za SQL stavek, ki ga bomo izvedli v SQL bazi
    SqlCommand dataCommand = new SqlCommand();
    dataCommand.Connection = dataConnection;
    dataCommand.Transaction = myTrans;
    try
    {
        //shranimo vrednost v celici ID_skupina izbrane vrstice
        int ID = Convert.ToInt32(textBox1.Text);
        /*@ID je parameter, preko katerega bomo transakciji posredovali
         vrednost spremenljike ID*/
        dataCommand.Parameters.AddWithValue("@ID", ID);
        /*oblikujemo SQL ukaz za brisanje vseh artiklov iz tabele Artikli,
          ki pripadajo skupini zapisani v textBox1*/
        dataCommand.CommandText = "delete Artikli where Skupina=@ID";
        //Poizkusimo izvesti dejansko brisanje vseh artiklov iz te skupine
        dataCommand.ExecuteNonQuery();
        myTrans.Commit();//dokončno shranimo narejene spremembe
    }
    catch (Exception ep)
    {
        myTrans.Rollback();//v primeru napake se vzpostavi prejšnje stanje
        MessageBox.Show("Napaka pri brisanju podatkov!");
    }
    finally
    {
        dataConnection.Close();//na koncu zapremo povezavo z bazo podatkov
        this.artikliTableAdapter.Fill(this.dSNabavaSQL.Artikli);
    }
}
```

Kadar bomo pisali aplikacijo namenjeno uporabnikom, ki bodo do nje dostopali npr. preko omrežnih povezav ali celo preko interneta (*remote users*), bomo v stavku *ConnectionString* zapisali tudi uporabniško ime in geslo, npr. takole:

```
string connetionString = "Data Source=ServerName;Initial
Catalog=DatabaseName;User ID=UserName;Password=Password";
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Povzetek

Projekti, ki delajo z bazo podatkov so med najzahtevnejšimi. Pomagamo si lahko s čarovnikom, ki preko gradnika *DataSet* in številnih vgrajenih razredov omogoča dokaj lagodno manipulacijo s tabelami poljubne baze podatkov. Spoznali pa smo tudi transakcije, ki predstavljajo najbolj zanesljiv način za urejanje tabel in za dodajanje novih zapisov. Pri ustvarjanju novih projektov nam bodo opisani postopki in metode v veliko pomoč, glede na naravo problema, ki ga moramo rešiti, pa se bomo odločili med uporabo čarovnika in transakcijskim modelom.

Države

Pridobljeno znanje za delo z bazo podatkov bomo sedaj uporabili za izdelavo večokenskega projekta *DržaveNaSvetu*, v katerem bomo prikazali popolno manipulacijo z bazo *DrzaveSQL*. Projekt bomo izdelali povsem programsko, torej brez uporabe gradnika *DataSet* oz. brez uporabe čarovnikov. Po navodilih iz začetka poglavja o bazah najprej ustvarimo bazo *DrzaveSQL* z dvema tabelama (*Drzave* in *Kontinenti*).

Struktura tabele *Drzave* naj bo takale:

- ID_Drzava: int (ključno polje*, Identity Specification* nastavimo na *Yes*, kar pomeni, da je polje *Autoincrement*, oz. da se bo *ID*- številka države določala avtomatično in sicer od 1 naprej, (*Allow Nulls:* NE);
- Drzava: nvarchar(255), (*Allow Nulls:* DA);
- Kontinnet: nvarchar(255), (*Allow Nulls:* DA);
- Povrsina: float, (*Allow Nulls:* DA);
- Prebivalstvo: float, (*Allow Nulls:* DA);
- BDP: float, (*Allow Nulls:* DA).

Struktura tabele *Kontinent*:

- ID_Kontinenta: int (ključno polje, *Identity Specification* naj bo *Yes, Allow Nulls*: NE);
- Ime: nvarchar(255) (*Allow Nulls*: NE).

V obe tabeli že med njunim ustvarjanjem vnesimo nekaj testnih podatkov, npr:

SRECO-PC\...\DRZ... - dbo.Kontinenti

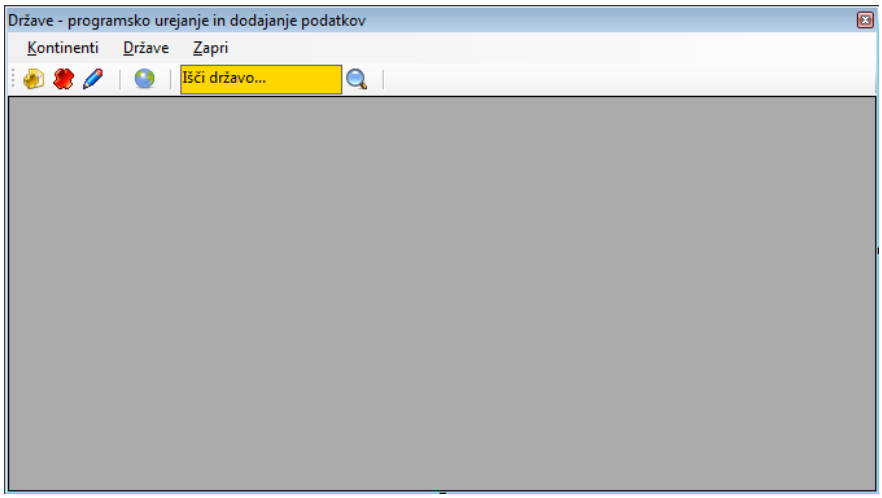
ID_Kontinenta	Ime
1	Evropa
2	Afrika
3	Avstralija
4	Oceanija
5	Azija
6	Severna Amerika
7	Južna Amerika
8	Srednja Amerika
9	Antarktika

SRECO-PC\...\DRZ....MDF - dbo.Drzave | SRECO-PC\...\DRZ... - dbo.Kontinenti | SRECO-PC\...\DRZ....MDF - dbo.Drzave

ID_Drzava	Drzava	Kontinent	Povrsina	Prebivalstvo	BDP
2	Združeno kraljes...	Evropa	244820	59511464	1290000000000
3	Združeni Arabski...	Azija	82880	2369153	41500000000
4	Združene držav...	Severna Amerika	9629091	275562673	9255000000000
5	Zambija	Afrika	752614	9582418	8500000000
6	West Bank	Azija	5860	2020298	3300000000
7	Vietnam	Azija	329560	78773873	143100000000
8	Venezuela		912050	23542649	182800000000

Slika 22: Testni podatki v tabelah *Kontinenti* in *Drzave* baze *DrzaveSQL*.

Glavni obrazec projekta se imenuje *FDrzave*, vsebuje naj menijsko vrstico in orodjarno, pod njima pa postavimo gradnik *DataGridView*, ki mu lastnost *Dock* nastavimo na *Fill*.



Slika 23: Glavni obrazec projekta *DržaveNaSvetu.*

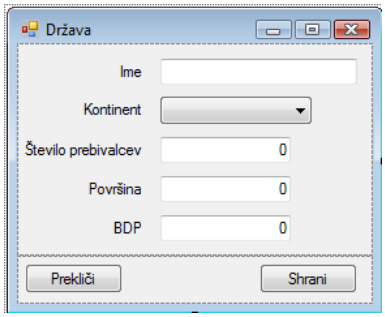
Vrstica z menijem vsebuje tri postavke: *Kontinenti*, *Države*, ter *Zapri*. Postavka *Države* naj vsebuje še podmeni z možnostmi *Dodaj državo*, *Uredi izbrano državo* in *Briši izbrano državo*. Gumbi v orodjarni so le bližnjice za postavke v menijski vrstici (*Dodaj državo*, *Brisanje izbrane države*, *Urejanje izbrane države* in *Kontinenti*). Dodano je še vnosno polje za vpis imena države in gumb za iskanje te države v tabeli.

Projekt vsebuje še tri podobrazce. Ker pa želimo v podobrazcih dostopati do objektov glavnega obrazca, moramo najprej ustrezno spremeniti projektno datoteko *Program.cs*.

```
class Program
{
```

```
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    // NAPOVED novega objekta tipa FDrzave (objekt bo STATIČEN)
    public static FDrzave drzave;
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        //ustvarjanje novega objekta tipa Fdrzave
        drzave = new FDrzave();
        Application.Run(drzave);
    }
}
```

Pred pisanjem odzivnih metod, ustvarimo še podobrazec za dodajanje oz. ažuriranje nove države (*Drzava.cs).* Vsi gradniki naj imajo lastnost *Modifiers* nastavljeno na *Public*. V gradnik tipa *ComboBox* bomo s pomočjo poizvedbe uvozimli podatke iz tabele *Kontinenti*. Prikazano je polje *ime* kontinenta, lastnost *ValueMember* gradnika pa je nastavljena na polje *Id_Kontinenta*. Gumba na obrazcu sta modalna: gumb *Shrani* vrne vrednost *DialogResult.OK*, gumb Pekliči pa *DialogResult.Cancel*.



Slika 24: Podobrazec za dodajanje nove države in za ažurirиranje podatkov o izbrani državi.

Lotimo se sedaj odzivnih metod glavnega obrazca. V njem najprej napovemo vse potrebne objekte za delo z bazo podatkov. Zapišimo tudi povezovalni niz. Uporabljali ga bomo v vseh obrazcih našega projekta, zato ga ustvarimo izven vseh metod. Povezavo z bazo in prikaz celotne tabele *Drzave* bomo na glavnem obrazcu ustvarili s pomočjo konstruktorja. Dodana je še odzivna metoda za zapiranje obrazca.

```
Public partial class Fdrzave : Form
{
    SqlConnection dataConnection;
    SqlDataAdapter daDrzave;
    /*objekti za povezavo z bazo so javni zato, da jih lahko uporabimo v
     podrejenih obrazcih*/
```

```
    public DataSet dsDrzave;
    //povezovalni niz
    public string izvor = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\DOKUMENTI\APJ+ŠOLA\APJ\SQL
SERVER\DrzaveSQL\DrzaveSQL.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True";
    //Konstruktor obrazca
    public Fdrzave()
    {
        InitializeComponent();
        try
        {
            //Ko se obrazec odpre bo v tabeli že seznam vseh vnesenih držav
            string SQL = "select * from Drzave";
            dataConnection = new SqlConnection(izvor);
            dsDrzave = new DataSet();
            daDrzave = new SqlDataAdapter(SQL, dataConnection);
            daDrzave.Fill(dsDrzave, "Drzava");
            dataGridView1.DataSource = dsDrzave.Tables[0];
        }
        catch
        {
            MessageBox.Show("Napaka pri dostopu do baze podatkov! ");
        }
    }
    //ZAPIRANJE obrazca
    private void zapriToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Close();
    }
}
```

V odzivna metoda za dodajanje nove države najprej ustvarimo nov objekt razreda *Drzava*. Tako ustvarjen obrazec odpremo modalno. če je uporabnik vanj uspešno vnesel podatke, ustvarimo transakcijo tipa *insert*. Za dodajanje novih podatkov v tabelo potrebujemo kar nekaj *Sql* parametrov, ki jim bomo dali imena, ki nas bodo asociirala na njihov pomen. Z metodo *ExecuteNonQuery* sprožimo vstavljanje v tabelo, z metodo *Commit* pa dokončno potrditi narejene spremembe va tabeli.

```
//DODAJANJE nove države
private void dodajToolStripMenuItem_Click(object sender, EventArgs e)
{
    Drzava Nova = new Drzava();
    Nova.Text = "Vnos nove države!";
    //Ko se obrazec odpre, bo v spustnem seznamu že prvi kontinent
    Nova.comboBox1.SelectedIndex = 0;

    if (Nova.ShowDialog() == DialogResult.OK)
    {
        //Pred začetkom transakcije odpremo povezavo z bazo podatkov
```

```
        dataConnection.Open();
        SqlTransaction myTrans = dataConnection.BeginTransaction();
        SqlCommand dataCommand = new SqlCommand();
        dataCommand.Connection = dataConnection;
        dataCommand.Transaction = myTrans;
        try
        {
            //deklaracija SQL parametrov za SQL transacijo
            dataCommand.Parameters.Add(new SqlParameter("@Drzava",
                Nova.textBox1.Text));
            dataCommand.Parameters.Add(new SqlParameter("@Kontinent",
                Nova.comboBox1.Text));
            dataCommand.Parameters.Add(new SqlParameter("@Povrsina",
                Nova.textBox2.Text));
            dataCommand.Parameters.Add(new SqlParameter("@Prebivalstvo",
                Nova.textBox3.Text));
            dataCommand.Parameters.Add(new SqlParameter("@BDP",
                Nova.textBox4.Text));
            //ukazni niz za vstavljanje novega zapisa v tabelo
            dataCommand.CommandText = "Insert INTO Drzave (Drzava,Kontinent,
Povrsina,Prebivalstvo,BDP) VALUES (@Drzava,@Kontinent,@Povrsina,@Prebivalstvo
,@BDP)";
            dataCommand.CommandType = CommandType.Text;
            //za zapis v tabelo potrebujemo metodo ExecuteNonQuery)
            dataCommand.ExecuteNonQuery();
            myTrans.Commit();//poizkus zapisa v tabelo
        }
        catch
        {
            //Če je prišlo do napake, vzpostavimo prejšnje stanje
            myTrans.Rollback();
            MessageBox.Show("Šifra skupine že obstaja ali pa napaka pri
shranjevanju!");
        }
        string SQL = "SELECT * FROM Drzave";
        dataConnection = new SqlConnection(izvor);
        daDrzave = new SqlDataAdapter(SQL, dataConnection);
        dsDrzave = new DataSet();
        daDrzave.Fill(dsDrzave, "Drzava");
        dataGridView1.DataSource = dsDrzave.Tables[0];
    }
}
```

Po uspešno izvedeni transakciji smo poskrbeli za osveževanje vsebine gradnika *DataGridView*, v katerem se pojavi novo vnesena država.

Za ažuriranje podatkov izbranega zapisa prav tako uporabimo objekt razreda *Drzava*. Razlika med dodajanjem in ažuriranjem pa je seveda v tem, da moramo pred prikazom obrazca za ažuriranje poskrbeti, da so na njem podatki izbranega zapisa. Ker smo vsem vizuelnim objektom

na obrazcu *Drzava* nastavili lastnost *Modifiers* na *Public*, to sedaj ne bo težko. Iz osnov *OOP* in poglavja o večokenskih aplikacijah vemo, da do njih dostopamo s pomočjo operatorja pika. Vrednosti podatkov bomo pobirali iz celic in jih kasneje prav tako shranjevali v celice gradnika *DataGridView*. To pa smo se naučili že v poglavju, v keterem smo podrobno predstavili ta gradnik.

Tokratna transakcija bo tipa *update*. Po uspešni izvedbi moramo še poskrbeti za osveževanje vsebine gradnika *DataGridView*. Podatki države, ki smo jih ravnokar spreminjali, bodo spremenejni tudi na ekranu.

```
//AŽURIRANJE podatkov o izbrani državi pri dvokliku na DataGridView
private void dataGridView1_DoubleClick(object sender, EventArgs e)
{
    Drzava Azuriraj = new Drzava();
    Azuriraj.Text = "Ažuriranje podatkov izbrane države države!";
    //podatek izbrane vrstice gradnika DataGridView prenesemo na obrazec
    Azuriraj.textBox1.Text = Convert.ToString(
                                dataGridView1.CurrentRow.Cells[1].Value);
    int poz=Azuriraj.comboBox1.FindString(Convert.ToString(
            dataGridView1.CurrentRow.Cells[2].Value),0);
    Azuriraj.comboBox1.SelectedIndex = poz;
    Azuriraj.textBox2.Text = Convert.ToString(
                                dataGridView1.CurrentRow.Cells[3].Value);
    Azuriraj.textBox3.Text = Convert.ToString(
                                dataGridView1.CurrentRow.Cells[4].Value);
    Azuriraj.textBox4.Text = Convert.ToString(
                                dataGridView1.CurrentRow.Cells[5].Value);
    int pozicija = Convert.ToInt32(dataGridView1.CurrentRow.Cells[0].Value);
    if (Azuriraj.ShowDialog() == DialogResult.OK)
    {
        //Pred začetkom transakcije odpremo povezavo z bazo podatkov
        dataConnection.Open();
        SqlTransaction myTrans = dataConnection.BeginTransaction();
        SqlCommand dataCommand = new SqlCommand();
        dataCommand.Connection = dataConnection;
        dataCommand.Transaction = myTrans;
        try
        {
            //deklaracija parametrov za SQL transacijo
            dataCommand.Parameters.Add(new SqlParameter("@ID_Drzava",
               pozicija));
            //ali lahko tudi:
            dataCommand.Parameters.AddWithValue("@ID_Drzava",pozicija);
            dataCommand.Parameters.Add(new SqlParameter("@Drzava",
               Azuriraj.textBox1.Text));
            dataCommand.Parameters.Add(new SqlParameter("@Kontinent",
               Azuriraj.comboBox1.Text));
            dataCommand.Parameters.Add(new SqlParameter("@Povrsina",
```

```
                Azuriraj.textBox2.Text));
            dataCommand.Parameters.Add(new SqlParameter("@Prebivalstvo",
                Azuriraj.textBox3.Text));
            dataCommand.Parameters.Add(new SqlParameter("@BDP",
                Azuriraj.textBox4.Text));
            dataCommand.CommandText = "Update Drzave  SET Drzava=@Drzava,
                    Kontinent=@Kontinent,Povrsina=@Povrsina,Prebivalstvo=
                    @Prebivalstvo,BDP=@BDP WHERE ID_Drzava=@ID_Drzava";
            dataCommand.CommandType = CommandType.Text;
            dataCommand.ExecuteNonQuery();
            //če je vse OK, podatke dokončno shranimo v tabelo
            myTrans.Commit();
        }
        catch
        {
            //Če je prišlo do napake, vzpostavimo prejšnje stanje
            myTrans.Rollback();
            MessageBox.Show("Napaka pri shranjevanju!");
        }
        string SQL = "SELECT * FROM Drzave";
        dataConnection = new SqlConnection(izvor);
        daDrzave = new SqlDataAdapter(SQL, dataConnection);
        dsDrzave = new DataSet();
        daDrzave.Fill(dsDrzave, "Drzava");
        dataGridView1.DataSource = dsDrzave.Tables[0];
    }
}
```

Dodajmo še odzivno metodo za brisanje zapisa iz tabele. Pred brisanjem uporabnika v sporočilnem oknu obvestimo, katero vrstica bo pobrisana. Če uporabnik brisanje potrdi naredimo transakcijo tipa *delete*. Po brisanju zopet poskrbimo za osveževanje zapisov na obrazcu.

```
//BRISANJE izbrane države
private void brisiToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Brišem državo " +
dataGridView1.CurrentRow.Cells[1].Value.ToString(), "Brisanje vrstice",
MessageBoxButtons.OKCancel, MessageBoxIcon.Question) == DialogResult.OK)
    {
        //zapomnim si šifro države, ki jo želim pobrisati
        int sifra = Convert.ToInt32(dataGridView1.CurrentRow.Cells[0].Value);
        dataConnection.Open();
        SqlTransaction myTrans = dataConnection.BeginTransaction();
        SqlCommand dataCommand = new SqlCommand();
        dataCommand.Connection = dataConnection;
        dataCommand.Transaction = myTrans;
        try
        {
```

```
            dataCommand.Parameters.Add(new SqlParameter("@ID_Drzava",sifra));

            //ali tudi takole:
            //SqlParameter param1 = new SqlParameter();
            //param1.ParameterName = "@ID_Drzava";
            //param1.Value = sifra;
            //dataCommand.Parameters.Add(param1);

            dataCommand.CommandText = "DELETE FROM Drzave WHERE ID_Drzava
                                       =@ID_Drzava";
            dataCommand.ExecuteNonQuery();
            //če je vse OK, podatke dokončno shranimo v tabelo
            myTrans.Commit();
        }
        catch
        {
            myTrans.Rollback();
            MessageBox.Show("Šifra skupine že obstaja ali pa napaka pri
shranjevanju!");
        }
        //Osvežimo podatke v dataGridView1 glede na novo stanje bazi
        string SQL = "SELECT * FROM Drzave";
        dataConnection = new SqlConnection(izvor);
        daDrzave = new SqlDataAdapter(SQL, dataConnection);
        dsDrzave = new DataSet();
        daDrzave.Fill(dsDrzave, "Drzava");
        dataGridView1.DataSource = dsDrzave.Tables[0];
    }
}
```

Uporabniku želimo ponuditi še možnost iskanja oziroma preverjanja, ali je neka država že v tabeli. V odzivni metodi bomo preverjanje izvedli kar v gradniku *DataGridView*. Če najdemo iskano državo, si zapomnimo njen indeks. Preko tega indeksa lahko nato na koncu pridemo do vseh podatkov iskane države.

```
//ISKANJE države, katere ime je zapisano v textBox1
private void toolStripButton5_Click(object sender, EventArgs e)
{
    /*v spremenljivko pozicija bomo shranili indeks najdene države - če
      države ne najdemo, ima spremenljivka vrednost -1*/
    int pozicija=-1;
    for (int i = 0; i < dataGridView1.Rows.Count; i++)
    {
        if (Convert.ToString(dataGridView1.Rows[i].Cells[1].Value) ==
toolStripTextBox1.Text)
        {
            pozicija = i;
            break;
        }
```

```
    }
    if (pozicija == -1)
        MessageBox.Show("Ta država ne obstaja!");
    else
    {
        //v gradniku DataGridView se postavimo na najdeno državo
        dataGridView1.Rows[pozicija].Selected = true;
        /*če hočemo, da bo izbrana vrstica tudi aktivna, določimo še
          aktivno celico te vrstice*/
        dataGridView1.CurrentCell = dataGridView1.Rows[pozicija].Cells[0];
    }
}
```

Naslednja odzivna metoda je prirejena dogodku *Enter*, ki se zgodi ob vstopu uporabnikove miške v gradnik *TextBox*. Dosedanja vsebina vnosnega polja se ob vsakem vstopu briše.

```
//BRISANJE vsebine gradnika textBox1
private void toolStripTextBox1_Enter(object sender, EventArgs e)
{
    toolStripTextBox1.Clear();
}
```

Na vrsti so odzivne metode razreda *Drzava*. Objekte tega razreda bomo, kot smo že videli, potrebovali za vnašanje novih držav, ter za spreminjanje podatkov obstoječih držav. Obrazec smo priravili že prej, napisati morame le še potrebne odzivne metode.

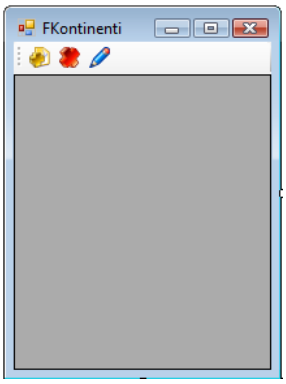
V konstruktorju poskrbimo za prenos vsebine tabele *Kontinenti* v gradnik tipa *ComboBox*, ki je na obrazcu. Nekaj podobnega smo delali že v eni od vaj, ne smemo pa pozabiti na lastnosti *DisplayMember* in *ValueMember*. S prvo povemo, kateri podatki bodo v spustnem seznamu prikazani, z drugo pa kateri podatek bo vrnjen ob izbiri. Dodali bomo že odzivno metodo dogodka *KeyPress*, ki jo bomo priredili vnosnim poljem. Pred zapirajem obrazca pa bomo še preverili, če je uporabnik korektno izpolnil vsa vnosna polja in izbral kontinent.

```
//konstruktor odzivne metode razreda Drzava
public partial class Drzava : Form
{
    //Konstruktor
    public Drzava()
    {
        InitializeComponent();
        /*v gradnik ComboBox programsko uvozimo vse kontinente, ki se
          nahajajo v tabeli Kontinenti*/
        SqlConnection povezava = new SqlConnection();//Objekt za povezavo
        DataSet dsDrzaveSQL; //DataSet za shranjevanje prenešenih podatkov
        SqlDataAdapter daKontinenti;//adapter za prenos dobaviteljev
        try
        {
            povezava = new SqlConnection(Program.drzave.izvor);
```

```
            SqlCommand poizvedba = new SqlCommand();
            poizvedba.CommandText = "SELECT * FROM Kontinenti";
            poizvedba.Connection = povezava;
            daKontinenti = new SqlDataAdapter(poizvedba.CommandText,
                                              povezava);
            dsDrzaveSQL = new DataSet();
            //objekt dsDrzaveSQL povežemo s tabelo Kontinenti
            daKontinenti.Fill(dsDrzaveSQL,"Kontinenti");
            //objekt comboBox1 povežemo s podatkovnim izvorom
            comboBox1.DataSource = dsDrzaveSQL.Tables[0];
            //prikazana bodo imena Kontinentov
            comboBox1.DisplayMember = "Ime";
            /*ob izbiri kontinenta bo vrednost izbire ID tega kontinenta*/
            comboBox1.ValueMember = "ID_Kontinenta";
            //Odpremo povezavo z bazo, da se izvede poizvedba
            poizvedba.Connection.Open();
            povezava.Close();//Zapremo povezavo z bazo
        }
        catch
        {
            MessageBox.Show("Napaka pri dostopu do baze!\nProjekt se bo
                             zaprl!");
            Application.Exit();
        }
    }

    //dogodek KeyPress gradnika texBox2
    private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
    {
        //dovoljen le vnos celega števila (številke in tipka BackSpace)
        if (((e.KeyChar < '0') || (e.KeyChar > '9'))&& (e.KeyChar !=
          (char)(8)))
            e.Handled = true;
    }

    //Dogodek gumba Shrani
    private void button1_Click(object sender, EventArgs e)
    {
        //Če uporabnik ne bo vnesel števila prebivalcev, se okno ne bo zaprlo
        if (textBox2.Text.Trim() == "")
        {
            MessageBox.Show("Vnesi število prebivalcev!");
            this.DialogResult = DialogResult.None;
        }
        //Če uporabnik ne bo vnesel površine, se okno ne bo zaprlo
        else if (textBox3.Text.Trim() == "")
        {
            MessageBox.Show("Vnesi površino!");
            this.DialogResult = DialogResult.None;
        }
```

```
        //Če uporabnik ne bo vnesel BDP, se okno ne bo zaprlo
        else if (textBox4.Text.Trim() == "")
        {
            MessageBox.Show("Vnesi BDP!");
            this.DialogResult = DialogResult.None;
        }
    }
}
```

Za ogled kontinentov, ki obstajajo v tabeli *Kontinenti* prav tako potrebujemo svoj obrazec, ki ga poimenujmo *FKontinenti.cs*. Vsebuje naj menijsko vrstico s tremi gumbi za dodajanje novega kontinenta, brisanje izbranega kontinenta in urejanje izbranega kontinenta. Podatki iz tabele bodo zopet prikazani v gradniku tipa *DataGridView*, ki mu lastnost *Dock* nastavimo na *Fill*.



Slika 25: Obrazec za prikaz, urejanje, brisanje in dodajanje kontinentov.

Obrazec bomo odpirali iz glavnega obrazca, zato imamo v glavnem meniju le-tega že pripravljeno postavko *Kontinenti*. Tule je odzivna metoda:

```
//PRIKAZ kontinentov
private void kontinentiToolStripMenuItem_Click(object sender, EventArgs e)
{
    FKontinenti Kont = new FKontinenti();
    Kont.ShowDialog();
}
```

Na obrazcu *FKontinenti* ni nobenega modalnega gumba, zato smo metodo *ShowDialog* klicali kar samostojno.

Kar nekaj dela imamo še z odzivnimi metodami tega obrazca. Kontinente želimo tudi dodajati, spreminjati njihove podatke in jih brisati iz tabele. V konstruktorju najprej poskrbimo za prenos vsebine tabele *Kontinenti* iz baze *Drzave*.

```
public partial class FKontinenti : Form
{
    SqlConnection dataConnection;
```

```
    SqlDataAdapter daKontinenti;
    DataSet dsDrzave; //DataSet za shranjevanje prenešenih tabel
    public FKontinenti()
    {
        InitializeComponent();
        //Klic metode za prikaz podatkov in oblik. gradnika DataGridView
        OblikujDataGridView();
    }
    //metoda za prikaz podatkov v dataGridView1 in za njegovo oblikovanje
    private void OblikujDataGridView()
    {
        string SQL = "select * from Kontinenti";
        dataConnection = new SqlConnection(Program.drzave.izvor);
        dsDrzave = new DataSet();
        daKontinenti = new SqlDataAdapter(SQL, dataConnection);
        daKontinenti.Fill(dsDrzave, "Kontinenti");
        dataGridView1.DataSource = "";
        /*iz dsDrzave prikažemo tabelo z indeksom 0 - Kontinetni (v tabeli
          z indeksom 0 so države*/
        dataGridView1.DataSource = dsDrzave.Tables[0];
        //prvi stolpec (ID kontinenta) v dataGridView1 ne bo prikazan
        dataGridView1.Columns[0].Visible = false;
        dataGridView1.Columns[1].Width = 150;
        dataGridView1.AllowUserToAddRows= false;//dodajanje vrstic ni možno
         //brisanje vrstic ni možno
        dataGridView1.AllowUserToDeleteRows= false;
        //urejanje neposredno na obrazcu NI možno
        this.dataGridView1.ReadOnly = true;
        dataGridView1.Rows[0].Selected = true;
        /*če hočemo, da bo izbrana vrstica tudi aktivna, določimo še
         aktivno celico te vrstice*/
        dataGridView1.CurrentCell = dataGridView1.Rows[0].Cells[1];
        dataConnection.Close();
    }
}
```

Odzivne metode za brisanje, dodajanje in urejanje podatkov iz tabele *Kontinenti* so zelo podobne tistim, ki smo jih uporabili pri delu s tabelo *Drzava*. Za dodajanje in ažuriranje bomo potrebovali tudi nov obrazec, na katerem bo eno samo vnosno polje in dva modalna gumba. Tule so najprej vse tri odzivne metode.

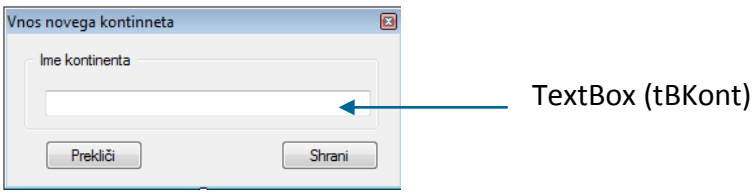
```
    //BRISANJE izbranega kontinneta
    private void toolStripButton2_Click(object sender, EventArgs e)
    {
        if (MessageBox.Show("Brišem kontinent
"+dataGridView1.CurrentRow.Cells[1].Value + "?","BRISANJE
POSTAVKE",MessageBoxButtons.YesNo,MessageBoxIcon.Question)==DialogResult.Yes)
        {
            /*POZOR: pred brisanjem kontinenta bi morali še preveriti, ali
```

```
            v tabeli Drzave obstaja kaka država iz tega kontinenta in v
            tem primeru brisanje preprečiti!!!*/
            int sifra;
            sifra=Convert.ToInt32(dataGridView1.CurrentRow.Cells[0]. Value);
            dataConnection.Open();
            SqlTransaction myTrans = dataConnection.BeginTransaction();
            SqlCommand dataCommand = new SqlCommand();
            dataCommand.Connection = dataConnection;
            dataCommand.Transaction = myTrans;
            try
            {
                dataCommand.Parameters.Add(new SqlParameter("@ID_Kont",
                                           sifra));
                dataCommand.CommandText = "DELETE FROM Kontinenti WHERE
                                           ID_Kontinenta =@ID_Kont";
                dataCommand.ExecuteNonQuery();
                myTrans.Commit();
            }
            catch
            {
                //Če je prišlo do napake, vzpostavimo prejšnje stanje
                myTrans.Rollback();
                MessageBox.Show("Napaka pri shranjevanju!");
            }
            dataConnection.Close();
            OblikujDataGridView();
        }
    }
    //DODAJANJE novega kontinenta
    private void toolStripButton1_Click(object sender, EventArgs e)
    {
        FNovKontinent novK = new FNovKontinent();
        if (novK.ShowDialog() == DialogResult.OK)
        {
            //Pred začetkom transakcije odpremo povezavo z bazo podatkov
            dataConnection.Open();
            SqlTransaction myTrans = dataConnection.BeginTransaction();
            SqlCommand dataCommand = new SqlCommand();
            dataCommand.Connection = dataConnection;
            dataCommand.Transaction = myTrans;
            try
            {
                //deklaracija parametrov za SQL transacijo
                dataCommand.Parameters.Add(new SqlParameter("@Ime",
                                           novK.tBKont.Text));
                dataCommand.CommandText = "Insert INTO Kontinenti (ime)
                                           VALUES (@Ime)";
                dataCommand.CommandType = CommandType.Text;
                dataCommand.ExecuteNonQuery();
                myTrans.Commit();
```

```
            }
            catch
            {
                //Če je prišlo do napake, vzpostavimo prejšnje stanje
                myTrans.Rollback();
                MessageBox.Show("Napaka pri shranjevanju!");
            }
            dataConnection.Close();
            OblikujDataGridView();
        }
    }
    //UREJANJE izbranega kontinenta
    private void toolStripButton3_Click(object sender, EventArgs e)
    {
        FNovKontinent novK = new FNovKontinent();
        //POZOR:tBKont mora imeti lastnost Modifiers nastavljeno na Public
       novK.tBKont.Text = dataGridView1.CurrentRow.Cells[1].Value.ToString();
        /*POZOR: gradnik groupBox1 mora imeti lastnost Modifiers
          nastavljeno na Public*/
        novK.groupBox1.Text = "Ažuriranje imena kontinenta";
        if (novK.ShowDialog() == DialogResult.OK)
        {
            //Pred začetkom transakcije odpremo povezavo z bazo podatkov
            dataConnection.Open();
            SqlTransaction myTrans = dataConnection.BeginTransaction();
            SqlCommand dataCommand = new SqlCommand();
            dataCommand.Connection = dataConnection;
            dataCommand.Transaction = myTrans;
            try
            {
                //deklaracija parametrov za SQL transacijo
                dataCommand.Parameters.Add(new SqlParameter("@Ime",
                  novK.tBKont.Text));
                dataCommand.Parameters.Add(new SqlParameter("@Kont",
                  Convert.ToInt32(dataGridView1.CurrentRow.Cells[0].Value)));
                /*spremenimo le ime kotinenta, ID ostane seveda enak
                  (ključno polje)*/
                dataCommand.CommandText = "Update Kontinenti SET Ime=@Ime
                                           WHERE ID_Kontinenta=@Kont";
                dataCommand.CommandType = CommandType.Text;
                dataCommand.ExecuteNonQuery();
                myTrans.Commit();
            }
            catch
            {
                myTrans.Rollback();
                MessageBox.Show("Napaka pri shranjevanju!");
            }
            dataConnection.Close();
            OblikujDataGridView();
```

```
        }
    }
}
```

Ostane nam še obrazec za dodajanje oz. urejanje kontinenta(*FNovKontinent.cs*). Gumba na obrazcu sta modalna: gumb *Shrani* vrne vrednost *DialogResult.OK*, gumb *Prekliči* pa *DialogResult.Cancel*.



Slika 26: Obrazec dodajanje/ažuriranje kontinenta.

```
public partial class FNovKontinent : Form
{
    public FNovKontinent()
    {
        InitializeComponent();
        /*gradnika tBKont in groupBox morata imeti lastnost Modifiers
         nastavljeno na Public*/
    }
}
```

PRIPRAVA NAMESTITVENEGA PROGRAMA, TESTIRANJE

Na neki točki v procesu razvoja *Windows* aplikacije, je potrebno projekt pripraviti za testiranje na ciljnem računalniku. Pripraviti ga je potrebno tudi tako, da ga bodo uporabniki lahko namestili na svoj računalnik in ga tudi uporabljali. Testiranje projekta pri uporabniku nam lahko v veliki meri pomaga pri iskanju boljših rešitev in pri odpravljanju težav, do katerih lahko pride pri uporabi aplikacije na drugem računalniku.

V preteklosti je bilo večina projektov nemščenih s pomočjo raznih namestitvenih (***Setup***) programov, ki so bili shranjeni na *CD*-ju ali nekje v omrežni soseščini. V zadnjih letih pa lahko *Windows* projekte namestimo neposredno preko *Web* strežnikov, kar je posebno ugodno pri nadgradnji projektov. Kot primer lahko vzamemo programe za protivirusno zaščito, ki jih je zaradi stalno novih in novih virusov potrebno nadgrajevati zelo pogosto. *Visual Studio 2010* nam za potrebe testiranja, namestitev in nadgradenj ponuja oba načina.

Obstajajo trije načini za razmnoževanje oz. testiranje aplikacij.

- *XCopy*
- *ClickOnce*
- Izdelava *Setup* projekta– a (namestitvenega programa)

Vsak od teh načinov ima svoje prednosti, pa tudi slabosti.

XCopy

Najstarejši način za prenos aplikacije na drug sistem je s pomočjo metode *xcopy* operacijskega sistema. Mapo, v katero je shranjena naša izvršilna datoteka (*.exe*) na sistemu, na katerem smo ga razvili, enostavno prekopiramo na uporabnikov disk. Če so v tej mapi vse potrebne datoteke za našo aplikacijo, se bo projekt pravilno odprl in uporaba aplikacije oziroma njeno testiranje bo možno tudi na tem računalniku. Prednost tega načina je, da ni potrebna nobena dodatna konfiguracija ali pa registracija produkta.

Postopek je naslednji : končna izvršilna verzija projekta, ki smo ga razvili, je na našem sistemu shranjena v podmapi našega projekta in sicer v mapi *…bin\Release*. S pomočjo ukaza za kopiranje datotek ***XCopy*** operacijskega sistema *DOS* celotno mapo prekopiramo na nek medij (npr. CD, ključ, medij v omrežni soseščini …).

Primer ukaza *XCopy*

C:>\ xcopy "C:\Moja aplikacija\bin\release" "E:\Moja aplikacija\" /s*

Seveda lahko kopiranje izvedemo tudi s pomočjo tehnike *Copy – Paste v Windows Explorerju*.

Ta način je seveda najenostavnejši, ima pa veliko pomanjkljivosti. Če namreč v mapi *Release* ni vseh potrebnih datotek, ali pa če ciljni uporabnik nima nameščenega okolja .NET Framework, aplikacije na njegovem računalniku ne bo možno zagnati.

ClickOnce

Pri izdelavi namestitvenega projekta, ki bo na voljo uporabnikom preko spletnega strežnika, ali pa tudi preko klasične metode (npr. instalacijski CD ipd.) nam pomaga poseben čarovnik (*Wizard*), vsebovan v razvojnem okolju *Visual C#*.

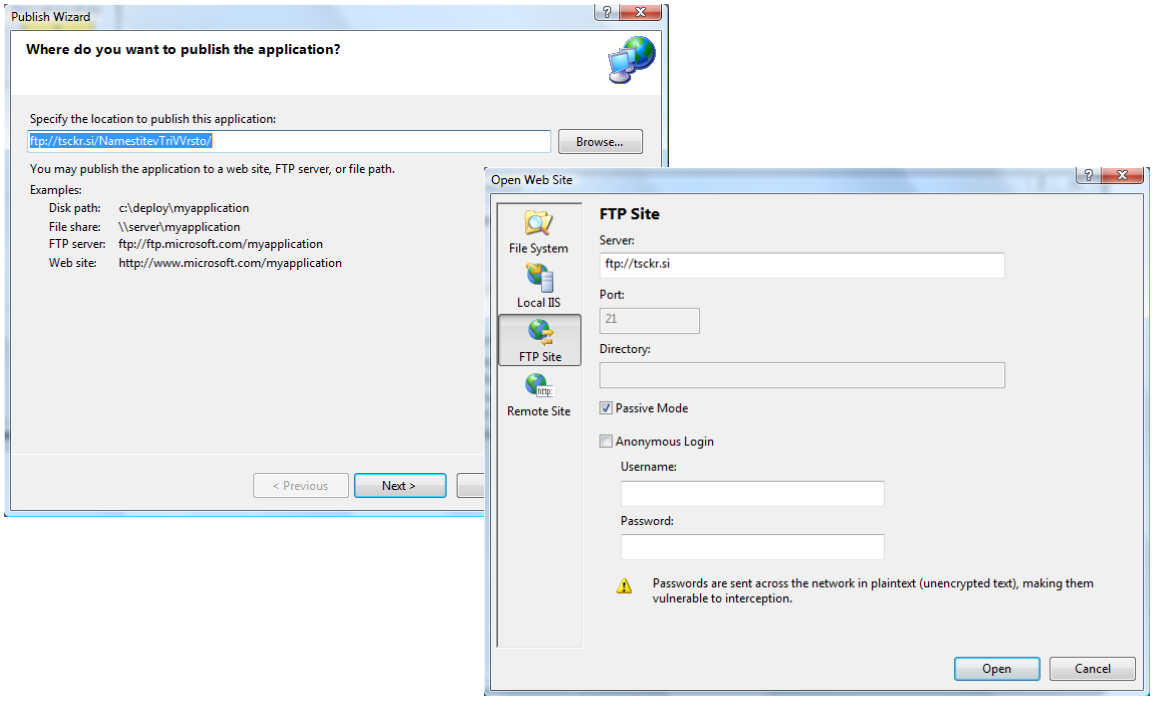
Izberimo projekt, za katerega bomo naredili nov namestitveni projekt in ga postavili na nek strežnik, kjer bo na voljo uporabnikom. Kot alternativo bomo ustvarili še namestitveni program, s pomočjo katerega bo možno naš projekt namestiti pri uporabniku kar preko ključa oz. CD-ja. Izberimo npr. projekt *TriVVrsto*, ki smo ga izdelali v enem od prejšnjih poglavij.

POZOR! V primeru, da bomo namestitvene datoteke ustvarili na nekem strežniku, bo na strežniku v izbrani mapi poleg namestitvenih datotek nastala tudi datoteka Publish.htm, ki omogoča instalacijo preko spleta (npr. s pomočjo *Microsoft Internet Explorer*-ja). Če pa bomo namestitvene datoteke ustvarili na nekem disku, bomo na le-tem dobili le namestitvene datoteke (najpomembnejša med njimi je *Setup.exe*, ki požene namestitev).

Odprimo torej najprej projekt *TriVVrsto*. Z desnim klikom na projekt v oknu *Solution Explorer* poženemo čarovnika za izdelavo namestitvene aplikacije. Odpre se novo okno, v katerem izberemo postavko *Publish*, da se odpre *Publish Wizard*.

V prvem koraku moramo navesti lokacijo, na kateri želimo objaviti (oz. kamor želimo postaviti) naše namestitvene datoteke. Na voljo imamo več možnosti. Namestitveno datoteko, ali pa več

datotek, lahko namreč ustvarimo na strežniku, kjerkoli na disku, na *ftp* strežniku ali pa na spletni strani. Možnosti se odprejo s klikom na gumb *Browse*. V našem primeru bomo namestitveno datoteko ustvarili na *ftp* strežniku *TŠC Kranj* (*ftp://tsckr.si*), seveda pa moramo pred tem v obrazec *Open Web Site* vnesti svoje uporabniško ime in geslo. Ciljno mapo na *ftp* strežniku ustvari namestitveni program sam!



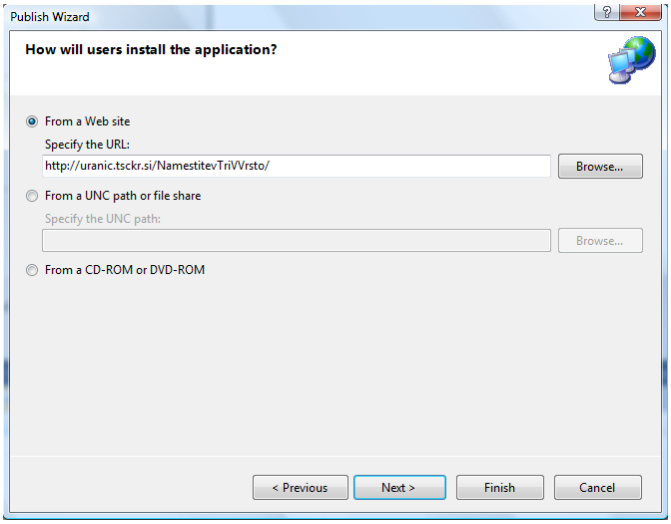
Slika 27: Ustvarjanje namestitvenih datotek na *ftp* strežniku.

Če bi hoteli namestitvene datoteke ustvariti na nekem lokalnem strežniku, bi v oknu *Open Web Site* izbrali *File System* in izbrali namestitveno mapo na nekem strežniku.

V drugem koraku določimo, kako bo uporabnik instaliral aplikacijo. Izbrati moramo eno od treh ponujenih opcij

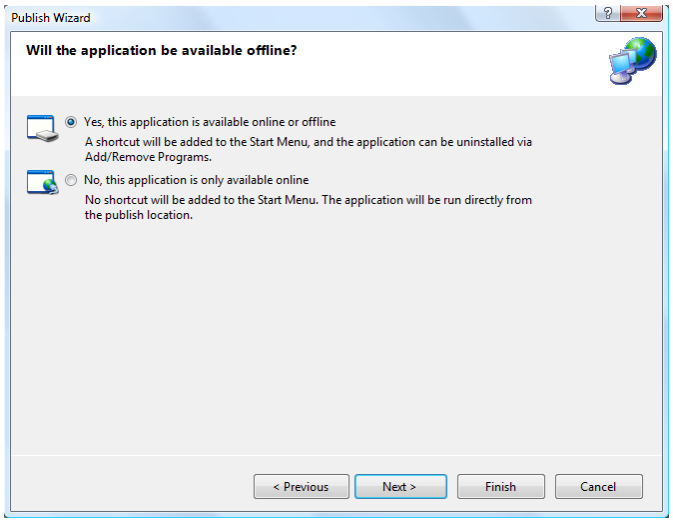
- *From a Web site* - instalacija z neke spletne strani .
- *From a UNC path or file share* – instalacija preko strežnika v *Windows* okolju (mreži).
- *From a CD-ROM or DVD-ROM* –instalacija s *CD*-ja oz *DVD*-ja.

V našem primeru izberimo prvo možnost: potrebno je še zapisati naslov strežnika in mape, v katero se bodo shranile namestitvene datoteke.



Slika 28: Okno v katerem določimo, kako bo uporabnik namestil svojo aplikacijo.

V tretjem koraku določimo, ali bo naša aplikacija dostopna *offline*. Privzeta je vrednost *Yes*, kar za našo trenutno namestitev pomeni, da jo uporabnik lahko namesti brez vnaprejšnje prijave na nek strežnik oz. spletno stran.

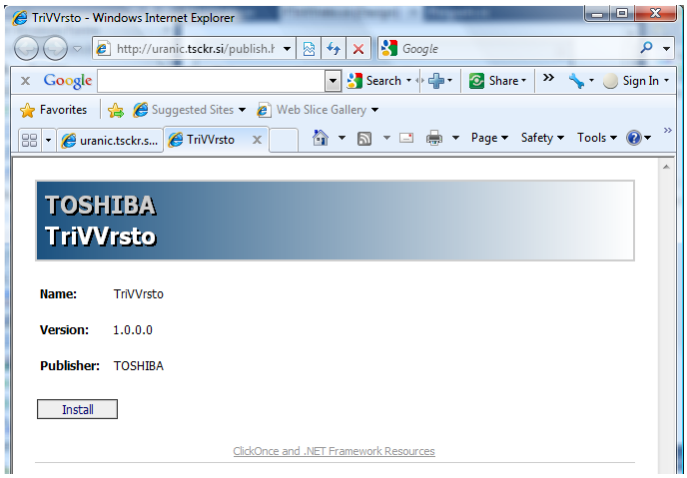


Slika 29: Okno v katerem določimo, ali bo za namestitev potrebna prijava ali ne!

V četrtem koraku imamo možnost potrditve vseh namestitev. S klikom na gumb *Finish* bo čarovnik na strežniku ustvaril ustrezno mapo, v njej pa vse potrebne datoteke za namestitev, hkrati pa še ustrezen certifikat, ki ga lahko uporabljamo za testiranje. Obenem se spremenijo/dodajo nekatere nastavitve (datoteke) v *Solution Explorer*-ju, to datoteke potrebne za zagotovitev prijavljanja, varnosti in razmnoževanja aplikacije. Za kasnejšni pregled in spreminjanje teh nastavitev lahko kadarkoli v *Solution Explorer*-ju izberemo projektno datoteko,

kliknemo desni miškin gumb in izberemo *Properties*. V oknu ki se prikaže izberemo ustrezen zavihek (npr. zavihek o varnosti – *Security*) in opravimo želene spremembe.

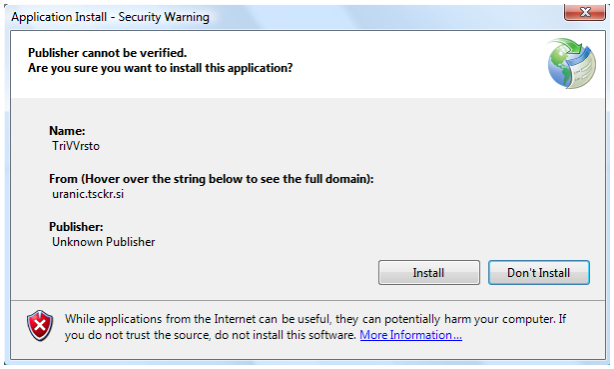
Namestitvene datoteke so sedaj pripravljene. Če smo v prvem koraku izbrali strežnik, *ftp* strežnik oz. spletno stran, je ob ustvarjanju namestitvenih datotek nastala tudi datoteka *Publish.htm*, ki omogoča namestitev preko spleta (npr. s pomočjo *Microsoft Internet Explorerja*), torej *namestitev aplikacije preko spletne strani*. Bodočim uporabnikom naše aplikacije sedaj le pošljemo ustrezen *link* za dostop do namestitve.



Slika 30: Namestitveno okno naše aplikacije.

Namestitev izvedemo tako, da se v internet brskalniku postavimo v mapo na spletnem strežniku in kliknemo datoteko *Publish.htm*, da se odpre okno za namestitev. Prikaže se namestitveno okno, v njem pa glavne lastnosti projekta, ki ga želimo namestiti.

Namestitev zaženemo s klikom na gumb *Install*. Na ekranu se prikaže pogovorno okno za namestitev nove aplikacije.



Slika 31: Varnostno opozorilo pred namestitvijo!

V splošnem pogovorno okno vsebuje tudi varnostno opozorilo, ker pač naša testna aplikacija vsebuje le testni certifikat, ne pa tudi certifikat s pooblastilom. Ne glede na to obvestilo bodo naši uporabniki vedeli, da aplikacija prihaja z mesta, ki mu lahko zaupajo, še posebno kadar gre za namestitev preko lokalnega intraneta. O tem seveda lahko uporabnike prej tudi obvestimo s posebnim sporočilom preko elektronske pošte. Varnostnega opozorila pa se seveda lahko znebimo z namestitvijo veljavnega certifikata.

Po namestitvi bo v meni *Start* dodana bližnjica za zagon aplikacije, možna pa bo tudi odstranitev te aplikacije preko opcije *Dodaj ali odstrani programe* v *Nadzorni plošči (Control Panel)*.

Možna je seveda tudi izdelava aplikacije, ki je dostopna le *online*. V tem primeru se aplikacija na uporabnikovem računalniku požene neposredno s strežnika. V tem primeru ni potrebno ustvarjanje bližnjice v *Start* meniju in obenem tudi ne zmožnost odstranitve aplikacije. Potrebna je le vsakokratna vzpostavitev povezave s strežnikom in *download* aplikacije preko tega strežnika pred njeno uporabo, kar pa lahko traja dalj časa.

Za vsako *ClickOnce* namestitev je možno tudi avtomatsko posodabljanje oz. *update*, če je vzpostavljena povezava z ustreznim strežnikom, na katerem je projekt ustvarjen. Recimo, da na strežniku obstaja novejša verzija aplikacije *TriVVrsto*. Ko uporabnik starta svojo verzijo aplikacije, le ta najprej preveri, ali na strežniku obstaja novejša različica. Če obstaja novejša verzija, se odpre pogovorno okno s sporočilom, da obstaja novejša različica ter nam ponudi *download* te nove verzije. S klikom na gumb *OK* pogovornega okna se bo nova različica namestila in se odslej naprej tudi izvajala na uporabnikovem računalniku (vse do naslednje nove verzije).

Kdaj in v katerem primeru se bo prikazalo okno z obvestilom in nastavitvami za nadgradnje naše aplikacije, pa lahko določimo v pogovornem oknu *Application Updates*. Okno odpremo tako, da v *Solution* Explorer-ju izberemo projekt, kliknemo desni miškin gumb, izberemo *Properties.* V oknu, ki se prikaže, izberemo zavihek *Publish* in kliknemo na gumb *Updates*. Odpre se pogovorno okno za določanje oz. spreminjanje privzetih lastnosti za posodobitve. Če na primer želimo, da se naša aplikacija zažene preden so naložene posodobitve, izberemo prvi radijski gumb. Tako se bo najnovejša verzija aplikacije zagnala šele, ko jo bomo odprli naslednjič. V tem primeru imamo možnost še dodatne nastavitve, saj lahko določimo tudi interval, kako pogosto naj naša aplikacija preverja, ali obstajajo posodobitve. Na vrhu tega sporočilnega okna obstaja tudi *Potrditveni gumb (CheckBox)* s pomočjo katerega lahko izključimo preverjanje obstoja novih verzij oziroma posodobitev.

Na dnu okna je še gumb za določanje minimalne zahtevane verzije programa, ki jo je še možno posodobiti, ter možnost izbire druge lokacije od koder se bo izvedla posodobitev.

Setup (namestitveni) program

Če nam prva dva načina za izdelavo novih verzij programov ne zadoščata, je na voljo še tretja, najbolj kompleksna. Žal pa ta verzija ni na voljo v *Visual C# Express Edition*, ampak le v višjih različicah razvojnega orodja ***Visual C#*** (npr. *Standard Edition*).

DOKUMENTIRANJE

Dokumentiranje razredov

Kot že vemo, lahko splošne komentarje v našo kodo dodajamo tako, da stavke s komentarjem začnemo z znakoma **//** za enovrstični komentar ali pa s parom znakov /* oz */ za večvrstični komentar.

Pri pisanju novih razredov in njihovih članov pa lahko uporabljamo tudi XML dokumentacijo. XML dokumentacija bo naše razrede naredila razumljivejše za ostale razvijalce, ki bodo mogoče nadaljevali z našim projektom, ali pa samo uporabljali razred, ki smo ga napisali. Tudi za nas je takšno pisanje dokumentacije koristno, saj če imamo veliko lastnih razredov, bomo s časoma pozabili njihov namen in možnosti uporabe. S pomočjo *XML* komentarjev pa bomo njihovo uporabo in namen hitro obnovili. Informacija, ki jo bomo napisali kot *XML* dokumentacijo, se bo tako ob uporabi našega razreda in ustrezne kombinacije tipk prikazala na ekranu in sicer v okvirčku pod imenom razreda ali metode.

Čeprav *XML* dokumentacija temelji na *XML* sintaksi, nam ni potrebno podrobno poznavanje *XML*-a da bi ga lahko uporabljali. Vedeti moramo le, da se vrstica z *XML* dokumentacijo prične s tremi znaki /// (tremi *slash*-i), nahajati pa se morajo takoj za razredom oz. njegovim članom, ki ga želimo dokumentirati.

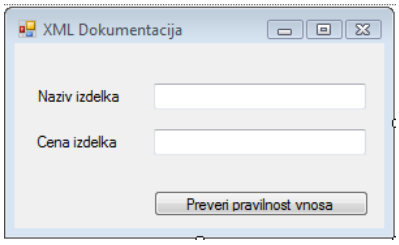
Dokumentacija za razred ali člana razreda lahko vsebuje enega ali več elementov dokumentiranja. Vsak element se prične z oznako za začetek, npr. *<summary>* in se konča z oznako za konec elementa, npr. *</summary>*. Vsebina tega elementa dokumentacije se nahaja med začetno in končno oznako.

XML	Razlaga
<summary>	Zagotavljanje temeljne razlage o nekem razredu, lastnosti, metodi ali drugem elementu.
<value>	Uporablja se za opis oz. razlago vrednosti neke lastnosti.
<returns>	Uporablja se za opis oz. razlago vrnjene vrednosti neke metode.
<param name>="name**"**	Uporablja se za opis oz. razlago parametrov metode.

Tabela 1: Tabela najpogostejših elementov XML dokumentacije.

Zapisano je že bilo, da je najlažji način za ustvarjanje *XML* dokumentacije tak, da v prvi vrstici, ki sledi najavi novega razreda, vtipkamo trojno poševnico oz. *slash* (///). *Visual Studio* nam nato samodejno naredi ogrodje *XML* dokumentacije, vključno s tem, ali je oznaka primerna za člana razreda, ki ga dokumentiramo. Če npr. vtipkamo /// v prazni vrstici pred članom razreda, ki ima parametre in vrača nek rezultat, bo *Visual Studio* tudi avtomatično generiral oznako *<summary>* za to metodo, oznako *<param name =…>* za posamezne parametre te metode in oznako *<return>* za vrednost, ki jo metoda vrača. Nato lahko med oznakami za začetek in konec nekega elementa *XML* dokumentacije pričnemo s tipkanjem opisa posamezne sekcije.

Projekt *XML Dokumentacija* demonstrira primer *XML* dokumentacije razredov. Na obrazcu sta le dve vnosni polji: v prvo naj bi uporabnik vnesel poljuben tekst, ki pa ne sme biti prazen, v drugo polje pa naj bi vnesel neko decimalno število. Po kliku na gumb *Preveri pravilnost vnosa* bomo s pomočjo dveh statičnih metod razreda *Validator* preverili pravilnost uporabnikovega vnosa. Razred *Validator* smo napisali sami in ga opremili z *XML* dokumentacijo.



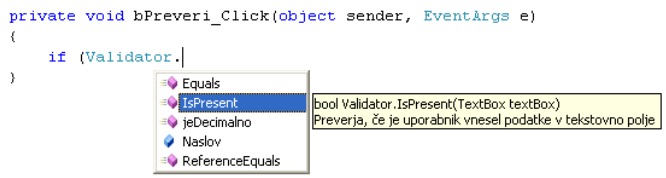
Slika 32: Projekt *XMLDokumentacija*.

```
public partial class FXML : Form
{
    public FXML()
    {
        InitializeComponent();
    }
    /// <summary>
    /// Vsebuje statične metode za kontrolo pravilnosti uporabnikovega vnosa
    /// </summary>
    public class Validator
    {
        public Validator()//konstruktor
        {
        }

        /// <summary>
        /// Besedilo, ki se bo izpisalo NA pogovornem oknu ob napačnem vnosu
        /// </summary>
        public static string Naslov="Napaka pri vnosu!";

        //statična metoda, zato jo lahko kličeno kar s pomočjo razreda
        /// <summary>
        /// Preverja, če je uporabnik vnesel podatke v tekstovno polje
```

```
        /// </summary>
        /// <param name="textBox">Tekstovno polje, ki se preverja</param>
        /// <returns>True (v redu), če je uporabnik vnesel podatke</returns>
        public static bool IsPresent(TextBox textBox)
        {
            if (textBox.Text=="")
            {
                /*Predpostavimo, da je pomen vnosnega polja zapisan v
                  lastnosti Tag*/
                MessageBox.Show("Polje "+textBox.Tag+" ne sme biti
prazno!",Naslov);
                textBox.Focus();
                return false;
            }
            return true;
        }
        //statična metoda, zato jo lahko kličeno kar s pomočjo razreda
        public static bool jeDecimalno(TextBox textBox)
        {
            try
            {
                Convert.ToDecimal(textBox.Text);
                return true;
            }
            catch
            {
                /*Predpostavimo, da je pomen vnosnega polja zapisan v
                  lastnosti Tag*/
                MessageBox.Show(textBox.Tag + " mora biti decimalno
število.", Naslov);
                textBox.Focus();
                return false;
            }
        }
    }

    private void bPreveri_Click(object sender, EventArgs e)
    {
        if (Validator.IsPresent(tBNaziv) && Validator.jeDecimalno(tBCena))
                MessageBox.Show("Vnos pravilen!");
    }
}
```

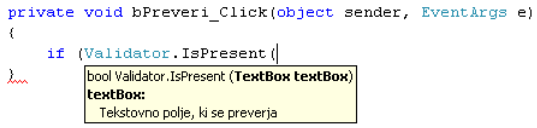
Ko smo *XML* dokumentacijo dodali v naš razred, jo bo *Visual Studio* uporabil npr. tedaj, ko bo uporabnik zapisal ime razreda, ali pa ime metode, pa tudi tedaj, ko se le z miško zapelje prek kode.

V prvem primeru, ko vtipkamo ime razreda *Validator* in za njim še znak pika, *IntelliSense* prikaže seznam članov razreda *Validator*. Med njimi je tudi metoda *IsPresent*, ki smo jo napisali. Ob njej se v okvirčku pojavi okno z opisano dokumentacijo za ta razred, ki smo jo napisali s pomočjo *XML*-a.



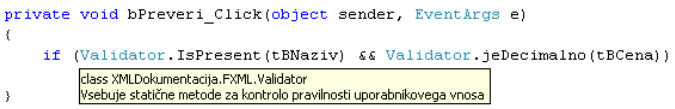
Slika 33: Sistem *IntelliSense* poleg imena metode prikaže tudi okno z našo dokumentacijo.

Ko uporabnik izbere metodo *IsPresent* in za njo zapiše oklepaj, *Visual Studio* v okvirčku pod metodo izpiše razlago za parameter te metode, tako kot smo jo napisali s pomočjo *XML* dokumentacije.



Slika 34: Izpis *XML* dokumentacije za parameter metode.

Informacijo o samem razredu pa lahko uporabnik dobi, če se s kazalnikom miške postavi kamorkoli na besedico *Validator* (ime razreda) v kodi.



Slika 35: Izpis dokumentacije o našem razredu.

LITERATURA IN VIRI

- Uranič S. Praktično programiranje (online). Kranj: TŠC, 2010. (citirano 1. 2. 2011). Dostopno na naslovu: http://uranic.tsckr.si/C%23/Prakticno%20programiranje.pdf
- Sharp, J. *Microsoft Visual C# 2005 Step by Step*. Washington: Microsoft Press, 2005.
- Murach, J., in Murach, M. *Murach's C# 2008*. Mike Murach @ Associates Inc. London, 2008.
- Petric, D. *Spoznavanje osnov programskega jezika C#*, diplomska naloga. Ljubljana: UL FMF, 2008.
- Jerše, G., in Lokar, M. *Programiranje II*, *Objektno programiranje*. Ljubljana: B2, 2008.
- Jerše, G., in Lokar, M. *Programiranje II*, *Rekurzija in datoteke*. Ljubljana: B2, 2008.
- Kerčmar, N. *Prvi koraki v Javi, diplomska naloga*. Ljubljana: UL FMF, 2006.
- Lokar, M. *Osnove programiranja: programiranje – zakaj in vsaj kaj*. Ljubljana: Zavod za šolstvo, 2005.
- Uranič, S. *Microsoft C#.NET*, (online). Kranj: TŠC, 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http://uranic.tsckr.si/C%23/C%23.pdf
- Uranič, S. *Microsoft Visual C#.NET*, (online). Kranj: TŠC, 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http://uranic.tsckr.si/VISUAL%20C%23/VISUAL%20C%23.pdf
- *C# Practical Learning 3.0*, (online). 2008. (citirano 1.2.2011). Dostopno na naslovu: http://www.functionx.com/csharp/
- Lokar, M., et.al. *Projekt UP – Kako poučevati začetni tečaj programskega jezika, sklop interaktivnih gradiv*, (online). 2008. (citirano 1. 1. 2011). Dostopno na naslovu: http://up.fmf.uni-lj.si/
- Lokar, M. *Wiki C#*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu http://penelope.fmf.uni-lj.si/C_sharp
- Coelho, E. *Crystal Clear Icons*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu: http://commons.wikimedia.org/wiki/Crystal_Clear
- Mayo, J. *C# Station Tutorial*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx
- *C# Station*, (online). 2008 (citirano 1.2.2011). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx
- *CSharp Tutorial*, (online). 2008. (citirano 1.12.2011). Dostopno na naslovu: http://www.java2s.com/Tutorial/CSharp/CatalogCSharp.htm
- *FunctionX Inc*, (online). 2008. (citirano 1.2.20111). Dostopno na naslovu: http://www.functionx.com/csharp/
- *SharpDevelop, razvojno okolje za C#*, (online). 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http: // www.icsharpcode.net/OpenSource/SD/.

- WIKI DIRI 06/07, (online). 2008. (citirano 1. 2. 2011). Dostopno na naslovu: http://penelope.fmf.uni-lj.si/diri0607/index.php/Kategorija:Naloge