UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Program: Praktična matematika

DELPHI IN PODATKOVNE BAZE

DIPLOMSKA NALOGA

Ljubljana, november 2006

Gerželj, B. Delphi in podatkovne baze. Diplomska naloga.
Ljubljana, Univerza v Ljubljani, Fakulteta za matematiko in fiziko, 2006

Kazalo vsebine

Kazalo slik

Zahvala

Iskreno se zahvaljujem mag. Matiju Lokar za mentorstvo in pomoč pri pisanju diplomske naloge. Hvala za njegovo potrpežljivo vodenje, za koristne nasvete in izbiro literature.

Iskrena hvala podjetju Marand za obrazložitev nekaterih pojmov.

Hvala prof. Bojani Levinger za lektoriranje diplomske naloge.

Zahvaljujem se vsem domačim in prijateljem, ki so mi v času študija stali ob strani. Hvala za razumevanje in potrpežljivost.

Hvala vsem.

Program diplomske naloge

V diplomski nalogi predstavite, kako s pomočjo programa Microsoft Access pripravimo bazo, v kateri hranimo tudi grafične datoteke in kako to bazo upravljamo s programom, razvitim z orodjem Borland Delphi.

Osnovna literatura:

- M. Cantu, Mastering Borland Delphi 2005, Sybex, 2005
- Gajič, Žarko, Delphi Programming, http://delphi.about.com/library/weekly/aa010101a.htm
- Gajič, Žarko, Pictures inside a database, http://delphi.about.com/library/weekly/aa030601.htm

mentor
mag. Matija Lokar

Povzetek

V diplomski nalogi se bom ukvarjala s podatkovnimi bazami in načinom, kako upravljamo z njimi s programskim jezikom. Kot osnovo za opisovanje dela z bazami bom prikazala razvoj dela aplikacije, ki bi lahko služila za upravljanje s podatki o umetninah neke galerije ali zbiralca umetniških del.
V ta namen bomo spoznali programa Microsoft Access in Borland Delphi 2005. Pogledali bomo, kaj so relacijske podatkovne baze in kako jih lahko uporabimo. S pomočjo orodja Microsoft Access bomo najprej ustvarili bazo (PodatkiUmetnine.mdb). Nekaj podatkov bomo vnesli ročno, znotraj programa Access, ostale pa bomo vnašali s pomočjo programa, razvitega v programskem jeziku Delphi. V Accessu bomo ustvarili tri tabele (Avtorji, Umetnine in Tipi Umetnin), s katerimi bomo kasneje operirali. Med seboj bomo te tabele povezali in s tem ustvarili relacije. Spoznali bomo tudi orodje za razvoj programov Delphi in med različnimi možnimi uporabljali programski jezik Delphi. Spoznali bomo gradnike za delo z bazami (Data Access, Data Controls, ADO gradniki). Povezali se bomo z bazo, ki smo jo naredili v Microsoft Accessu. V jeziku Delphi bomo razvili program, ki nam bo omogočal vstavljanje podatkov v podatkovno bazo. Zato bomo spoznali tudi gradnike za vstavljanje podatkov. S pomočjo programa bomo v podatkovni bazi tudi spreminjali podatke, jih urejali in brisali. V podatkovni bazi bomo hranili tudi grafične podatke. Zato bomo morali spoznati način, kako hranimo tovrstne podatke in kako upravljamo z njimi.
Razviti program bo le osnova za spoznavanje določenih pojmov v povezavi z razvojem programov v določenem programskem jeziku, ki delajo z bazami podatkov in ni mišljen kot neka popolnoma uporabna aplikacija, saj mu do tega manjka še veliko.

Math. Subj. Class. (2000): 68P20, 68N15, 68N19, 68U35.

Computing Review Class System (1998): H.2.8, H.2.7, D.3.2., D.3.3, J.5.

Ključne besede: podatkovna baza, relacija, primarni ključ, tuji ključ, gradnik, Microsoft Access, Borland Delphi, gradnja uporabniškega vmesnika, programski dostop do baze.

Key words: database, relation, primary key, secundary key, component, Microsoft Access, Borland Delphi, user interface, programming access to a database.

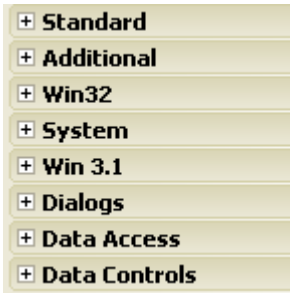
Uvod

O problemu

V diplomski nalogi se bom ukvarjala s podatkovnimi bazami in načinom, kako upravljamo z njimi s programskim jezikom Delphi. Kot osnovo za opisovanje dela z bazami bom prikazala razvoj dela aplikacije, ki bi lahko služila za upravljanje s podatki o umetninah neke galerije ali zbiralca umetniških del. Tovrstni program bi na primer lahko uporabljala določena galerija, muzej ali druga ustanova, pa tudi posameznik, ki ima doma kaj več kot dve sliki in en kipec. Poleg podatkov o samih umetninah bo vsebovala še nekatere podatke o umetnikih, avtorjih slik, kipcev, fotografij, ki so v bazi, ter osnovne podatke o tehnikah, v katerih so izdelane umetnine.
Na začetku bom povedala nekaj splošnih podatkov o programih Borland Delphi in Microsoft Access, za kaj bom uporabila ta dva programa in katere funkcije teh dveh programov bom uporabila. Naprej pa bom razložila še nekatere pojme, ki jih bom uporabljala pri pisanju diplomske naloge, kot so podatkovna baza, relacije med podatki, … V nadaljevanju bom opisovala izdelovanje podatkovne baze v Accessu, razložila, čemu jo bomo rabili, opisala izdelovanje programa v Delphiju in kako povezati ta dva programa. V zaključku bom še na kratko opisala, kako razviti program deluje in kje vse bi ga lahko uporabili.

Uporabljena orodja

Ko pišemo aplikacijo v Delphiju, ki črpa in shranjuje podatke v podatkovni bazi, potrebujemo orodja za dostop do podatkov v bazi in seveda bazo samo. Vedeti moramo, do kakšnih podatkov in na kakšen način bomo do njih dostopali.
Kot orodji bomo uporabili programa Access in Delphi. Najprej bomo naredili bazo v MS Accessu, kjer bomo lahko podatke vpisovali ročno. Nato bomo v Delphiju napisali program, ki bo omogočal vpisovanje podatkov v bazo preko obrazca.
Čeprav bi o programu Delphi lahko napisali celo knjigo, ga bomo poskusili opisati v nekaj stavkih. Delphi je predmetno usmerjeno programsko okolje za programiranje v Oknih (v operacijskem sistemu Windows). Delphi vsebuje vse pripomočke, ki so potrebni pri izdelavi programa. Pri pisanju programske kode uporabljamo urejevalnik. Priloženo ima tudi orodje za izdelavo baz (Database Desktop) in za osnovno oskrbovanje (vpis, posodabljanje, brisanje podatkov) baz s podatki, program za izdelavo zaslonske pomoči in celo program za risanje sličic, s katerimi opremljamo gumbe in druge programske elemente.
Prav tako se bomo v diplomski nalogi pogosto srečali s pojmom, ki je v Delphiju poimenovan »components«. Za ta izraz srečamo različne prevode, kot so sestavniki, gradniki, komponente in drugo. Pri podjetju Marand, ki je slovenski zastopnik podjetja Borland, ki izdeluje program Delphi in verjetno podjetje, ki se v Sloveniji največ ukvarja s programom Delphi, so predlagali, naj uporabljamo izraz gradnik. Gradniki, ki so na voljo v Delphiju, so združeni v skupine, ki so na voljo v zavihkih. Najpogosteje bomo uporabili prvih nekaj zavihkov:



Slika1. Zavihki, ki jih bomo največ uporabljali

Predstavimo na kratko še program Microsoft Access. Microsoft Access je program za delo z relacijskimi podatkovnimi bazami. Kot vsi drugi izdelki iz sistema Microsoft Office je Microsoft Office Access 2003 zasnovan tako, da omogoča hitro zbiranje, prikazovanje in uporabo podatkov.

Gerželj, B. Delphi in podatkovne baze. Diplomska naloga.
Ljubljana, Univerza v Ljubljani, Fakulteta za matematiko in fiziko, 2006

Ker se bom s to diplomsko nalogo tudi sama učila programiranja, kjer delam s podatkovnimi bazami, bom skozi nalogo predpostavila, da tudi bralec šele spoznava osnove dela v okolju Delphi. Zato bom skozi celotno nalogo podrobno opisovala posamezne korake o načinu dela z gradniki, kako določati lastnosti gradnika, … Osnove dela v Delphiju naj bi bralec vseeno že poznal (kako odpreti datoteko, preklapljati med pogledom na kodo in pogledom na izgled aplikacije, kako pognati program, ...).

Podatkovne baze

Podatkovne baze so računalniško zapisane množice podatkov, ki so organizirane tako, da zagotavljajo sprotne, celovite in nasploh kakovostne informacije o sistemu podatkov. Podatkovna baza je sklop podatkov, medsebojnih sklicevanj na podatke in sistem za razvrščanje in urejanje podatkov v bazi. Podatkovna baza je torej tudi klasična knjižnica, a v vsakdanjem pogovoru baza pomeni računalniški sistem za hrambo podatkov.

Podatkovne baze so se pojavile zaradi potrebe po hitrem dostopu do informacij, saj hramba podatkov iz preteklosti omogoča premišljeno odločanje o prihodnosti. S takšnim namenom so se gradile knjižnice in tak namen vodi današnja podjetja, da ustvarjajo baze podatkov o povpraševanju, strankah, vremenskih razmerah in sploh vsem, kar ima možnost vpliva na bodoče poslovanje.

V različnih publikacijah je podatkovna baza poimenovana tudi podatkovna zbirka, podatkovna datoteka, zbirka podatkov, baza podatkov in podobno. Mi bomo skozi diplomsko nalogo uporabljali enotno poimenovanje, to je podatkovna baza.

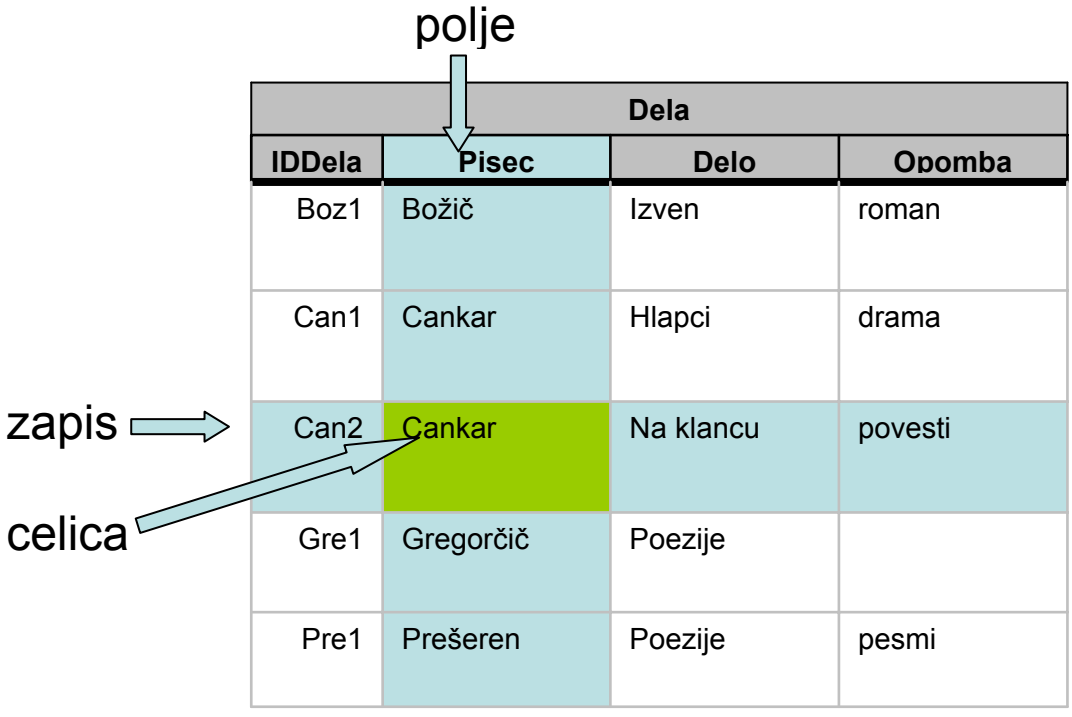
Mnogi uporabniki enačijo podatkovno bazo s tabelo podatkov, toda temu ni ravno tako. Podatkovne baze so običajno sestavljene iz večjega števila tabel, ki so večinoma med seboj povezane. In prav ta povezava tabel nam predstavlja "pravo" podatkovno bazo. V našem primeru bodo podatkovno bazo sestavljale tri tabele. V prvi tabeli bomo imeli podatke o avtorjih umetnin, v drugi tabeli podatke o umetninah, v tretji pa podatke o tipih umetnin. Vse tri tabele skupaj bodo sestavljale podatkovno bazo.

Relacijske baze

Danes se običajno uporabljajo tako imenovane relacijske baze. Osnovna ideja relacijskega modela baze je, da vsebuje množico med seboj povezanih tabel ali na kratko množico relacij. Baza lahko vsebuje veliko različnih tabel.
Relacijske podatkovne baze se uporabljajo praktično povsod, kjer je treba v organizirani obliki hraniti večje število podatkov. Relacijo običajno predstavimo v tabelarični obliki. Vsak element tabele (vrstica) hrani podatke o enem zapisu, npr. o enem študentu, eni knjigi, … Vsak stolpec pa označuje določeno lastnost teh zapisov (npr. vpisna številka, število strani, …)
V uporabi je več sistemov za delo z relacijskimi podatkovnimi bazami. Nekateri so prosta programska oprema (najbolj znana sta MySQL in PostgreSQL), drugi spet plačljiva (Oracle, Microsoft SQL, ...). Primerni so tako za manjše kot tudi za velike baze, na primer za bazo študentov na neki fakulteti ali pa za hranjenje vseh podatkov o davčnih zavezancih v Republiki Sloveniji.
Različni sistemi za delo s podatkovnimi bazami se med seboj sicer razlikujejo v podrobnostih in zmogljivosti, vendar pa danes več ali manj vse upravljamo s podobnimi programskimi jeziki. Ti izvirajo iz standardiziranega jezika za delo z bazami podatkov. To je *SQL* (Standard Query Language). Ko se naučimo uporabljati en sistem za delo z bazami, je prehod na drugega lahek.

Osnovni pojmi

V tem razdelku bomo na kratko predstavili pojme oziroma izraze, ki jih bomo uporabljali pri delu s podatkovnimi bazami. Pojme bomo razložili na enostavnem primeru dveh tabel, ki predstavljata književna dela in avtorje književnih del.
Na primeru prve tabele, ki predstavlja književna dela, pojasnimo osnovne pojme, kot so podatkovna tabela, polje, zapis, celica in tip podatkovne baze.
Tabelo ***Dela*** sestavljajo štiri polja: *IDDela*, *Pisec*, *Delo* in *Opomba*. O polju govorimo tudi pri posameznem zapisu. Tako je na primer niz »Na klancu« vrednost polja »Delo« tretjega zapisa.
Zapis pomeni vrstico tabele. V našem primeru je to posamezno književno delo. Zapisi se v tabeli razlikujejo po vrednosti polja *IDDela*. Polje *IDDela* je sestavljeno iz prvih treh črk priimka pisca in zaporedno številko vpisanega dela posameznega pisca. Presečišče stolpca in vrstice imenujemo celica.
Nekateri tipi podatkovnih baz imajo vgrajeno tudi samodejno preverjanje podatkov, da v številsko polje ne moremo vnesti črk, iz tabele zbrisati podatka, na katerega se sklicuje druga tabela in podobno.



Dela			
IDDela	**Pisec**	**Delo**	**Opomba**
Boz1	Božič	Izven	roman
Can1	Cankar	Hlapci	drama
Can2	Cankar	Na klancu	povesti
Gre1	Gregorčič	Poezije	
Pre1	Prešeren	Poezije	pesmi

Slika2. Primer tabele, kjer razložimo pojme

Primarni ključ vsebuje eno ali več polj (stolpcev), katerih vrednost ali vrednosti enolično določajo vsak zapis v tabeli. Primarni ključ mora biti obvezno določen, torej ne sme biti na primer vrednosti Null. Primarni ključ uporabljamo tudi za ustvarjanje relacije med tabelo in tujimi ključi v drugih tabelah. V primeru zgoraj bi na primer kot primarni ključ lahko izbrali stolpec *IDDela*.

Relacije med podatki

Če želimo informacije iz več tabel združiti v poizvedbi, obrazcu, poročilu ali strani za dostop do podatkov, definiramo relacije med tabelami. Relacije opisujejo odnose med elementi ene množice ali pa odnose med elementi različnih množic.
Relacija deluje tako, da povezuje zapise v dveh ali več tabelah, ki imajo ujemajoče se podatke v poljih s ključem. Običajno so ti ujemajoči se podatki v poljih z enakim imenom v obeh tabelah, ni pa nujno, da se ustrezna stolpca v tabelah imenujeta enako. V večini primerov sta ti ujemajoči se polji primarni ključ v prvi tabeli in tuji ključ v drugi tabeli.
Tuji ključ vsebuje eno ali več polj v tabeli (stolpcev), ki se sklicujejo na polje ali polja primarnega ključa v drugi tabeli. Tuji ključ kaže na relacije med tabelami.
Poznamo več vrst relacij. Tako ločimo relacijo »ena proti mnogo«, relacijo »mnogo proti mnogo« in relacijo »ena proti ena« .
Relacija »ena proti mnogo« je najbolj običajna vrsta relacije. V relaciji »ena proti mnogo« ima lahko zapis iz tabele ***A*** veliko ujemajočih se zapisov v tabeli ***B***, zapis iz tabele ***B*** pa samo en ujemajoči se zapis v tabeli ***A***. Tako je v našem zgledu pisatelj lahko avtor več knjig (je večkrat naveden kot pisec v tabeli ***Dela***), vsaka knjiga pa ima le enega avtorja (navedenega v stolpcu Pisec v tabeli ***Dela)***, saj smo naš podatkovni model zelo poenostavili in knjige nimajo po več avtorjev.
Pri relaciji »mnogo proti mnogo« ima lahko zapis iz tabele ***A*** veliko ujemajočih se zapisov v tabeli ***B***, pa tudi zapis iz tabele ***B*** ima lahko veliko ujemajočih se zapisov v tabeli ***A***. To vrsto relacije je v sistemih za delo z relacijskimi bazami mogoče izvesti samo tako, da vpeljemo tretjo tabelo (z imenom stična tabela), katere primarni ključ je sestavljen iz dveh polj - tujih ključev iz tabel ***A*** in ***B***. Relacijo »mnogo proti mnogo« torej izvedemo kot dve relaciji »ena proti mnogo« s pomočjo tretje tabele.
Pri relaciji »ena proti ena« ima lahko vsak zapis iz tabele ***A*** samo en ujemajoči se zapis v tabeli ***B*** in tudi vsak zapis iz tabele ***B*** ima lahko samo en ujemajoči se zapis v tabeli ***A.*** Ta vrsta relacije ni pogosta, saj bi takrat tabeli ***A*** in ***B*** lahko združili v eno tabelo. Relacijo »ena proti ena« uporabimo, če želimo tabelo z mnogo polji razdeliti na več tabel, če želimo iz varnostnih razlogov izolirati del tabele ali če želimo shraniti informacije, ki se nanašajo samo na podmnožico glavne tabele.
Vrsta relacije, ki jo Microsoft Access ustvari, je odvisna od tega, kako so določena polja v relaciji. Relacija »ena proti mnogo« se ustvari, če je samo eno od polj v relaciji primarni ključ ali ima enoličen indeks. Enolični indeks je indeks, ki ga definiramo tako, da nastavimo lastnost Indexed polja na Da (brez dvojnikov). Enolični indeks v indeksiranem polju ne dovoljuje dvojnikov – torej da bi imela dva zapisa enako vrednost polja. Če določimo, da je polje primarni ključ, bo to samodejno opremljeno z enoličnim indeksom.
Relacija »ena proti ena« se ustvari, če sta obe polji v relaciji primarna ključa in če imata enolična indeksa. Relacijo »mnogo proti mnogo« izvedemo kot dve relaciji, »ena proti mnogo« s tretjo tabelo, katere primarni ključ je sestavljen iz dveh polj - tujih ključev iz ostalih dveh tabel.
Sedaj si pojem relacija oglejmo še na primeru. Prejšnji tabeli dodamo še tabelo ***Pisci***.

Tabela ***Pisci***:

Pisci : Tabela

	Pisec	Obdobje	Rojen	Umrl	Lirika	Epika	Dramatika
▶	Božič	Po vojni	1932		☐	☑	☑
	Cankar	Moderna	1876	1918	☐	☑	☑
	Gregorčič	Pozna romantika	1844	1906	☑	☐	☐
	Prešeren	Romantika	1800	1849	☑	☐	☐
*					☐	☐	☐

Slika3. Tabela Pisci

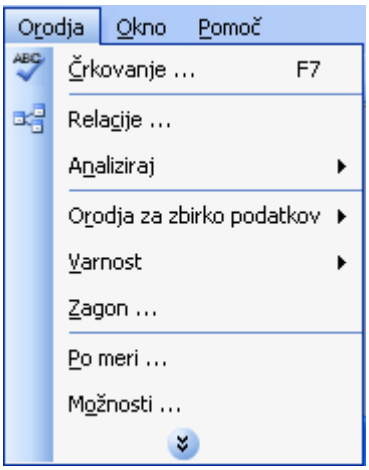
Že od prej imamo tabelo ***Dela***:

Dela : Tabela

IDDela	Pisec	Delo	Opomba
Boz1	Božič	Izven	roman
Can1	Cankar	Hlapci	drama
Can2	Cankar	Na klancu	povesti
Gre1	Gregorčič	Poezije	
Pre1	Prešeren	Poezije	pesmi

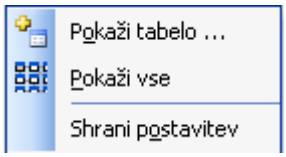
Slika4. Tabela Dela

Podatki o književnikih se nahajajo v tabeli »***Pisci***«, podatki o njihovih delih pa v tabeli »***Dela***«. Ustvarimo relacijo med njima. V menijski vrstici izberemo ***Orodja*** in kliknemo na izbiro ***Relacije***.



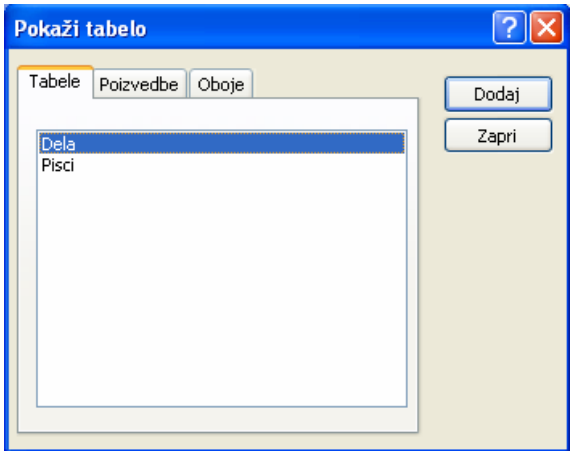
Slika5. Odpremo Relacije

Odpre se prazno okno, na katerega kliknemo z desnim gumbom miške. V meniju izberemo ***Pokaži tabelo***.



Slika6. Na formo nanesemo vse tabele

Odpre se okno z ustvarjenimi tabelami.

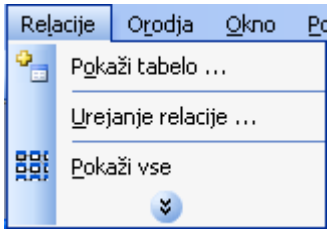


Slika7. Označimo obe tabeli

Izberemo obe tabeli in v oknu se prikažeta obe s polji, ki ju sestavljata. Ustvariti moramo še relacije med njima.

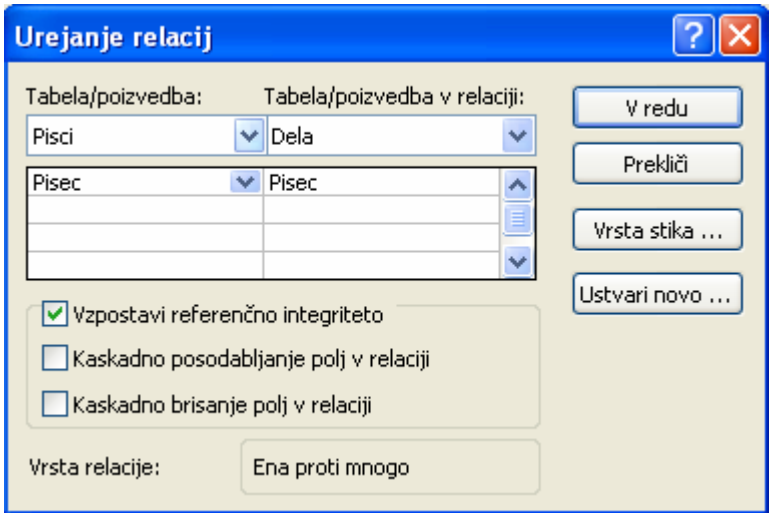
Slika8. Obe tabeli se pojavita na formi

Relacije med tabelama ustvarimo tako, da v menijski vrstici kliknemo na meni ***Relacije*** in nato ***Urejanje relacije***.



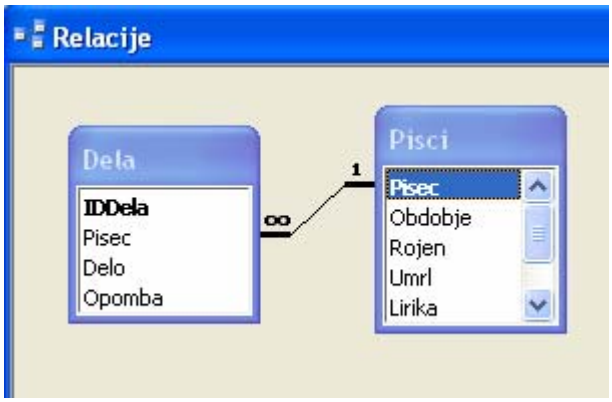
Slika9. Ustvarimo relacijo med tabelama

Odpre se okno, kjer izberemo obe tabeli in ustvarimo relacijo med njima. V izbirniku ***Tabela/poizvedba*** izberemo tabelo ***Pisci.*** Kot drugo tabelo (***Tabela/poizvedba v relaciji***) si izberemo ***Dela***. V obeh tabelah z izbiro polja *Pisec* vzpostavimo relacijo ***Ena proti mnogo***.



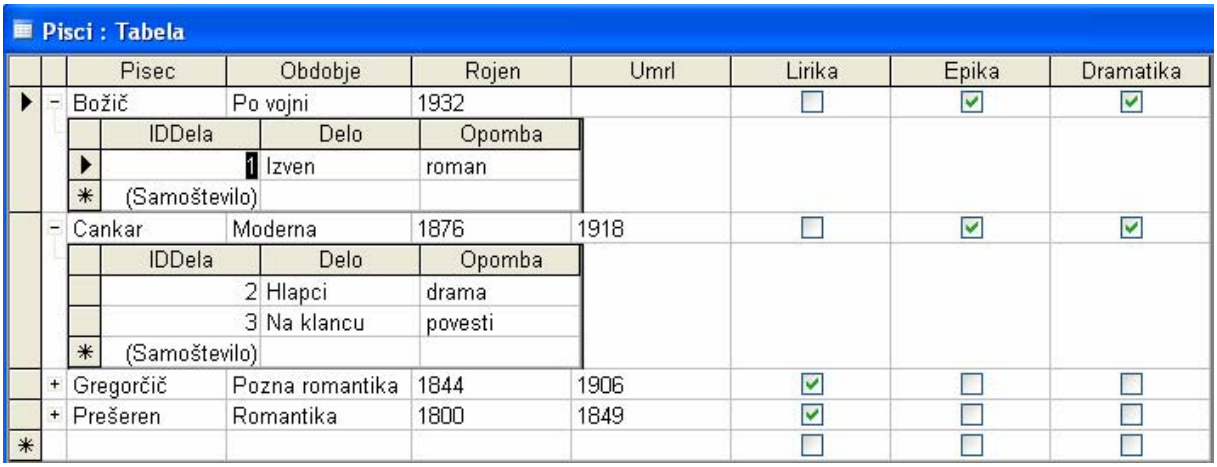
Slika10. Označimo, kateri dve polji bomo povezali

Ob kliku na gumb ***V redu*** se ustvari naslednja relacija:



Slika11. Tabeli sta v relaciji

Enolični ID, kot je polje *Pisec* iz tabele, enolično določa posamezen zapis v tej tabeli. To pomeni, da v tabeli »***Pisci***« ne moremo imeti dveh zapisov, ki bi imela v polju *Pisec* enako vrednost (ne moremo imeti dveh piscev z istim imenom). V tabeli »***Dela***« polje *Pisec* ni enolični identifikator zapisa. Namreč posamezen pisec je lahko avtor več del. Z vzpostavitvijo relacije med tema dvema poljema lahko Microsoft Access ugotovi povezane zapise iz obeh tabel. To pomeni, da lahko Microsoft Access ve, kako je potrebno povezati te zapise iz obeh tabel med seboj. V tabeli, kjer je to polje primarni ključ, so tako vidni zapisi iz obeh tabel.
Sedaj, ko imamo tabeli povezani med seboj, nam Microsoft Access omogoča, da jasno vidimo vsa dela posameznega književnika. Oglejmo si, kako je videti sedaj tabela »***Pisci***«.



Pisci : Tabela

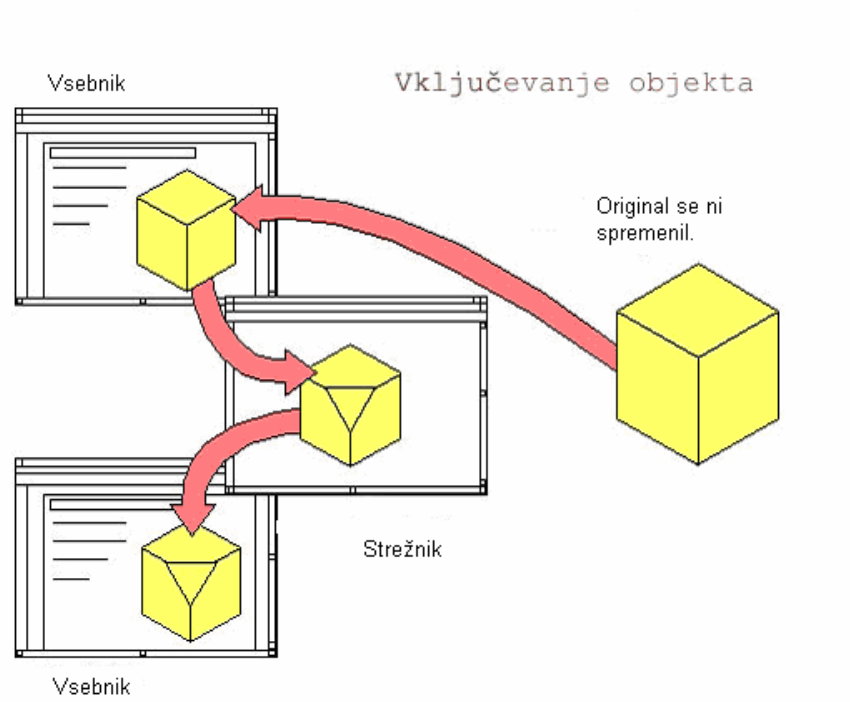
	Pisec	Obdobje	Rojen	Umrl	Lirika	Epika	Dramatika
▶ −	Božič	Po vojni	1932		☐	☑	☑
	IDDela	Delo	Opomba				
	▶ 1	Izven	roman				
	✳ (Samoštevilo)						
−	Cankar	Moderna	1876	1918	☐	☑	☑
	IDDela	Delo	Opomba				
	2	Hlapci	drama				
	3	Na klancu	povesti				
	✳ (Samoštevilo)						
+	Gregorčič	Pozna romantika	1844	1906	☑	☐	☐
+	Prešeren	Romantika	1800	1849	☑	☐	☐
✳					☐	☐	☐

Slika12. Povezana tabela

Kadar ustvarimo relacijo med tabelami, ni nujno, da imajo povezana polja enaka imena. Povezana polja pa morajo biti istega podatkovnega tipa, razen če je primarni ključ polje tipa Samoštevilo. Samoštevilo je podatkovni tip v Microsoft Accessovi zbirki podatkov, ki samodejno shrani enolično število za vsak zapis, kot je dodan v tabelo. Ta števila, torej podatki tipa Samoštevilo, se lahko ujemajo tudi s podatki tipa Število, a mora biti izpolnjen tudi dodatni pogoj. Polje tipa Samoštevilo se ujema s poljem tipa Število le, če je lastnost VelikostPolja obeh polj enaka. Polji tipa SamoŠtevilo in Število tako lahko na primer primerjamo, če je lastnost VelikostPolja obeh polj Dolgo celo število. Prav tako velja, da če sta obe povezani polji tipa Število, morata imeti isto nastavitev lastnosti VelikostPolja.

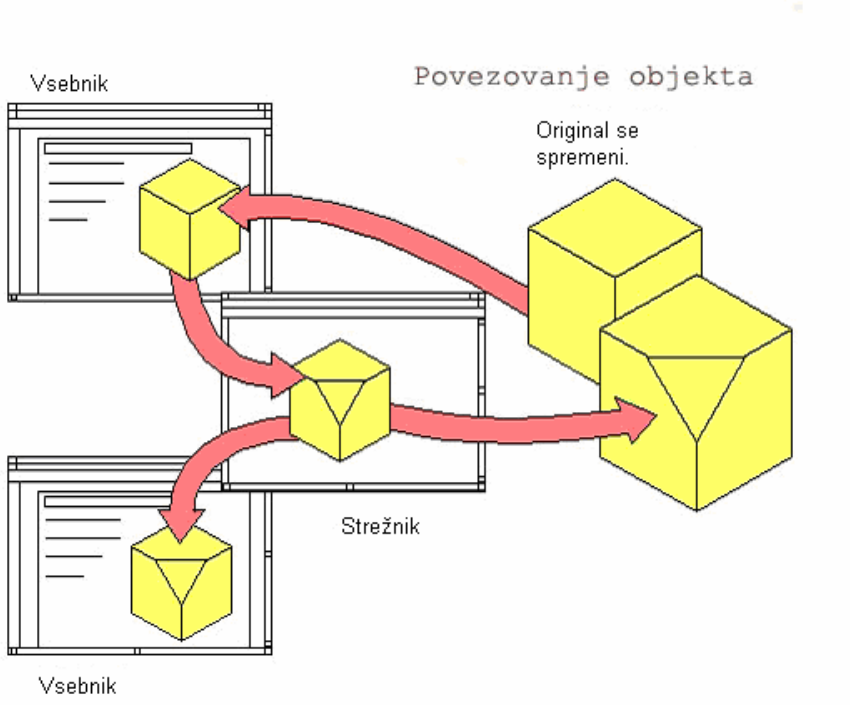
OLE – Object Linking and Embedding

OLE – povezovanje in vključevanje objektov – nam omogoča, da v datoteke vstavljamo objekte, kot so grafika, video posnetki, … Ti objekti so na datotekah, ki so lahko preglednice, običajne tekstovne datoteke z besedili, elektronske prosojnice, …. S pomočjo principa OLE lahko oblikujemo povezavo od baze do grafične datoteke, ki vsebuje sliko. Brez tega postopka (OLE) bi morali vedno, kadar bi želeli v bazo vstaviti sliko, ki smo jo morda spremenili, zagnati grafično aplikacijo in vsako sliko posebej prenesti v bazo s funkcijama kopiraj in prilepi. OLE objekte je mogoče vstavljati, prikazovati, kopirati, lepiti, urediti in spremeniti znotraj same baze. Povezavo do OLE objektov je vedno možno spremeniti in posodobiti.
Oglejmo si razliko med povezovanjem in vključevanjem objektov.



Slika13. Vključevanje objekta

Če je objekt vključen v dokument, dokument vsebuje njegovo kopijo. Poleg samega objekta imamo v dokument (vsebnik) vključeno tudi informacijo, v kateri aplikaciji je objekt nastal (vir objekta – strežnik za objekt). Samega objekta, vključenega v dokument, ne moremo urejati v samem dokumentu (kar verjetno tudi ne bi bilo možno, saj znotraj aplikacije nimamo ustreznih orodij). Če želimo vključeni objekt popraviti (urediti, posodobiti), dvakrat kliknemo nanj, ali pa izberemo primeren urejevalnik, ko je objekt poudarjen. S tem se zažene strežniška aplikacija, kjer objekt uredimo. Če vključen dokument spremenimo, se originalni objekt (tisti, ki smo ga prvotno vključili), ni spremenil.



Slika14. Povezovanje objekta

Če objekt vstavimo kot povezavo, dokument vsebuje le kazalec na prvotno datoteko (prvotni objekt). Če spremenimo povezani objekt, spremenimo tudi prvotnega. Velja pa tudi obratno. Če v originalni, strežniški aplikaciji, kadarkoli posodobimo originalni objekt, je posodobljen tudi objekt v vsebniku, torej v dokumentu, ki vsebuje vključeni objekt.

Uporabili bomo način povezovanja objekta. Objekta ne bomo vstavili v polje, ampak ga bomo vnesli le kot povezavo na datoteko, kjer so slike shranjene.

Načrtovanje podatkovne baze

Preden začnemo podatke hraniti v podatkovni bazi, jo moramo seveda sestaviti. Pri tem opredelimo, katere podatke bomo hranili v bazi, kakšne vrste bodo ti podatki, kako bodo strukturirani in podobno.
Izdelali bomo podatkovno bazo, ki jo bomo uporabljali skozi celoten projekt. Postopek bomo izpeljali od samega začetka, od snovanja baze pa vse do njene dejanske izdelave v Accessu. Potek same izdelave podatkovne baze je opisan v razdelku Izdelava podatkovne baze.
Načrt podatkovne baze je odvisen od njene predvidene uporabe. Samo načrtovanje podatkovne baze je lahko dokaj zapleten postopek, ki zahteva veliko znanja in temeljite priprave. Na srečo naš zastavljeni problem ni tak, da bi zahteval zelo zapleteno bazo. Kljub temu pa je dovolj kompleksen, da si bomo lahko ogledali bistvene pojme pri delu z bazami podatkov.
Kot smo povedali že v opredelitvi problema v razdelku O problemu, bomo v podatkovni bazi hranili podatke o različnih umetninah. V bazi bodo tri tabele. Naredili bomo tabele ***Umetnine***, ***Avtorji*** in ***Tipi Umetnin***. V tabeli ***Umetnine*** bomo hranili podatke o umetninah. Med drugimi podatki bomo tukaj shranili tudi podobo umetnine, kar nam bo omogočalo, da bomo razložili, kako lahko v bazi hranimo in s programom Delphi prikazujemo grafične datoteke. V tabeli ***Avtorji*** bomo hranili podatke o avtorju in njegove osebne podatke, v tabeli ***Tipi Umetnin*** pa kakšen tip umetnine imamo: ali je to slika, fotografija, kipec, freska …
V tabeli ***Umetnine*** bomo imeli naslednja polja:

- *IDUmetnine*, ki vsebuje podatek tipa Samoštevilo. To pomeni, da se števila vpisujejo samodejno po vrsti od 1 naprej. Polje IDUmetnine je primarni ključ te tabele.
- Polje *IDAvtorja* kot podatek vsebuje besedilo, ki je sestavljeno iz prve črke imena in prve črke priimka. Če imamo dva umetnika z istimi začetnicami imena in priimka, dodamo še zaporedno številko umetnika z istimi začetnicami.
- Polje *Naslov* je tipa besedilo, kjer lahko vpišemo kakršenkoli podatek. V tem polju hranimo podatek o naslovu umetnine.
- Polje *Opis* je tudi tipa besedilo. V njem hranimo podatke o tem, kaj slika, kipec, fotografija, ... prikazuje oziroma kaj ponazarja.
- V polje *Tip*, tipa besedilo, vpišemo tip umetnine, ki je bodisi kipec, bodisi slika, ….
- Polje *VelikostUmetnine* nam omogoča, da vnesemo podatke o velikosti umetnine. Ker bomo v bazi hranili zelo različne umetnine (slike, kipe, …), smo se odločili, da bo tudi to polje tipa besedilo, kar nam bo dejansko omogočalo vnos različnih mer.
- Polje *LetnicaNastanka* je tipa besedilo, ki nam pove letnico, kdaj je umetnina nastala. To polje je sicer tipa besedilo, a bomo s programskimi kontrolami dosegli, da bo vanj lahko vpisana le smiselna letnica – torej, da bodo podatki oblike 1834 in ne npr. XXXX.
- Zadnje polje *SlikaUmetnine* je tipa OLE predmet. Razlago, kaj OLE predmeti so, smo si ogledali v prejšnjem razdelku. V to polje bomo shranjevali slike posameznih umetnin.

V tabeli ***Avtorji*** bomo imeli naslednja polja:

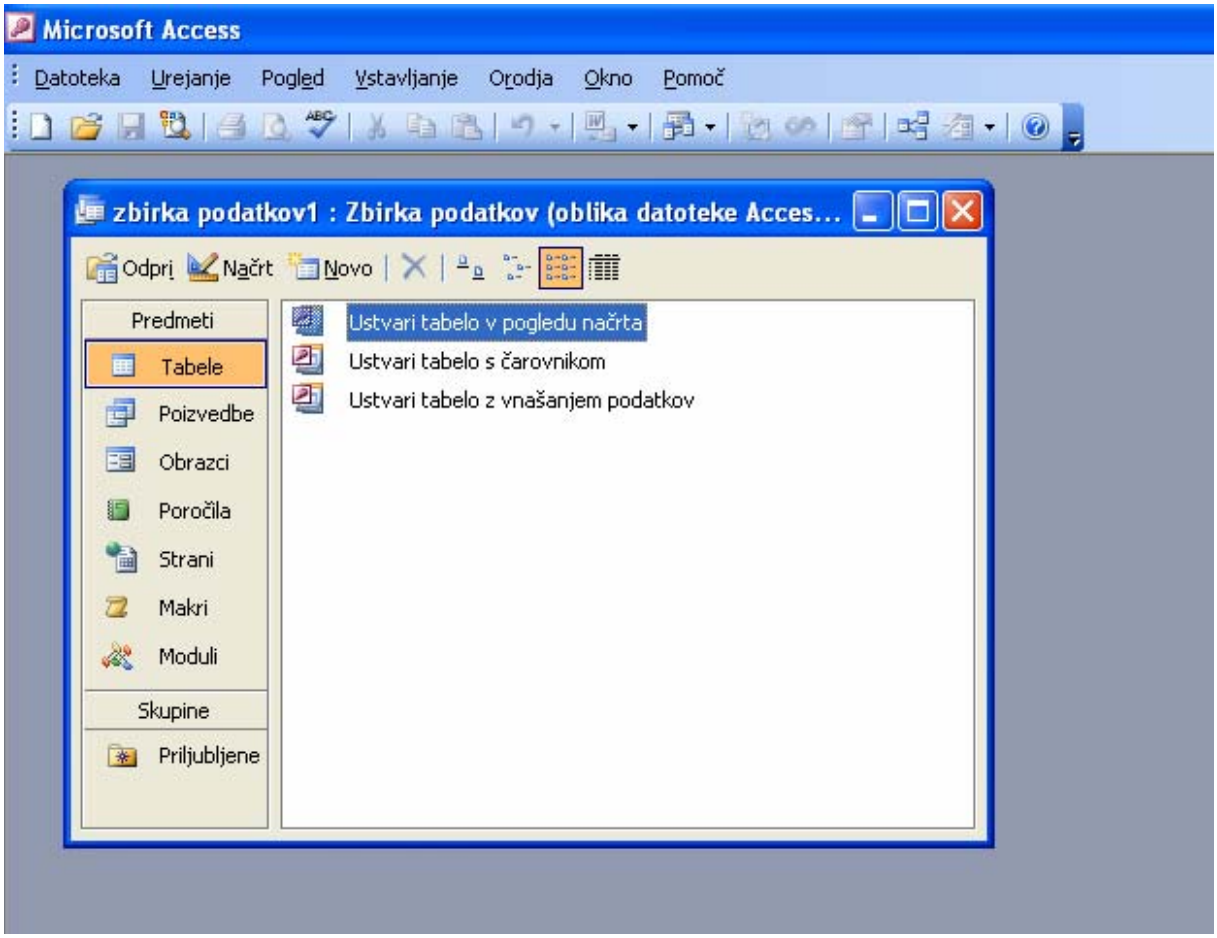
- *IDAvtorja*, ki ima za podatkovni tip besedilo in je primarni ključ tabele.
- Naslednji polji sta *Ime Avtorja* in *Priimek Avtorja*, ki imata za podatkovni tip prav tako besedilo.
- Zadnje polje je *DatumRojstva* umetnika, ki ima za podatkovni tip podatek v obliki datuma.

V tabeli ***Tipi Umetnin*** imamo polje *Tip*, ki ima za podatkovni tip besedilo. To polje je označeno tudi kot primarni ključ. Označuje nam tip umetnine, ki je lahko kip, slika, …

Izdelava podatkovne baze

Sedaj, ko vemo, kakšno bazo želimo imeti, jo izdelajmo. Zato seveda potrebujemo ustrezno orodje.
Program Delphi ni namenjen izdelavi podatkovnih baz. V ta namen obstajajo posebni programi. Ker pa pogosto potrebujemo možnost, da na hitro sestavimo kako enostavno bazo podatkov, okolje Delphi vsebuje orodje Database Desktop. Tega lahko zaženemo iz Delphija (v meniju ***Tools*** z ukazom ***Database Desktop***) ali pa samostojno. Omogoča izdelavo enostavnih podatkovnih baz, poleg tega

pa tudi vsa opravila s podatki nasploh: vpis, posodabljanje in brisanje. Glavna omejitev orodja je, da je v bazi lahko le ena tabela – ne omogoča nam torej vzpostavitve relacij. Prav iz tega razloga smo se odločili, da bomo bazo naredili s pomočjo Accessa.
Podatkovno bazo bomo torej naredili v Accessu. Iz analize, ki smo jo predstavili prej, vemo, da bodo našo bazo sestavljale tri tabele: ***Umetnine***, ***Avtorji*** in ***Tipi Umetnin***.
Zaženemo Access in izberemo ***Datoteka/Nova***. Bazo poimenujmo **PodatkiUmetnine.mdb.**
Sedaj moramo dodati tabele, ki sestavljajo bazo. Pri tem imamo tri možnosti:



Slika15. Ustvarimo tabele

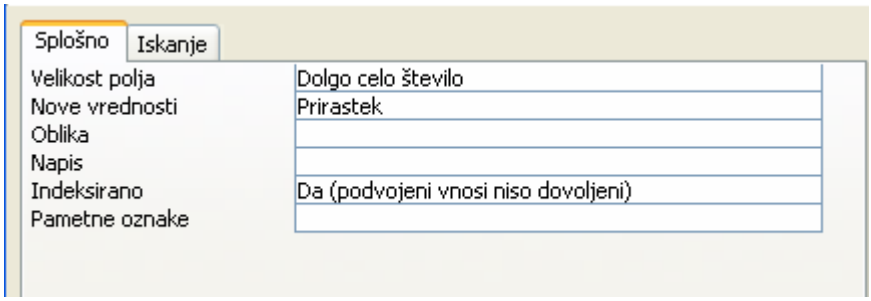
- Ustvarimo tabelo z navajanjem imen polj, podatkovnih tipov in lastnosti polj.
  Najprej navedemo imena stolpcev v posamezni tabeli, ki jih bomo uporabljali. Poleg stolpcev navedemo še tip polja, ali je to besedilo, slika, število, datum,…
- Ustvarimo tabelo s pomočjo čarovnika. Pri tem izbiramo iz seznama pogostih poslovnih in osebnih tabel. Odpre se okno s tabelami, ki so že vnaprej pripravljene. Izberemo tabele, ki že imajo narejene stolpce in tipi podatkov so že določeni.
- Ustvarimo tabelo s vnašanjem podatkov.
  Odpre se prazna tabela, kjer so stolpci poimenovani kot Polje1, Polje2,… Vanjo vnašamo podatke po želji.

Odločili smo se, da bomo tabele oblikovali z navajanjem polj, podatkovnih tipov in lastnosti polj. Ta možnost nam je najbolj ustrezala, ker lahko poljubno izbiramo med posameznimi polji in lahko polja poimenujemo po lastni želji.

Takole zgleda načrt tabele ***Umetnine***:

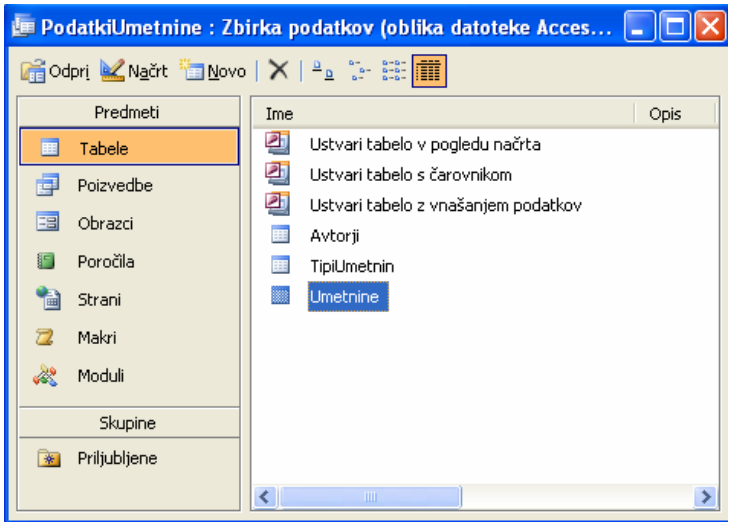
Slika16. Poljem v tabeli nastavimo podatkovne tipe

V tem delu okna določimo tip posameznega polja, v spodnjem delu okna pa imamo lastnosti posameznega polja, ki jih lahko spreminjamo. Lastnosti polja *ID Umetnine* so naslednja:



Slika17. Polju lahko nastavimo tudi lastnosti

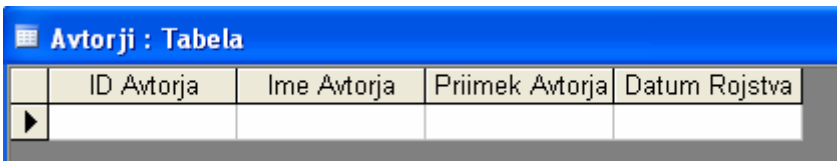
V tabeli ***Umetnine*** opazimo pred poljem *ID Umetnine* ikono ključa, ki ponazarja primarni ključ. Vsaka tabela naj bi imela natanko en primarni ključ. V tabeli lastnosti imamo možnost še nastavljanja številnih drugih lastnosti, kot so kontrola pravilnosti podatkov, vnosne maske in podobno. Ker pa bomo z bazo upravljali programsko, bomo za to tako ali tako poskrbeli s pomočjo programa, ki ga bomo razvili v Delphiju. Zato dodatnih nastavitev ne bomo vnašali.
Na enak način oblikujemo še preostali dve tabeli ***Avtorji*** in ***Tipi Umetnin***. Tabela ***Avtorji*** vsebuje polja *ID Avtorja*, *Ime Avtorja*, *Priimek Avtorja* in *Datum Rojstva*. Tabela ***TipiUmetnin*** pa vsebuje polje *Tip Umetnine*.



Slika18. Ustvarili smo vse tri tabele

Ročni vnos podatkov

Narejene so vse tri tabele. Baza je zaenkrat prazna, ker jo bomo kasneje polnili s pomočjo programa. Ker pa nas zanima, če smo pravilno nastavili lastnosti podatkov, bomo prvih nekaj podatkov vnesli ročno.
Tabele, ki sestavljajo bazo, so sedaj pripravljene. Vnesimo v te tabele nekaj podatkov. Najprej si pripravimo tabelo avtorjev. Odpremo tabelo ***Avtorji*** in vidimo, da imamo prazno tabelo.



Slika19. Prazna tabela Avtorji

Za preizkus bomo vnesli nekaj podatkov ročno. V prvo polje *ID Avtorja*, ki je tipa besedilo, vnesemo niz RS, ki ponazarja prvo črko imena in priimka, v drugo polje *Ime Avtorja*, ki je tudi tipa besedilo, vnesemo ime umetnika - Rudi, v tretje polje *Priimek Avtorja*, vnesemo priimek umetnika - Skočir, v zadnje polje *Datum Rojstva*, ki je tipa datum, pa vnesemo datum rojstva umetnika - 12. 3. 1959.

Slika20. Ročni vnos podatkov v tabeli Avtorji

Prav tako vnesemo še nekaj podatkov:

MB	Michelangelo	Buonarroti	6.3.1475
PP	Pablo	Picasso	20.10.1881

Tabela ***Avtorji*** po vstavljanju teh treh umetnikov zgleda takole:

Slika21. Vnesli smo nekaj podatkov, s katerimi bomo operirali

Prav tako vnesemo nekaj podatkov v tabelo ***Umetnine***. V polje *ID Umetnine*, ki je tipa Samoštevilo, se avtomatično vnese število 1, v polje *ID Avtorja*, tipa besedilo, vnesemo niz RS, v polje *Naslov*, ki je tipa besedilo, vnesemo naslov umetnine – Pikova dama, v polje *Opis*, tipa besedilo, vnesemo opis slike – Karta, na kateri je prikaz pikove dame , v polje *Tip*, tipa besedilo, vnesemo tip umetnine - slika, v polje *Velikost Umetnine*, tipa besedilo, vnesemo dimenzije umetnine, 30x60, v polje *Letnica Nastanka*, tipa besedilo, vnesemo letnico nastanka umetnine – 1982, v zadnje polje *Slika Umetnine*, tipa OLE predmet, pa vnesemo sliko umetnine. Postopek vstavljanja slike bomo opisali v nadaljevanju.
Tudi v to tabelo vnesemo še nekaj podatkov ročno.

Umetnine : Tabela

ID Umetnine	ID Avtorja	Naslov	Opis	Tip	Velikost Umetnine	Letnica Nastanka	Slika Umetnine
1	RS	Pikova dama	Karta, na kateri je prikaz pikove dame	slika	30x60	1982	
2	RS	Rudarji	Prikaz idrijskih rudarjev	slika	90x90	2000	
3	PP	La Vie	Trpljenje ljudi	slika	50x60	1890	
4	MB	Mojzes	Kip Mojzesa	kipec		1515	

Slika22. Tabela Umetnine, po ročnem vnosu podatkov

V tabeli ***Tipi Umetnin*** imamo eno samo polje. V polje *Tip Umetnine*, tipa besedilo, vnesemo tip umetnine. Kot *Tip Umetnine* vnesemo podatka kipec in slika.

Tipi Umetnin : Tabela

Tip Umetnine
kipec
slika

Slika23. Tabela Tipi Umetnin

Vstavljanje slik v bazo

Sedaj si poglejmo, kako lahko v bazo dodamo sliko (grafično datoteko). V MS Accessu sliko lahko shranimo v polje tipa OLE Object. Shrani se kot tako imenovani BLOB (**B**inary **L**arge **OB**ject). BLOB je torej način hranjenja velike količine binarnih podatkov. Ta tip podatkov uporabljamo za hranjenje oblikovanega besedila, velikih tekstovnih datotek, video posnetkov, glasbenih datotek in slik. Podatkovne baze hranijo slike v polju tipa BLOB. Če kliknemo z miško na nov zapis, se prikaže nova slika v obrazcu. Program Access nam dovoli le vstavljanje slik, ki so v določenih grafičnih formatih. Na srečo so med njimi tisti najbolj pogosti, formati JPEG, GIF in BMP.

Poglejmo, kako bomo v našo bazo vstavili neko sliko umetnine. Zaženemo MS Access in odpremo bazo **PodatkiUmetnine.mdb**. Odpremo tabelo ***Umetnine*** in kliknemo na polje *Slika Umetnine* v tisti vrstici, kjer je umetnina, katere sliko želimo dodati.

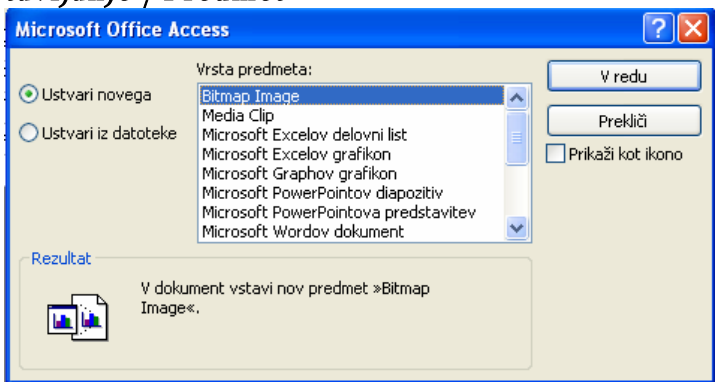
Umetnine : Tabela

ID Umetnine	ID Avtorja	Naslov	Opis	Tip	Velikost Umetnine	Letnica Nastanka	Slika Umetnine
1	RS	Pikova dama	Karta na kateri je prikaz pikove dame	slika	30x60	1982	
2	RS	Rudarji	Prikaz idrijskih rudarjev	slika	90x90	2000	
3	PP	La Vie	Trpljenje	slika	50x60	1890	
4	MB	Mojzes	Kip Mojzesa	kipec		1515	
(Samoštevilo)							

Slika24. V tabelo Umetnine bomo vnesli slike Umetnin

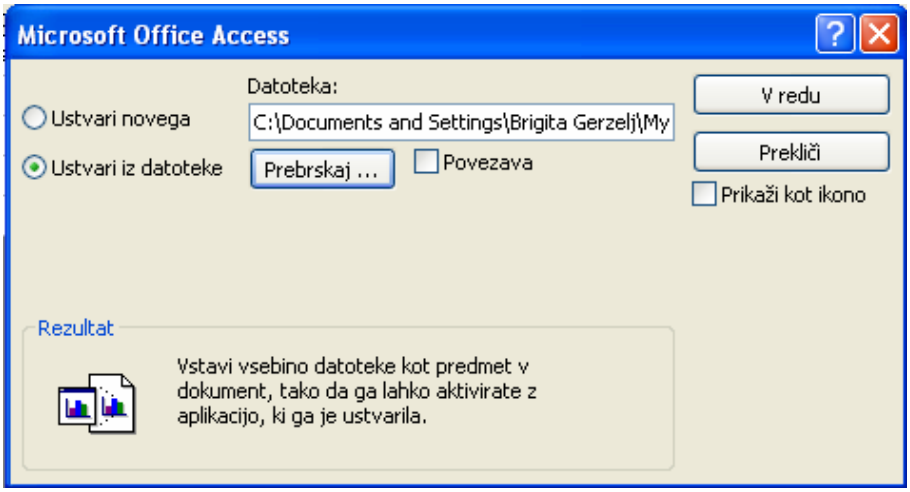
Nato

1. Izberemo ***Vstavljanje / Predmet***



Slika25. Poiskati želimo ustrezno sliko

2. Izberemo ***Ustvari iz datoteke***



Slika26. Na disku poiščemo sliko

3. Kliknemo na gumb ***Prebrskaj*** in na disku poiščemo ustrezno sliko. Na koncu kliknemo gumb ***V redu***, da potrdimo nastavitve.

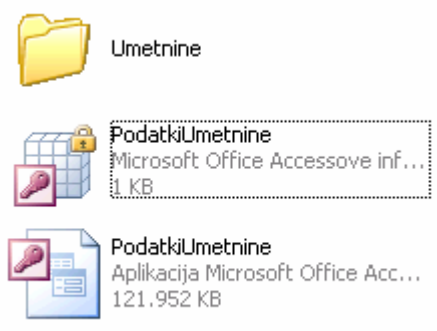
Po vstavljanju slike je tabela videti kot

Umetnine : Tabela

	ID Umetnine	ID Avtorja	Naslov	Opis	Tip	Velikost Umetnine	Letnica Nastanka	Slika Umetnine
▶	1	RS	Pikova dama	Karta na kateri je prikaz pikove dame	slika	30x60	1982	Package
	2	RS	Rudarji	Prikaz idrijskih rudarjev	slika	90x90	2000	
	3	PP	La Vie	Trpljenje	slika	50x60	1890	
	4	MB	Mojzes	Kip Mojzesa	kipec		1515	
*	(Samoštevilo)							

Slika27. Na mestu, kjer bi morala biti slika, se izpiše Package

Vidimo, da v ustrezni celici, kjer naj bi bila slika, te ni, ampak le piše Package. Slike v pogledu na tabelo ne vidimo, prikaže se nam šele takrat, ko dvakrat kliknemo na celico v polju *Slika Umetnine*. Odpre se nam v urejevalniku slik, ki je na nivoju operacijskega sistema določen za prikaz tovrstnih slik, kot na primer v programu Microsoft Photo Editor ali v programu Paint.
Po istem postopku vstavimo še preostale tri slike.
Tako, sedaj imamo vse potrebne podatke. V tabeli imamo shranjene podatke, v zadnjem polju *Slika Umetnine* pa hranimo pot, kjer imamo shranjene slike umetnin. Če bi želeli bazo podatkov prenesti na drug računalnik, bi morali prenesti datoteke: PodatkiUmetnine.mdb, PodatkiUmetnine.ldb in imenik Umetnine, kjer imamo shranjene slike umetnin.

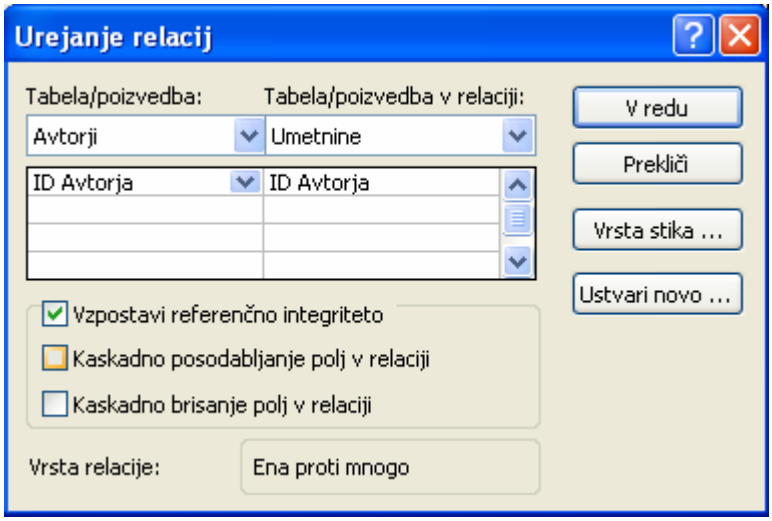


Slika28. Potrebne datoteke za odprtje baze

Sedaj z bazo lahko delamo – spreminjamo podatke, vstavljamo nove, brišemo, ... Hočemo pa, da bi vse te operacije nadzirali s pomočjo programa, ki nam bo omogočal večjo fleksibilnost, kot jo nudi neposredna uporaba programa Access.

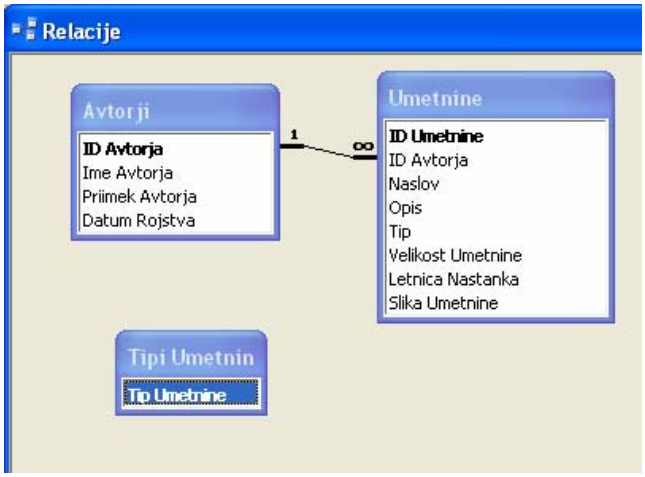
Izvajanje relacij v Accessu

Zaradi lažjega dostopa do podatkov smo se odločili, da po oblikovanju tabele tudi povežemo med seboj. S tem nam bo uspelo, da lahko med pregledovanjem tabele ***Umetnine*** dobimo podatke o avtorju in o tipih umetnin kljub temu, da se ne nahajamo v tabeli ***Avtorji in TipiUmetnin.*** To bo možno, ker bodo vse tri tabele med seboj v relaciji. V Accessu bomo odprli okno ***Relacije***, nanj nanesli vse tri tabele in jih povezali med seboj.
V našem primeru bomo uporabili relacije »ena proti mnogo«. V relaciji »ena proti mnogo« ima lahko zapis iz tabele ***Umetnine*** veliko ujemajočih se zapisov v tabeli ***Avtorji***, zapis iz tabele ***Avtorji*** pa ima lahko samo en ujemajoči se zapis v tabeli ***Umetnine***. To pomeni, da je avtor v prvi tabeli ***Umetnine*** lahko avtor večih umetnin, vsaka umetnina pa je delo le enega avtorja.
Prvi dve tabeli bomo povezali preko polj z enakima imenoma, drugo in tretjo tabelo pa z različnima imenoma polj. V obeh primerih so polja istega tipa. Med seboj lahko povežemo samo stolpce z istimi tipi podatkov. Povežemo jih lahko ročno – to pomeni, da z miškinim gumbom kliknemo na ustrezno polje in ga povlečemo do ustreznega polja druge tabele. Tako povežemo *ID Avtorja* iz tabele ***Umetnine*** z *ID Avtorja* v tabeli ***Avtorji***. Lahko pa polji povežemo tudi iz menijske vrstice, kjer izberemo ***Urejanje relacij***. Odpre se okno, kjer lahko izberemo polje iz leve tabele in polje iz desne tabele.



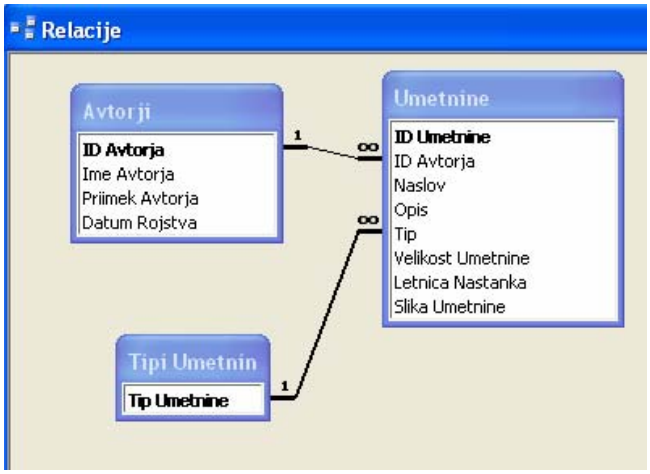
Slika29. Tabele povežemo v relacije

Ob kliku na gumb ***V redu*** nam program poveže obe polji. Ko povežemo prvi dve tabeli, je slika naslednja:



Slika30. Razlika je vidna tudi na formi

Povezati moramo še tretjo tabelo ***TipiUmetnin***. Po povezavi tabel med seboj so relacije naslednje:



Slika31. Vse tri tabele v relaciji

Na primeru iz diplomske naloge je tabela ***Avtorji*** videti takole:

Avtorji : Tabela

ID Avtorja	Ime Avtorja	Priimek Avtorja	Datum Rojstva
MB	Michelangelo	Buonarroti	6.3.1475

ID Umetnine	Naslov	Opis	Tip	Velikost Umetnine	Letnica Nastanka	Slika Umetnine
4	Mojzes	Kip Mojzesa	kipec		1515	Package
(Samoštevilo)						

ID Avtorja	Ime Avtorja	Priimek Avtorja	Datum Rojstva
PP	Pablo	Piccasso	20.10.1881

ID Umetnine	Naslov	Opis	Tip	Velikost Umetnine	Letnica Nastanka	Slika Umetnine
3	La Vie	Trpljenje ljudi	slika	50x60	1890	Package
(Samoštevilo)						

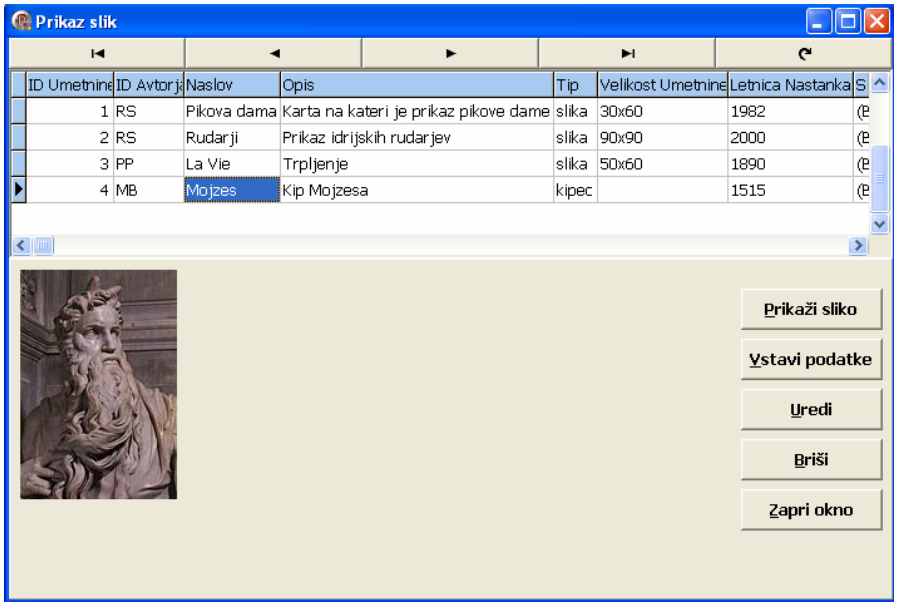
ID Avtorja	Ime Avtorja	Priimek Avtorja	Datum Rojstva
RS	Rudi	Skočir	12.3.1959

ID Umetnine	Naslov	Opis	Tip	Velikost Umetnine	Letnica Nastanka	Slika Umetnine
1	Pikova dama	Karta na kateri je prikaz pikove	slika	30x60	1982	Package
2	Rudarji	Prikaz idrijskih rudarjev	slika	90x90	2000	Package
(Samoštevilo)						

Slika32. Tabela Avtorji po povezavi tabel med seboj

Uporaba orodja Delphi

Podatkovno bazo smo naredili. Sedaj bomo naredili še program v Delphiju, ki nam bo te podatke obdeloval. Del programa nam bo podatke spreminjal, brisal, vstavljal nove podatke in prikazoval sliko umetnine.
V glavnem oknu ***PrikazSlike*** bomo podatke prikazovali. Imeli bomo tabelo, kjer se bodo videli podatki o posamezni umetnini. Ob kliku na gumb bomo sliko umetnine lahko tudi videli. Za vstavljanje podatkov se bo odprlo novo okno ***VstavljanjePodatkov***, ki nam bo vse to omogočalo. Sam izgled glavnega dela programa bo naslednji:

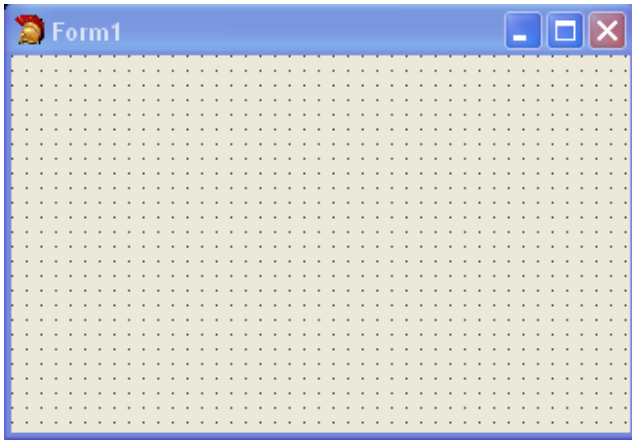


Slika33. Predvideni izgled programa v Delphiju

Oglejmo si, kako tak program naredimo. Najprej zaženemo Delphi. V Delphiju lahko razvijamo različne tipe aplikacij. Pri tem lahko uporabljamo različne programske jezike (C#, Delphi) in razvojne knjižnice (.NET, Win32). Osnovni tipi aplikacij so:

- ASP.NET Web Application – C# Builder
- Windows Forms Application – C# Builder
- Control Library – C# Builder
- ASP.NET Web Application – Delphi for .NET
- VCL Forms Application – Delphi for .NET
- Windows Forms Application – Delphi for .NET
- VCL Forms Application – Delphi for Win32
- Package - Delphi for Win32
- Form - Delphi for Win32
- Unit - Delphi for Win32

Izmed različnih tipov aplikacij smo izbrali ***VCL Forms Application - Delphi for Win32***. Po odprtju aplikacije se prikaže prazna forma, kamor lahko vstavljamo gradnike, ki sestavljajo vizualni del programa in urejevalnik programske kode, kamor zapišemo potrebno kodo, ki se izvaja po zagonu programa. Omenjena forma bo ob zagonu prikazana kot ustrezno okno. `Forma` je preprosta pravokotna ploskev in je osnovni Delphijev gradnik. Nanj pri izdelavi programa polagamo razne vidne in nevidne gradnike, kot so gumbi, vnosna polja, napisi, drsniki, …Vsak Delphijev program ima vsaj eno formo, v praksi pa ponavadi tudi več. Prazna forma je videti takole:



Slika34. Izgled forme

Delphi je zasnovan tako, da samodejno zapiše vso programsko kodo, ki jo zmore zapisati. Ob odprtju Delphi ponudi prazno formo z imenom ***Form1*** in njej ustrezen programski zapis v enoti ***Unit1***. Delphi ima za vse gradnike privzeta imena, ki pa jih lahko sami spreminjamo. Pri določanju imen se pogosto držimo pravila, da s prvimi tremi ali dvema znakoma označimo, za kaj je uporabljeno to ime – za gumb, tabelo, oznako, okno, …..
Prazno formo smo prikazali že zgoraj. Ustrezni program v programskem jeziku Delphi, ki nam ob zagonu tako prazno formo ustvari in ga okolje ustvari avtomatično, je naslednji:

Unit1

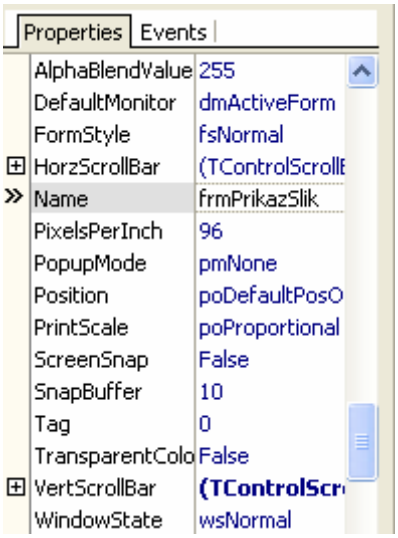
```
1  unit Unit1;
2
3  interface
4
5  uses
6    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7    Dialogs;
8
9  type
10   TForm1 = class(TForm)
11   private
12     { Private declarations }
13   public
14     { Public declarations }
15   end;
16
17 var
18   Form1: TForm1;
19
20 implementation
21
22 {$R *.dfm}
23
24 end.
25
```

Slika35. Urejevalnik kode

Tipi vseh Delphijevih gradnikov so zapisani v knjižnici Delphijevih gradnikov, ki se izvirno imenuje Visual Component Library ali na kratko VCL. Vse knjižnice v Delphiju navedemo v urejevalniku kode pod ključno besedo **uses**.
Naš program zaenkrat sestavlja eno okno. Tega bomo uporabili za prikaz podatkov, ki so shranjeni v bazi, njihovo vstavljanje in popravljanje, ter grafični prikaz umetnine ob kliku na gumb ***Prikaži Sliko***.

Kot smo omenili že prej, Delphi sam avtomatično poimenuje osnovne gradnike in pripadajoče metode. Ker pa nam kasneje, ko bomo imeli opraviti z več gradniki, imena kot so ***Form1*** in podobna ne bodo pomenila kaj dosti, jih je smiselno preimenovati. Njihovo ime določimo v pregledovalniku lastnosti, z lastnostjo `Name`.
Tako za formo v pregledovalniku lastnosti nastavimo lastnosti:

```
Name = frmPrikazSlik
Caption = Prikaz slik
```



Slika36. Nastavitev lastnosti Name

Druga lastnost (`Caption`) je napis, ki se prikaže v zgornji vrstici okna.

Slika37. Nastavitev lastnosti Caption

Na koncu shranimo datoteko pod imenom **PrikazSlike.pas.** Spremembe so vidne prav tako na formi, kot v pregledovalniku lastnosti.

Vse spremembe, ki jih naredimo s pregledovalnikom lastnosti, se zapišejo v ustrezne datoteke dfm, ki so vključene v program z ukazom {$R *.dfm}. Več o tem bomo povedali kasneje.

Slika38. Izgled forme po nastavitvi lastnosti Name

PrikazSlike

```
1  unit PrikazSlike;
2
3  interface
4
5  uses
6    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7    Dialogs;
8
9  type
10   TfrmPrikazSlik = class(TForm)
11   private
12     { Private declarations }
13   public
14     { Public declarations }
15   end;
16
17 var
18   frmPrikazSlik: TfrmPrikazSlik;
19
20 implementation
21
22 {$R *.dfm}
23
24 end.
```

Slika39. Urejevalnik kode po nastavitvi lastnosti

Kodo pišemo v programski modul (izvirno unit). Predstavlja po določenih pravilih zapisano zbirko konstant, podatkovnih tipov, spremenljivk, procedur in funkcij. Delphi izdela za vsak obrazec svoj programski modul. Programski modul se začne z rezervirano besedo **unit** in konča z rezervirano besedo **end**. Modul sestavljajo trije deli, ki jih označujejo rezervirane besede **interface**, **implementation** in **initialization**. Prva dva sta obvezna, medtem ko tretjega zapišemo po potrebi. Rezervirana beseda **interface** označuje vmesnik programskega modula in vsebuje samo napovedi in ne programske kode. Vmesnik je javen, kar pomeni, da lahko do teh napovedi dostopajo tudi drugi programski moduli. rezervirana beseda **implementation** označuje izvedbeni del programskega modula. Vanj zapisujemo kodo procedur in funkcij. Po potrebi lahko tudi v izvedbeni del zapišemo napovedi. Rezervirana beseda **initialization** (začetne nastavitve) označuje blok, v katerem vpeljemo začetne vrednosti spremenljivk, ki jih uporablja modul. Te zadnje rezervirane besede ne uporabljamo, ker ne potrebujemo začetnih vrednosti spremenljivk.
V rezervirani besedi **uses** so našteti vsi drugi moduli, do katerih ima modul dostop. Rezervirana beseda **uses** mora stati pred vsemi drugimi napovedmi.

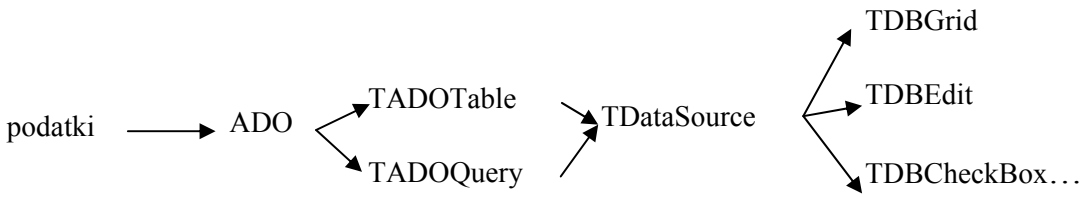
Rezervirana beseda **type** opisuje tip obrazca skupaj z vsemi njegovimi predmeti in njegovimi odzivnimi procedurami. Sprva je tu le stavek za izpeljavo tipa obrazca (`TfrmPrikazSlik`) iz osnovnega tipa `TForm`.
Z rezervirano besedo **var** napovemo javne spremenljivke programskega modula. Na začetku je tu le obrazec `frmPrikazSlik`, ki ga je zapisal Delphi. Na tem mestu napovemo tudi vse druge javne dele, kot so konstante, procedure, …
Zapis {$R *.DFM} je po Delphijevih slovničnih pravilih komentar, saj se začne z zavitim predklepajem. A če komentar takoj nadaljujemo z dolarskim znakom ($), ga spremenimo v navodilo prevajalniku, katere datoteke naj prevaja in kako naj jih prevaja. Tu mu naročimo, naj na tem mestu vstavi prevode vseh datotek DFM. Omenili smo že, da se v teh datotekah hranijo lastnosti, ki jih s pregledovalcem lastnosti nastavljamo gradnikom.

Delphi in podatkovne baze

Delo z velikimi količinami organiziranih podatkov je eno najpomembnejših področij uporabe računalnika nasploh. Zato ni čudno, da Delphi, ki želi biti orodje za celovito izdelavo vseh mogočih računalniških aplikacij, ponuja močna orodja za hitro izdelavo programov za delo s podatkovnimi bazami. Že osnovna različica Delphija omogoča delo z lokalnimi podatkovnimi bazami tipa Paradox, dBase, FoxPro, Access in InterBase, kot tudi s podatki, organiziranimi v običajnih datotekah. Njegova najmočnejša različica pa omogoča delo tudi z bazami tipa Oracle, Sybase in drugimi strežniško-odjemalsko organiziranimi podatkovnimi bazami.
Za to, da Delphi lahko dela z različnimi tipi podatkovnih baz (tudi hkrati), je odgovoren programski vmesnik, ki se imenuje Borland Database Engine ali na kratko BDE. BDE sestavlja množica funkcij, ki delujejo na ravni operacijskega sistema. Te skrbijo za prenos podatkov iz podatkovnih baz v taki obliki, da jih lahko v programu neposredno uporabljamo.

Gradniki za delo s podatkovnimi bazami

Za delo s podatkovnimi bazami imamo v Delphiju vrsto gradnikov. Te najdemo združene v zavihke. Najpomembnejši gradniki so na zavihkih `Data Access, Data Controls, dbGo (ADO gradniki)` in še nekaterih, ki pa jih mi ne bomo uporabljali.
Shematsko si lahko pot od podatkovne baze do prikaza podatkov v programu predstavljamo z naslednjo skico:



Slika40. Gradniki za delo s podatkovnimi bazami

Slika pove naslednje: Imamo bazo. Iz nje pridobimo podatke. To nam omogočata gradnika `ADOTable` in `ADOQuery`. To so podatki v »surovi« obliki. Da poenotimo pogled na vse in jih morebiti še ustrezno predelamo, te gradnike povežemo z gradnikom `DataSource`, ki podatke oblikuje na tak način, da jih lahko prikažemo in obdelujemo s pomočjo gradnikov vrste `Data Controls (DBGrid, DBEdit, …)`, ki so gradniki za vizualni prikaz podatkov.

Gradniki Data Access

Gradniki `Data Access` so gradniki za dostop do podatkov. Ti gradniki nimajo svoje vidne podobe in jih na samem uporabniškem vmesniku ni. To pomeni, da ikono gradnika vidimo samo med oblikovanjem programa, ne pa tudi med izvajanjem. V praksi večinoma uporabljamo le podatkovni vir `DataSource`. Gradnik `DataSource` v programu predstavlja posrednika med podatki v tabeli in gradnikom za prikaz podatkov.

Gradniki Data Controls

Gradniki `Data Controls` so gradniki za prikaz podatkov. Teh gradnikov je v Delphiju več. Najpogosteje se uporablja gradnik tipa `TDBGrid` (gradnik za prikaz podatkov v tabelarični obliki). Obstajajo pa tudi gradnik tipa `TDBEdit` (gradnik za prikaz besedila), gradnik tipa `TDBImage` (gradnik za prikaz grafičnih podatkov) in še veliko drugih gradnikov, ki prikazujejo besedilo v različnih oblikah (izbirna polja, seznami, …). Imena vseh gradnikov se začenjajo z oznako DB, ustrezni tip pa z TDB. Ti gradniki imajo svojo grafično podobo in jih med izvajanjem programa vidimo na uporabniškem vmesniku. Služijo za to, da uporabnik vidi podatke, ki smo jih iz baze pridobili v gradnik `Data Access` (`DataSource`). Gradniki za prikaz podatkov torej dobivajo podatke od gradnikov `Data Access` (`DataSource`) in ne od gradnikov `DbGo` (`ADOTable`).

ADO gradniki

Na zavihku `DbGo` najdemo gradnike `ADO` (ADO – Microsoft ActiveX Data Objects). Ti zagotovijo povezavo do podatkovnih virov. Dopuščajo dostop do ADO podatkovnih baz in dobivajo podatke iz tabele v bazi.
Trije bistveni gradniki so `ADOConnection` za povezavo na bazo, `ADOCommand` za izvajanje ukazov SQL in `ADODataSet`, ki vsebuje rezultate ukazov. Poleg tega obstajajo še trije gradniki. To so `ADOTable`, `ADOQuery` in `ADOStoredProc`. Uporabljamo jih za ADO aplikacije. Tudi ti gradniki nimajo svoje vidne podobe.
Aplikacija narejena z `ADO` gradniki ne zahteva vmesnika BDE. BDE je bil standarden v začetnih verzijah Delphija, vendar Borland meni, da sedaj ni več potreben. Pri dostopu do katerekoli baze z `ADO` gradniki potrebujemo ADO knjižnico. `ADO` gradnike uporabimo, če želimo delati samo v okolju Windows in uporabljamo Access ali ostale Microsoftove podatkovne baze.

Uporaba gradnikov

Najprej bomo potrebovali bazo, ki smo jo naredili v Accessu. Ko se bomo povezali nanjo, bomo lahko črpali podatke iz nje. Potrebovali bomo gradnike za delo s podatkovno bazo, ki smo jih opisali zgoraj. Uporabili bomo gradnika za črpanje podatkov iz baze. To sta gradnika `Table` in `Query`. V njih najdemo vse lastnosti in metode za neposredno delo s polji v podatkovni bazi. Gradnika zagotovita povezavo do podatkovnega vira `DataSource`. Gradnik `DataSource` črpa podatke iz gradnikov `Table` ali `Query`, kjer so sedaj shranjeni podatki. Predstavlja posrednika med tabelo in gradnikom za prikaz podatkov. Kot smo omenili, je gradnikov za prikaz podatkov več. Izbiramo lahko med `DBGrid`, `DBEdit`, `DBCheckBox`, …

Uporabljeni gradniki za povezavo na bazo

Prvi uporabljeni gradnik je tipa `TDataSource` (). Najdemo ga na zavihku `Data Access`, kjer so še drugi gradniki za dostopanje do baz podatkov. `DataSource` je ključni del programov v Delphiju, ki delajo z bazami podatkov. Omogoča gradnikom `Data Controls`, ki prikazujejo podatke, da do podatkov sploh pridejo. Zato mora vsaka forma, ki vsebuje kak gradnik `Data Controls`, vsebovati tudi gradnik `DataSource`. Funkcija gradnika `DataSource` je torej, da zagotovi povezavo med gradnikom, ki vsebuje podatke (npr. gradnik `ADOTable`) in gradnikom za prikaz podatkov (npr. `DBGrid`). Gradnik `DataSource` je bistven pri delu s podatkovnimi bazami, ker gradnik za dostop do podatkov ne more komunicirati direktno z gradnikom za prikaz podatkov.
Gradnik `DataSource` postavimo na poljubno mesto na formi, saj ni važno, kje je. Kot smo že povedali, nima svoje grafične podobe in je med izvajanjem programa neviden.
Drugi gradnik je tipa `TADOTable`. Najdemo ga na zavihku `dbGo`. Predstavlja tabelo, dobljeno iz podatkov, shranjenih v bazi . Gradnik `ADOTable` zagotavlja direkten dostop do vseh zapisov in polj vsake tabele v podatkovni bazi.

Tudi ikono, ki označuje uporabo tega gradnika,  , postavimo na poljubno mesto na formi.

Skica našega dostopa do baze je torej naslednja:

Podatkovna baza (zajem podatkov)	Data Access (podatki v programu)	Data Control (prikaz podatkov)

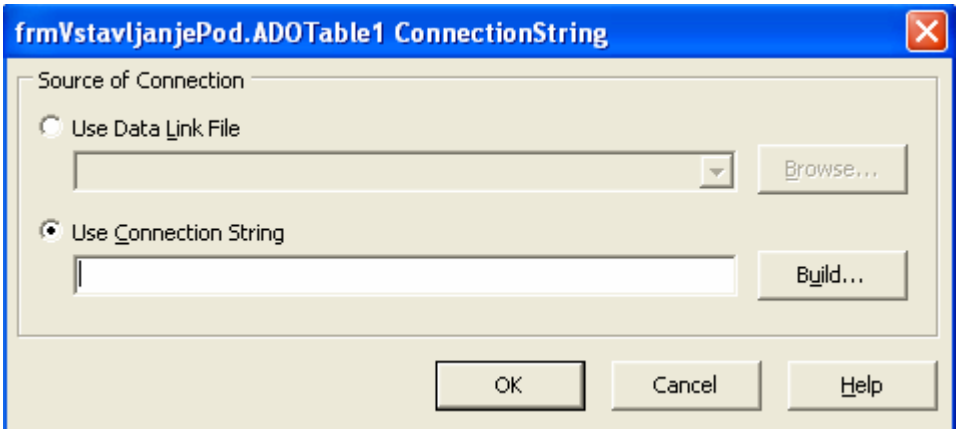
podatki → ADO → ADOTable → TDataSource → TDBGrid

Slika41. Gradniki, ki jih bomo uporabili

Ponovimo še enkrat. Imamo bazo. Iz nje pridobimo podatke. To nam omogoča gradnik `ADOTable`. To so podatki v »surovi« obliki. Zato gradnik `DataSource` podatke oblikuje na tak način, da jih lahko prikažemo in obdelujemo s pomočjo gradnikov vrste `Data Controls`, ki so gradniki za vizualni prikaz podatkov.

Povezava na bazo

Zato, da podatke v bazi sploh lahko uporabljamo, moramo torej vzpostaviti povezavo med programom in bazo podatkov. Zato si oglejmo, kako se priključimo na bazo, kjer imamo shranjene podatke. Povezali se bomo na bazo z imenom **PodatkiUmetnine.mdb**, ki smo jo naredili prej v Accessu.
Na formo postavimo gradnik tipa `TADOTable`. To bo edini gradnik te vrste, poimenovali ga bomo `ADOPrikaz`. Ta bo dobil podatke iz baze. Povežemo ga z bazo, narejeno v Accessu. Ena od osnovnih lastnosti tega gradnika je lastnost *`ConnectionString`*.
Ta niz nam pove, kje je shranjena podatkovna baza in kako dostopamo do nje. Nastavimo ga tako, da uporabimo pregledovalnik lastnosti za gradnik `ADOPrikaz`. Poiščemo lastnost *`ConnectionString`*, kliknemo dvakrat na pravokotniček in dobimo naslednje okno:



Slika42. Nastavitev lastnosti ConnectionString

Ko delamo povezavo, imamo dve možnosti:

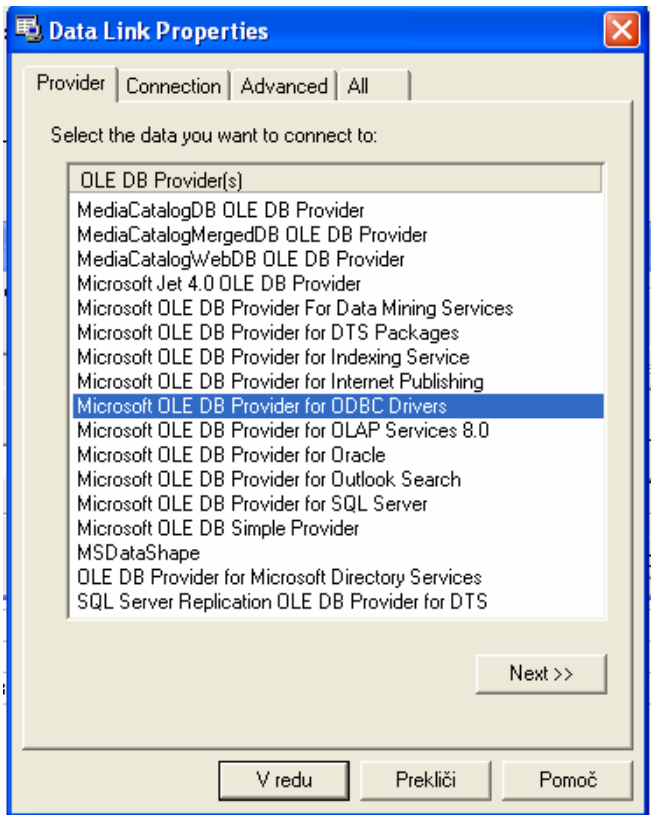
- uporabimo Data Link File (.UDL)
  Ta način povezave na bazo uporabimo, če imamo že vnaprej pripravljeno tako imenovano datoteko Data Link. Čeprav lahko damo datoteki Data Link katerokoli končnico, je priporočljiva uporaba končnice UDL. Kako mora biti oblikovana datoteka UDL, lahko vidimo v datoteki `dbdemos.udl`, ki se ustvari pri namestitvi Delphija. Njena vsebina je:

```
[oledb]
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source= C:\Program Files\Common Files\Borland
Shared\Data\dbdemos.mdb;
```

Vsebuje torej dva ključna podatka. Prvi je način povezovanja (oskrbovalec podatkov), drugi pa izvor podatkov (nahajališče baze podatkov). Datoteko Data Link lahko pripravimo z uporabo poljubnega tekstovnega urejevalnika.

- naredimo povezavo ročno
  V okence Use `ConnectionString` lahko neposredno vpišemo ustrezni povezovalni niz. Bolj običajno si pomagamo s klikom na gumb ***Build***. Tu nam program preko zaporedja korakov avtomatično ustvari ustrezni povezovalni niz.

Mi bomo povezavo naredili ročno. Kliknemo na gumb ***Build*** in dobimo naslednje:



Slika43. Nastavitev oskrbovalca podatkov

Na izbiro imamo štiri zavihke. Uporabili bomo prva dva (***Provider*** in ***Connection***). Na prvem (***Provider***) bomo izbrali oskrbovalca podatkov. Izberemo ***Microsoft Jet 4.0 OLE DB Provider***. Ta nam omogoča dostop do vira podatkov, ki je baza podatkov, pripravljena s programom MS Access.
Na naslednjem zavihku (***Connection***) bomo poiskali bazo, ki smo jo naredili v Accessu. Kliknemo na gumb ... in poiščemo datoteko z bazo, ki smo jo naredili (**PodatkiUmetnine.mdb**). Nato preverimo povezavo (***Test Connection***). Če se lahko na bazo uspešno povežemo, smo povezovalni niz uspešno nastavili. Če pa povezava ni uspešna, poskusimo znova. Podatke na ostalih zavihkih pustimo tako, kot so nastavljeni. Ko smo vse uspešno naredili, kliknemo še ***OK***. S ponovnim klikom na ***OK*** zapremo okno, kjer smo nastavili lastnost `ConnectionString`.

Če si v pregledovalniku lastnosti sedaj ogledamo vrednost lastnosti `ConnectionString` , je ta videti nekako takole:

```
Provider=Microsoft.Jet.OLEDB.4.0;DataSource=C:\NewFolder\PodatkiUmet
nine.mdb;Persist Security Info=False
```

To bi pomenilo, da moramo ob prenosu našega programa na kak drug računalnik datoteko z bazo imeti točno na napisanem mestu, na predpisanem diskovnem pogonu in v predpisanem imeniku. Ker ne želimo, da je datoteka vezana na točno določen imenik (disk ***C***, imenik ***NewFolder)***, bomo lastnost `DataSource` ročno spremenili v

```
Data Source=baza\PodatkiUmetnine.mdb;
```

Če imamo torej našo aplikacijo v imeniku s potjo X, moramo torej tam narediti podimenik baza (torej s potjo ***X\baza***) in datoteko z bazo **PodatkiUmetnine.mdb** postaviti tja.
Z vzpostavitvijo povezave smo poskrbeli, da se podatki iz baze, kadar jih v programu potrebujemo, avtomatsko "preselijo" v gradnik `ADOPrikaz`. Da lahko gradniki za vizualni prikaz podatkov dostopajo do teh podatkov, potrebujemo še vmesni člen. To je gradnik tipa `TDataSource`, ki bo podatke, shranjene v gradniku `ADOPrikaz`, preoblikoval v obliko, da jih bomo lahko prikazali na formi.
V zavihku `Data Access` si izberemo gradnik vrste `TDataSource` in ga s pomočjo vlečenja postavimo na poljubno mesto na formo. Položaj gradnika na formi ni pomemben, saj tako ali tako med izvajanjem programa ni viden. Gradnik bomo poimenovali `IzvorPrikazSlike.`
Seveda bi lahko vso to kodo napisali "ročno". Vendar je veliko hitreje, če omenjene akcije (postavljanje gradnikov, določanje lastnosti, povezav, …) opravimo na vizualni način in pustimo Delphiju, da kodo gradi avtomatsko.
V pregledovalniku lastnosti smo zapisali naslednje lastnosti:

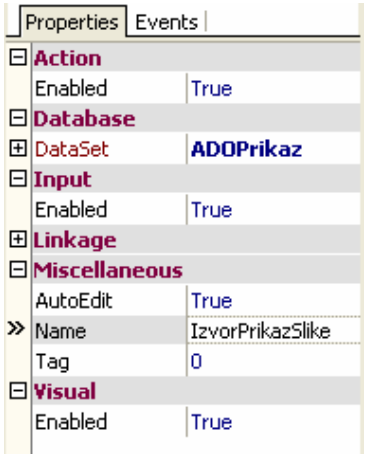
```
ADOTable.Name = ADOPrikaz;
ADOPrikaz.ConnectionString = Provider=Microsoft.Jet.OLEDB.4.0; Data
Source= baza\PodatkiUmetnine.mdb;Persist Security Info=False;
DataSource.Name = IzvorPrikazSlike;
```

Oglejmo si, kako je v tem trenutku videti koda, ki jo je zgradil Delphi.

```
1  unit PrikazSlike;
2
3  interface
4
5  uses
6    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
7    Dialogs, DB,  ADODB;
8
9  type
10   TfrmPrikazSlik = class(TForm)
11     IzvorPrikazSlike: TDataSource;
12     ADOPrikaz: TADOTable;
13
14   private
15     { Private declarations }
16   public
17     { Public declarations }
18   end;
19
20 var
21   frmPrikazSlik: TfrmPrikazSlik;
22
23 implementation
24
25 {$R *.dfm}
26
27 end.
```

Slika44. Urejevalnik kode po nanosu gradnikov DataSource in ADOTable na formo

Vse omenjene lastnosti (povezovalni nizi, ...) so shranjene na drugih mestih (v datotekah s podaljškom dfm) in jih bo Delphi v kodo vključil v samem procesu prevajanja in povezovanja. V programu bi seveda lahko črpali podatke iz več baz, ali pa bi podatke pridobivali kot rezultat različnih poizvedb (`Query`). Zato bi potrebovali več gradnikov za vzpostavitev povezave z bazo. Prav tako bi lahko vzpostavili več gradnikov za preoblikovanje podatkov. Ker je problem, ki ga opisujemo, enostaven, delamo le z eno bazo in imamo le po enega od gradnikov ustrezne vrste (`TADOTable` in `TDataSource`). Kljub temu moramo gradnika `ADOPrikaz` in `IzvorPrikazSlike` povezati med seboj. Nad gradnikom `IzvorPrikazSlike` uporabimo pregledovalnik lastnosti (Object Inspector) in nastavimo povezavo. Pri skupini lastnosti *`Database`* poiščemo lastnost *`DataSet`*. Vidimo, da je okence prazno (lastnost ni nastavljena). Kliknemo na puščico, da dobimo spustni seznam. V njem imamo na izbiro le `ADOPrikaz` in kliknemo nanj.



Slika45. Nastavitev lastnosti DataSet

S kodo bi to zapisali na naslednji način:

```
IzvorPrikazSlike.DataSet = ADOPrikaz
```

Omenili smo že, da Delphi vse nastavitve, ki jih izvršujemo v pregledovalniku lastnosti, zapisuje v datoteko .dfm. Poglejmo si, kako v tem trenutku izgledajo nastavitve, ki smo jih zapisali za gradnik vrste `TDataSource`, poimenovanega `IzvorPrikazSlike`. V datoteko **PrikazSlike.dfm** je Delphi zapisal naslednje vrstice:

```
object IzvorPrikazSlike: TDataSource
  DataSet = ADOPrikaz
  Left = 16
  Top = 216
end
```

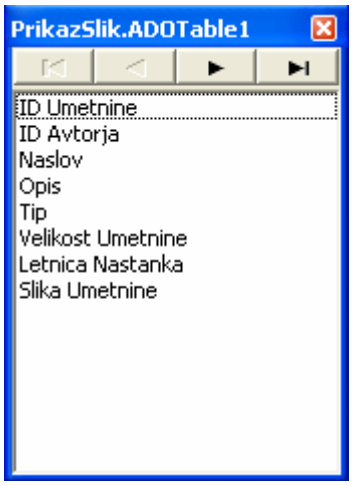
Na koncu moramo nastaviti še ime tabele, do katere dostopa gradnik `ADOPrikaz`. Spet uporabimo pregledovalnik lastnosti in nastavimo lastnost *`TableName`* na ime tabele. Ponudi se nam seznam vseh treh tabel, ki smo jih naredili v Accessu. Ker bomo v osrednjem delu prikazali podatke o umetninah, bomo uporabili tabelo ***Umetnine***, saj so tam zbrani tisti podatki, ki nas zanimajo. Ustrezna koda bi bila

```
ADOPrikaz.TableName = Umetnine
```

Gradniku `ADOTable` moramo povedati tudi to, katera polja iz tabele ***Umetnine*** v bazi naj črpa. Privzeto se ne prenesejo nobena polja, zato v gradniku `ADOPrikaz` še ni nobenega. Zato po povezavi gradnikov oblikujemo seznam polj. Dvakrat kliknemo na gradnik `ADOPrikaz`. Odpre se prazen

seznam polj. Kliknemo gumb ***Add***, označimo vsa polja, ki jih želimo prenesti iz ustrezne tabele v bazi, in kliknemo ***OK***. Mi bomo prenesli kar vsa polja iz tabele ***Umetnine***.

Seznam polj je naslednji:



Slika46. Seznam polj

Ko pogledamo na naš pregledovalnik kode v Delphiju, v datoteko **PrikazSlike.pas**, opazimo, da so se vsa imena stolpcev v tabeli spremenila v
Ime Gradnika + Ime Polja Tabele

Tako so polja iz tabele ***Umetnine***, poimenovana na naslednji način:

```
ADOPrikazIDUmetnine: TAutoIncField;
ADOPrikazIDAvtorja: TWideStringField;
ADOPrikazNaslov: TWideStringField;
ADOPrikazOpis: TWideStringField;
ADOPrikazTip: TWideStringField;
ADOPrikazVelikostUmetnine: TWideStringField;
ADOPrikazLetnicaNastanka: TWideStringField;
ADOPrikazSlikaUmetnine: TBlobField;
```

Vidimo tudi, kako je Delphi preslikal ustrezne tipe iz baze v tipe, kot jih pozna sam. Če v urejevalniku kode pogledamo kakšno kodo je zgradil program Delphi do sedaj, vidimo naslednje:

```
unit PrikazSlike;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs;

type
  TfrmPrikazSlik = class(TForm)
    IzvorPrikazSlike: TDataSource;
    ADOPrikaz: TADOTable;
    ADOPrikazIDUmetnine: TAutoIncField;
    ADOPrikazIDAvtorja: TWideStringField;
    ADOPrikazNaslov: TWideStringField;
    ADOPrikazOpis: TWideStringField;
    ADOPrikazTip: TWideStringField;
```

```
    ADOPrikazVelikostUmetnine: TWideStringField;
    ADOPrikazLetnicaNastanka: TWideStringField;
    ADOPrikazSlikaUmetnine: TBlobField;

  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmPrikazSlik: TfrmPrikazSlik;

implementation

{$R *.dfm}

end.
```

V pregledovalniku lastnosti pa smo nastavili:

```
ADOTable.Name = ADOPrikaz;
ADOPrikaz.ConnectionString = Provider=Microsoft.Jet.OLEDB.4.0; Data
Source= baza\PodatkiUmetnine.mdb;Persist Security Info=False;
DataSource.Name = IzvorPrikazSlike;
IzvorPrikazSlike.DataSet = ADOPrikaz;
ADOPrikaz.TableName = Umetnine;
```

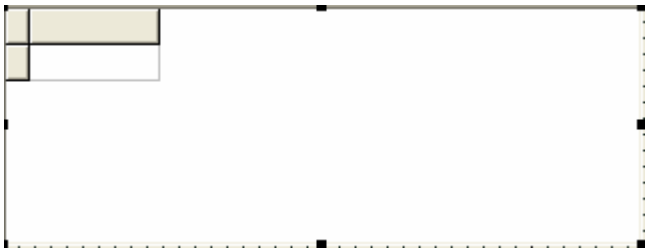
V datoteki **PrikazSlike.dfm** so lastnosti zapisane na naslednji način:

```
  object IzvorPrikazSlike: TDataSource
    DataSet = ADOPrikaz
    Left = 16
    Top = 216
  end
  object ADOPrikaz: TADOTable
    Active = True
    ConnectionString =
      'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
baza\PodatkiUmetnine.mdb;Persist Security Info=False'
    CursorType = ctStatic
    TableName = 'Umetnine'
    Left = 48
    Top = 216
    object ADOPrikazIDUmetnine: TAutoIncField
      DisplayWidth = 12
      FieldName = 'ID Umetnine'
      ReadOnly = True
    end
    object ADOPrikazIDAvtorja: TWideStringField
      DisplayWidth = 10
      FieldName = 'ID Avtorja'
      Size = 50
    end
    object ADOPrikazNaslov: TWideStringField
      DisplayWidth = 12
      FieldName = 'Naslov'
      Size = 50
    end
```

```
    object ADOPrikazOpis: TWideStringField
      DisplayWidth = 35
      FieldName = 'Opis'
      Size = 50
    end
    object ADOPrikazTip: TWideStringField
      DisplayWidth = 5
      FieldName = 'Tip'
      Size = 50
    end
    object ADOPrikazVelikostUmetnine: TWideStringField
      DisplayWidth = 17
      FieldName = 'Velikost Umetnine'
      Size = 50
    end
    object ADOPrikazLetnicaNastanka: TWideStringField
      DisplayWidth = 17
      FieldName = 'Letnica Nastanka'
      Size = 50
    end
    object ADOPrikazSlikaUmetnine: TBlobField
      DisplayWidth = 14
      FieldName = 'Slika Umetnine'
    end
  end
```

Pregledovanje podatkov iz baze

Ko smo na opisan način vzpostavili povezavo med našim programom in bazo, imamo s tem dostop do podatkov v bazi. Denimo, da jih želimo prikazati. V gradniku `IzvorPrikazSlike` imamo podatke v obliki, primerni za prikaz z različnimi gradniki vrste `Data Controls`. Eni od bolj uporabljanih so gradniki tipa `TDBGrid`. Gradnik tipa `TDBGrid` najdemo na zavihku `Data Controls`. Uporablja se za listanje skozi zapise, dobljene iz baze. Podatke prikazuje v tabelarični obliki.



Slika47. Gradnik DBGrid

Gradnik te vrste je po zagonu programa viden, zato moramo vedeti, kam ga postaviti. Gradnik dobi ime `DBGrid1,` zato ga preimenujemo v `DBTabelaUmetnine.` Nato ga povežemo z gradnikom `DataSource`, tako da v pregledovalniku lastnosti nastavimo

```
DBTabelaUmetnine.DataSource = IzvorPrikazSlike
```

Gradnik `DBGrid` lahko prikaže vse stolpce iz podatkovne baze, ali pa samo določene. Tabeli lahko spreminjamo tudi širino stolpcev. Mi bomo prikazali vse stolpce iz podatkovne baze iz tabele ***Umetnine***. Če bi hoteli prikazati samo določene stolpce iz tabele, bi to nastavili v pregledovalniku lastnosti. Izbrali bi lastnost *Columns*, dvakrat kliknili v polje poleg lastnosti in izbrali stolpce, ki jih želimo prikazati.
Na formo postavimo še gradnik tipa `TDBNavigator.` Tudi tega najdemo na zavihku `Data Controls`. Uporabljamo ga za premikanje skozi podatke, zapisane v bazi. Z njim lahko izvajamo tudi operacije, kot so vstavljanje praznega zapisa, osveževanje podatkov, brisanje zapisa, …

Gradnik tipa `TDBNavigator` lahko uporabimo na formi, ki vsebuje gradnik, namenjen prikazu podatkov. Taka gradnika sta na primer `DBGrid`, ki smo ga uporabili in opisali prej, ali pa `DBEdit`, ki lahko prikaže eno celico iz tabele. S pomočjo gradnika `DBNavigator` uporabniku omogočimo lažje premikanje med podatki, njihovo urejanje in pregledovanje. V ta namen je gradnik sestavljen iz množice gumbov. Ko izberemo en gumb gradnika `DBNavigator`, se izvede njegova funkcija, kot jo bomo opisali v nadaljevanju. S tem gradnikom lahko prikažemo enega ali vse izmed naslednjih gumbov:

Slika48. Gradnik DBNavigator

Opišimo, kaj ti gumbi omogočajo. Opisali jih bomo po vrsti od leve proti desni.

Gumb	Funkcija gumba
Prvi	Postavi se na prvi zapis v bazi, onemogoči uporabo gumbov Prvi in Prejšnji in omogoči uporabo gumbov Naslednji in Zadnji.
Prejšnji	Postavi se na zapis pred trenutnim (označenim), omogoči uporabo gumbov Zadnji in Naslednji.
Naslednji	Postavi se na naslednji zapis in omogoči gumba Prvi in Prejšnji.
Zadnji	Postavi se na zadnji zapis v bazi, onesposobi gumba Zadnji in Naslednji, usposobi gumba Prvi in Prejšnji.
Vstavi	Vstavi nov zapis pred trenutnim zapisom.
Briši	Izbriše trenutni zapis in označi naslednji zapis za trenutnega.
Uredi	Postavi zapis v stanje urejanja.
Položaj	Zapiše spremembo trenutnega zapisa v bazo.
Prekliči	Prekliče urejanje trenutnega zapisa in ga vrne v obliko pred urejanjem.
Osveži	Posodobi in obnovi podatke v bazi. Največkrat se ga uporablja pri delu v omrežju, kjer program uporablja več uporabnikov.

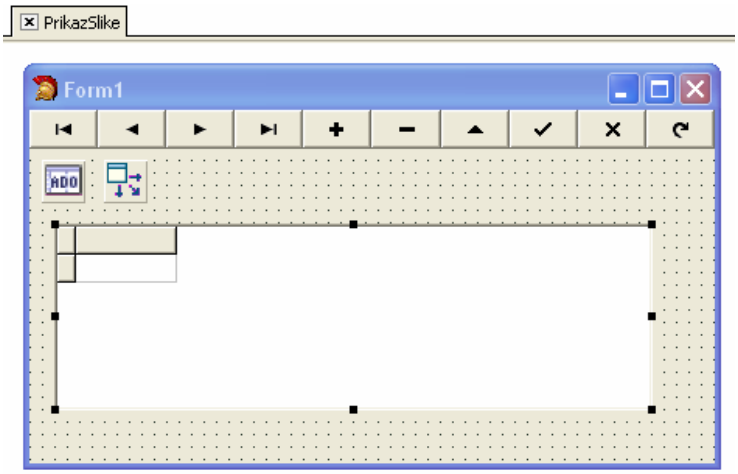
Tudi ta gradnik moramo povezati z izvorom podatkov *DataSource*. V pregledovalniku lastnosti nastavimo:

```
DBNavigator.DataSource = IzvorPrikazSlike;
```

Ko smo že v pregledovalniku, nastavimo še

```
DBNavigator.Align = alTop;
```

Lastnost *Align* nam postavi gradnik na določeno mesto. Na izbiro imamo: *alBottom, alClient, alCustom, alLeft, alNone, alRight in alTop.* Mi smo postavili gradnik `DBNavigator` čisto na vrh naše forme.V tem trenutku je naše okno videti takole



Slika49. Izgled forme po nanosu naštetih gradnikov

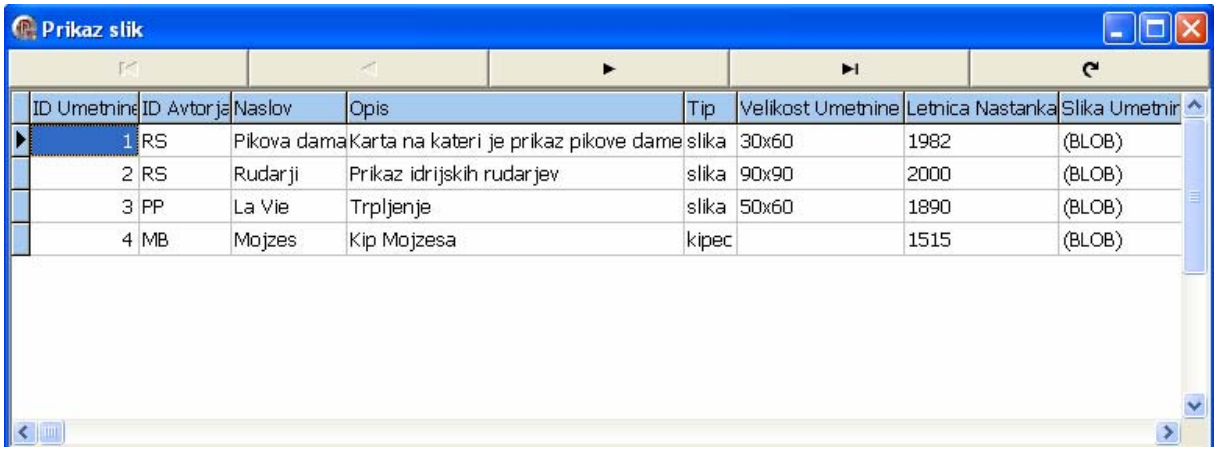
Program poženemo, a se nič ne spremeni. Gradnik `DBTabelaUmetnine` nam ne pokaže nobenih podatkov. Privzeto namreč izvori podatkov (tabele) niso aktivni, saj pogosto želimo šele programsko povedati, da naj se nekaj začne dogajati s podatki. Če pa želimo videti podatke takoj po zagonu programa, v pregledovalniku lastnosti nastavimo

```
ADOPrikaz.Active = True;
```

Ker bomo podatke dodajali in brisali s pomočjo programa napisanega v Delphiju, nekaterih gumbov na `DBNavigatorju` ne bomo rabili, zato jih bomo odstranili. To naredimo tako, da nastavimo lastnosti `DBNavigatorja` na:

```
VisibleButtons.nbInsert = False;
VisibleButtons.nbDelete = False;
VisibleButtons.nbEdit = False;
VisibleButtons.nbPost = False;
VisibleButtons.nbCancel = False;
```

Če sedaj poskusimo zagnati program, zagledamo želene podatke.



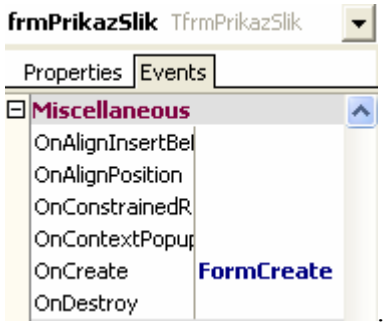
Slika50. Izled forme po zagonu programa

Vse lastnosti gradnikov smo nastavili v pregledovalniku lastnosti, omenili pa smo, da jih lahko nastavimo tudi programsko. Da bomo prikazali obe možnosti, smo najprej nastavili lastnosti gradnikov v pregledovalniku lastnosti, nato pa bomo prikazali, kakšen je izgled procedure, če bi vse te lastnosti nastavili programsko. Seveda moramo poskrbeti, da se vse te lastnosti nastavijo takoj, ko program

zaženemo. Zato uporabimo dogodek forme *OnCreate*, ki se zgodi ob zagonu programa, tik preden se forma ustvari. Napisati moramo torej ustrezen podprogram.
Kliknemo na samo formo, v pregledovalniku dogodkov poiščemo dogodek *OnCreate*. Dvakrat kliknemo poleg dogodka v prazen pravokotniček in odpre se urejevalnik kode s kodo

```
procedure TfrmPrikazSlik.FormCreate(Sender: TObject);
var
begin
end;
```

v pregledovalniku lastnosti pa se izpiše:



Slika51. Izbira dogodka OnCreate

Ker imamo formo poimenovano `frmPrikazSlik`, procedura pa se bo izvedla ob kreiranju forme (ob dogodku *FormCreate*), se naša procedura imenuje `TfrmPrikazSlik.FormCreate(Sender: TObject)`, kjer je

- `TfrmPrikazSlik` – ime razreda
- `FormCreate` – dogodek, ob katerem se sproži procedura
- `Sender: TObject` – parameter tipa `TObject`. S tem parametrom dobi procedura informacijo, kateri predmet jo je poklical.

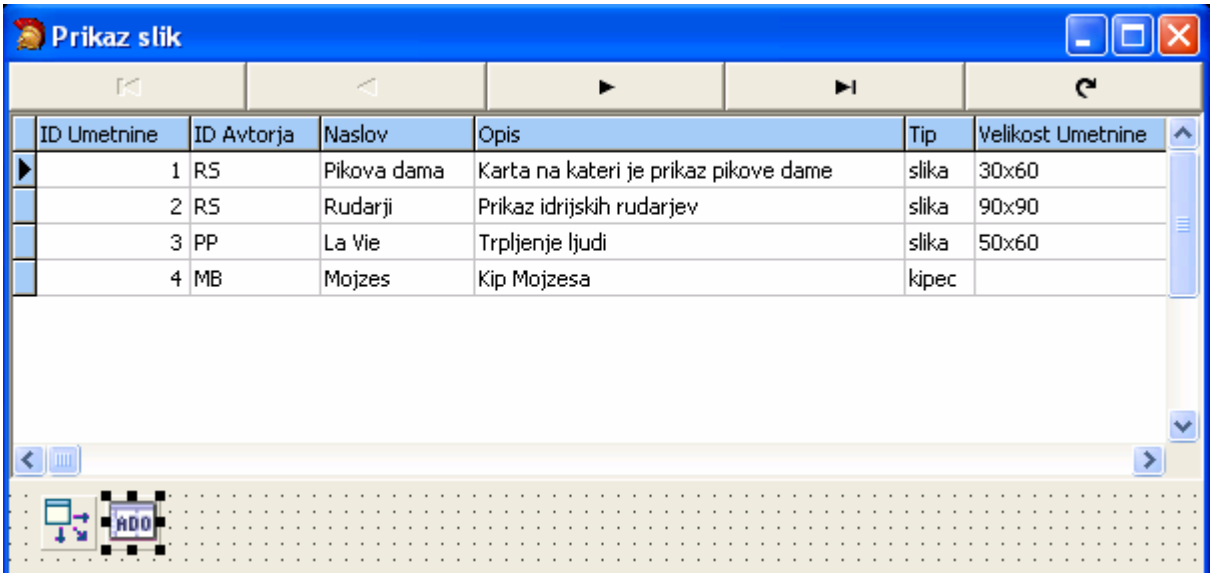
V telo procedure (med `begin` in `end`)pa vnesemo lastnosti, ki smo jih nastavili v pregledovalniku lastnosti. Najprej nastavimo *ConnectionString*, nato pa še povežemo gradnike med seboj tako, kot smo to naredili zgoraj. Po vnosu ustreznih ukazov v telo podprograma je celotni podprogram torej:

```
procedure TfrmPrikazSlik.FormCreate(Sender: TObject);
var baza, conStr: string;
begin
   { nastavitev lastnosti }
   baza := 'baza\PodatkiUmetnine.mdb'; {kje je baza}
   conStr := 'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' +
             baza +';Persist Security Info=False';

   ADOPrikaz.ConnectionString := conStr;
   ADOPrikaz.TableName := Umetnine;
   IzvorPrikazSlike.DataSet := ADOPrikaz;
   DBTabelaUmetnine.DataSource := IzvorPrikazSlike;

   { želimo, da se podatki takoj črpajo iz baze }
   ADOPrikaz.Active := True;
end;
```

In kaj pomeni naslednja koda? Najprej smo v spremenljivko `baza` zapisali niz, ki pove, kje imamo shranjeno bazo, ki smo jo naredili. V spremenljivko `conStr` smo zapisali *`ConnectionString`*. Nato še povežemo podatke med seboj. Povemo, s katero tabelo bomo delali (***Umetnine***), izvore povežemo s tabelo in prikažemo podatke v gradniku `DBTabelaUmetnine`. Na koncu izvore aktiviramo, da se podatki takoj črpajo iz baze.
V metodi *OnCreate* nismo nastavili prav vseh lastnosti. Nekatere smo izpustili, ker smo te nastavili v pregledovalniku lastnosti. Na primer

```
DBNavigator.Align = alTop;
DBNavigator.DataSource = IzvorPrikazSlike;
```

Čeprav smo te lastnosti v kodi izpustili, bo program ob zagonu upošteval tudi te lastnosti, saj smo jih nastavili v pregledovalniku lastnosti. Kot smo povedali že prej, so zapisane v ustrezni datoteki dfm in jih prevajalnik vključi z ukazom `{$R` . V primeru, da bi neko lastnost nastavili v pregledovalniku lastnosti in v metodi *OnCreate*, bi obveljala slednja, ker metoda povozi nastavitve v pregledovalniku lastnosti.
Tako, sestavili smo program, ki zna v tabelarični obliki prikazati podatke iz baze. Naša tabela je sedaj napolnjena s tistimi podatki iz baze, ki smo jih prej ročno vnesli. Forma je sedaj videti takole:



Slika52. Vsi potrebni gradniki za prikaz podatkov

Koda, ki ustreza zapisu zgornje slike, generirana s strani Delphija, pa izgleda takole:

```
unit PrikazSlike;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,  Dialogs, DB, ADODB, Grids, DBGrids;

type
  TfrmPrikazSlik = class(TForm)
    DBTabelaUmetnine: TDBGrid;
    IzvorPrikazSlike: TDataSource;
    ADOPrikaz: TADOTable;
```

```
    DBNavigator1: TDBNavigator;
    ADOPrikazIDUmetnine: TAutoIncField;
    ADOPrikazIDAvtorja: TWideStringField;
    ADOPrikazNaslov: TWideStringField;
    ADOPrikazOpis: TWideStringField;
    ADOPrikazTip: TWideStringField;
    ADOPrikazVelikostUmetnine: TWideStringField;
    ADOPrikazLetnicaNastanka: TWideStringField;
    ADOPrikazSlikaUmetnine: TBlobField;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmPrikazSlik: TfrmPrikazSlik;
implementation

{$R *.dfm}

procedure TfrmPrikazSlik.FormCreate(Sender: TObject);
var baza, conStr: string;
begin
   { nastavitev lastnosti }
   baza := 'baza\PodatkiUmetnine.mdb'; {kje je baza}
   conStr := 'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' +
             baza +';Persist Security Info=False';
   ADOPrikaz.Close;
   ADOPrikaz.ConnectionString := conStr;
   //ADOPrikaz.TableName := Umetnine;
   IzvorPrikazSlike.DataSet := ADOPrikaz;
   DBTabelaUmetnine.DataSource := IzvorPrikazSlike;
   { želimo, da se podatki takoj črpajo iz baze }
   ADOPrikaz.Active := True;
   ADOPrikaz.Open;
end.
```

V pregledovalniku lastnosti ima gradnik `ADOPrikaz` tipa `TADOTable` nastavljene naslednje lastnosti (poleg seveda lastnosti *`Name`*):

```
ConnectionString = Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
baza\PodatkiUmetnine.mdb;Persist Security Info=False;
Tablename = Umetnine;
Active = True
```

V programu imamo tudi gradnik `IzvorPrikazSlike` tipa `TDataSource`, ki ima nastavljeno lastnost

```
DataSet = ADOPrikaz;
```

Gradnik `DBTabelaUmetnine` tipa `TDBGrid` ima nastavljeni lastnosti

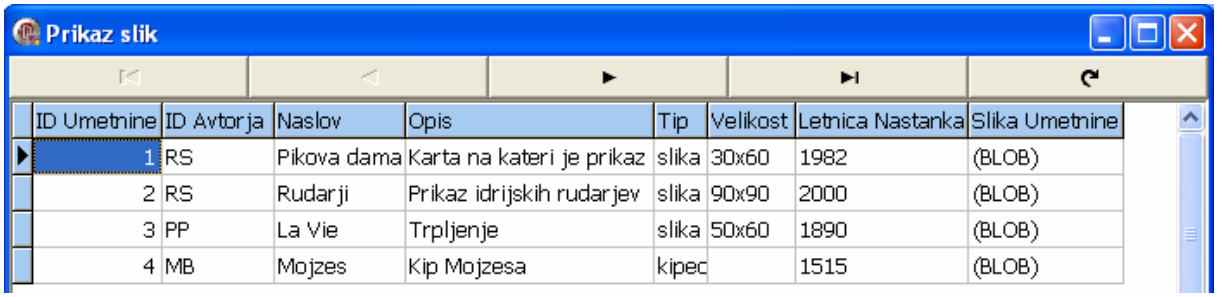
```
DataSource = IzvorPrikazSlike;
Align = alTop
```

Gradnik `DBNavigator` tipa `TDBNavigator` pa ima spremenjeni lastnosti

```
DataSource = IzvorPrikazSlike;
Align = alTop
```

Prikaz slike iz podatkovne baze

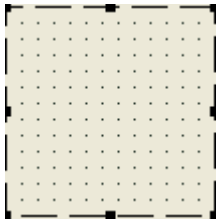
V tem razdelku si bomo ogledali, kako v programu prikažemo in obdelujemo grafične podatke v bazi, narejeni s programom Access. Na začetku, ko smo oblikovali bazo, smo v tabeli ***Umetnine*** zadnje polje poimenovali *Slika Umetnine* in ji določili tip OLE Object. Vanj smo shranili sliko umetnine. Pri prikazu vsebine baze v tabelarični obliki slika ni vidna. V tabeli vidimo le oznako, da gre za podatek tipa BLOB. Mi pa bi radi videli sliko.



ID Umetnine	ID Avtorja	Naslov	Opis	Tip	Velikost	Letnica Nastanka	Slika Umetnine
1	RS	Pikova dama	Karta na kateri je prikaz	slika	30x60	1982	(BLOB)
2	RS	Rudarji	Prikaz idrijskih rudarjev	slika	90x90	2000	(BLOB)
3	PP	La Vie	Trpljenje	slika	50x60	1890	(BLOB)
4	MB	Mojzes	Kip Mojzesa	kipec		1515	(BLOB)

Slika53. Prikazali smo vse podatke razen slike

Znotraj programa Access sliko prikažemo tako, da dvakrat kliknemo na polje *Slika Umetnine.* S tem zaženemo ustrezni, s strani operacijskega sistema določeni pregledovalnik slik, ki nam prikaže sliko. Sedaj pa poglejmo, kako lahko prikažemo sliko v našem programu.
V ta namen bomo potrebovali gradnik, kjer lahko prikazujemo grafične podatke. Tak je na primer gradnik tipa `TImage`, ki ga najdemo na zavihku `Additional`. Slikovno datoteko, ki je lahko bitna slika, ikona ali metadatoteka, naložimo v lastnost *`Picture`* tega gradnika.



Slika54. Gradnik Image

Gradnik `Image`, ki smo ga poimenovali `ADOSlika`, nima nobenih lastnosti, s katerimi bi ga v pregledovalniku lastnosti avtomatsko povezali s katerim od gradnikov (tako kot smo na primer prej povezali `ADOTable` z `DataSource`), zato bomo morali to povezavo ustvariti s pomočjo programske kode. V gradniku `ADOSlika` bomo prikazali sliko umetnine, katere zapis smo označili v gradniku `DBTabelaUmetnine`. To bi lahko naredili na več načinov, denimo tako, da bi se slika pokazala vedno, ko se postavimo v novo vrstico tabele `ADOSlika`. A bolj običajni način je, da se to zgodi le ob kliku na poseben gumb. Zato postavimo na formo gumb `Button1`, ki ga najdemo na zavihku `Standard`. Na tem zavihku namreč najdemo tiste gradnike, ki jih najpogosteje uporabljamo pri sestavljanju uporabniških vmesnikov v programih v Oknih.
Kot smo omenili že prej, in to tudi upoštevali, želimo, da imajo gradniki imena, ki označujejo njihov namen. Zato ga bomo iz privzetega imena `Button1` preimenovali. Kot smo že omenili, moramo to narediti takoj, saj imamo ob kasnejšem preimenovanju gradnikov lahko težave. V pregledovalniku lastnosti poiščemo lastnost *`Name`* in jo nastavimo na `btnPrikazi`. Vidimo, da se ob spremembi imena gradnika spremeni tudi napis na njem in da na gumbu sedaj namesto `Button1` piše *`btnPrikazi`*. Seveda nam to ni všeč. Zato v pregledovalniku lastnosti poiščemo lastnost *`Caption`*

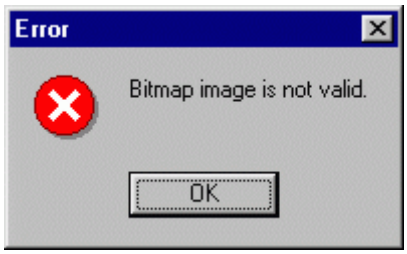
in jo nastavimo na "***Prikaži sliko***". S tem se spremeni napis na gumbu, ime gradnika pa seveda ostane `btnPrikazi`.
Želimo torej, da se ob kliku na gumb v gradniku `ADOSlika` prikaže slika. Zato moramo zapisati ustrezna navodila v podprogram, ki se izvede ob dogodku *OnClick*. Ker je to najbolj pogost dogodek nad gumbi, se ob dvakratnem kliku na gumb znajdemo v urejevalniku kode, z vpisano kodo

```
procedure TfrmPrikazSlik.btnPrikaziClick(Sender:TObject);
begin
end;
```

Do te kode lahko pridemo tudi, če v pregledovalniku dogodkov gumba `btn_prikazi` na zavihku dogodkov (`Events`) poiščemo *OnClick* in dvakrat kliknemo v prazno polje poleg zapisa.
In kakšno kodo bomo vpisali v podprogram? Želimo, da se ob kliku na gumb z napisom ***Prikaži sliko***, prikaže v gradniku `ADOSlika` slika umetnine, ki jo opisuje vrstica tabele, v kateri je kurzor.
Poskusimo prikazati sliko na čisto preprost način. Poizkusimo s kodo

```
procedure TfrmPrikazSlik.btn_prikaziClick(Sender: TObject);
begin
  ADOSlika.Picture.Graphic := ADOPrikazSlikaUmetnine;
end;
```

Seveda nam ne uspe. Problemov je cel kup. Prvi je ta – na kateri zapis se pravzaprav nanaša `ADOPrikazSlikaUmetnine`. Izkaže se, da to ni problem, saj se ta spremenljivka vedno nanaša na ustrezno polje iz tekočega (označenega) zapisa. A napaka



Slika55. Program ne dela, javi nam napako

nam da misliti, da je morda kaj narobe z vsebino naše baze podatkov. Iskanje po pomoči v Delphiju nam sprva pravi, da smo morda izbrali napačni gradnik in da bi bilo bolj prav uporabiti gradnik tipa `TDBImage`. Pa tudi ta ne bo pravi, saj pričakuje grafično datoteko v formatu BMP, naše slike pa so v formatu JPEG. Pomoč pravi, da moramo v tem primeru uporabiti gradnik tipa `TImage`, kar smo storili, in podatke napolniti preko gradnika tipa `TJpegImage`. Da to lahko naredimo (da so ti gradniki na voljo), je potrebno dodati tudi knjižnico za delo s slikami – to je knjižnica `jpeg`. Njeno uporabo napovemo s stavkom

```
uses jpeg;
```

Dodajmo ta stavek in začnimo pisati metodo

```
procedure TfrmPrikazSlik.btn_prikaziClick(Sender: TObject);
var
   SlikaUmetnine : TJpegImage;
begin
  SlikaUmetnine := TJpegImage.Create;
  // v SlikaUmetnine
  ???
  ADOSlika.Picture.Graphic := SlikaUmetnine;
end;
```

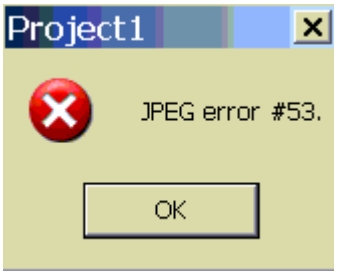
In kako v gradnik `SlikaUmetnine` »spraviti« sliko iz naše baze (torej kaj v zgornji kodi napisati namesto ???) ? Natančna razlaga dogajanja presega okvire naše diplomske naloge, saj bi morali razložiti tokove podatkov, način, kako gradniki tipa `TJpegImage` razumejo "svoje" podatke, ... Tu bomo le skicirali rešitev. Podrobnejšo razlago si lahko preberemo na internetni strani http://delphi.about.com/od/database/l/aa030601a.htm.
Kar se Delphija tiče, je v ustreznem polju nekaj tipa BLOB, torej neko zaporedje dvojiških podatkov. Iz tega zaporedja podatkov moramo ustvariti ustrezno sliko. To naredimo tako, da iz tega zaporedja ustvarimo tok podatkov tipa `ADOBlobStream` z ukazom

```
TADOBlobStream.Create(ADOPrikazSlikaUmetnine, bmRead);
```

Prvi parameter metode je ime polja tipa BLOB, drugi pa je v Delphi vgrajena konstanta, ki nam pove, da bomo ta tok podatkov brali. S tem tokom podatkov bomo napolnili gradnik tipa `TJpegImage`, ki ga bomo potem uporabili kot lastnost *`Picture.Graphics`* gradnika `ADOSlika`, kjer bomo sliko prikazali. Celotno kodo damo v tako imenovani varovalni blok (`try` … `finally` … `end`), ki nam ob neuspešnem poskusu javi napako. Na koncu uničimo narejene objekte in sprostimo potreben spomin. To naredimo z metodo *`Free`*.

```
procedure TfrmPrikazSlik.btn_prikaziClick(Sender: TObject);
var
  bS  : TADOBlobStream;
  Pic : TJpegImage;
begin
  bS := TADOBlobStream.Create
        (ADOPrikazSlikaUmetnine, bmRead);
  try
     Pic:=TJpegImage.Create;
     try
        Pic.LoadFromStream(bS);
        ADOSlika.Picture.Graphic:=Pic;
     finally
        Pic.Free;
     end;
  finally
     bS.Free
  end;
end;
```

Ko program poženemo in kliknemo na gumb z napisom ***Prikaži sliko***, nam program javi napako:



Slika56. Javi nam novo napako

Kaj se je zgodilo? Če pobrskamo po Internetu za opis napake številka 53, preberemo, da je težava v tem, da naj bi bila to napačna oziroma pokvarjena JPEG datoteka. Ker vemo, da je zagotovo shranjena datoteka prava datoteka tipa JPEG, mora biti razlog drugje.
Standard, ki opisuje sestavo datotek tipa JPEG, pravi, da se datoteka začne z zaporedjem bitov '0xFFD8' (šestnajstiški zapis) in konča z zaporedjem bitov '0xFFD9'. Prvo zaporedje bitov 0XFFD8

pomeni SOI (Start of Image), drugo, 0XFFD9, pa pomeni EOI (End of Image). In če bi pogledali, kaj je v ustreznem polju baze zapisano, bi opazili, da na začetku polja ni omenjene vrednosti. Problem je v tem, da Access v polju tipa BLOB ne hrani "čiste" datoteke JPEG, ampak pred samo sliko doda še določene podatke. Da bomo lahko prebrali ustrezne grafične podatke, bomo morali v zgornjem polju tipa BLOB poiskati niz 'FFD8' in prebrati sliko. Žal niz 'FFD8' ni nujno vedno na istem mestu v datoteki. Zato bomo potrebovali funkcijo, ki pove, kje v polju je ta označba SOI in s tem vedeli, od kod moramo začeti prebirati "čiste" podatke o sliki.
Ideja, ki nas pripelje do te funkcije (ki smo jo povzeli po literaturi), je ta, da najprej iz podatkovnega polja tipa BLOB preberemo podatke v tok podatkov `ADOBlobStream`. Ta tok podatkov prebiramo, dokler ne naletimo na ustrezno zaporedje znakov.

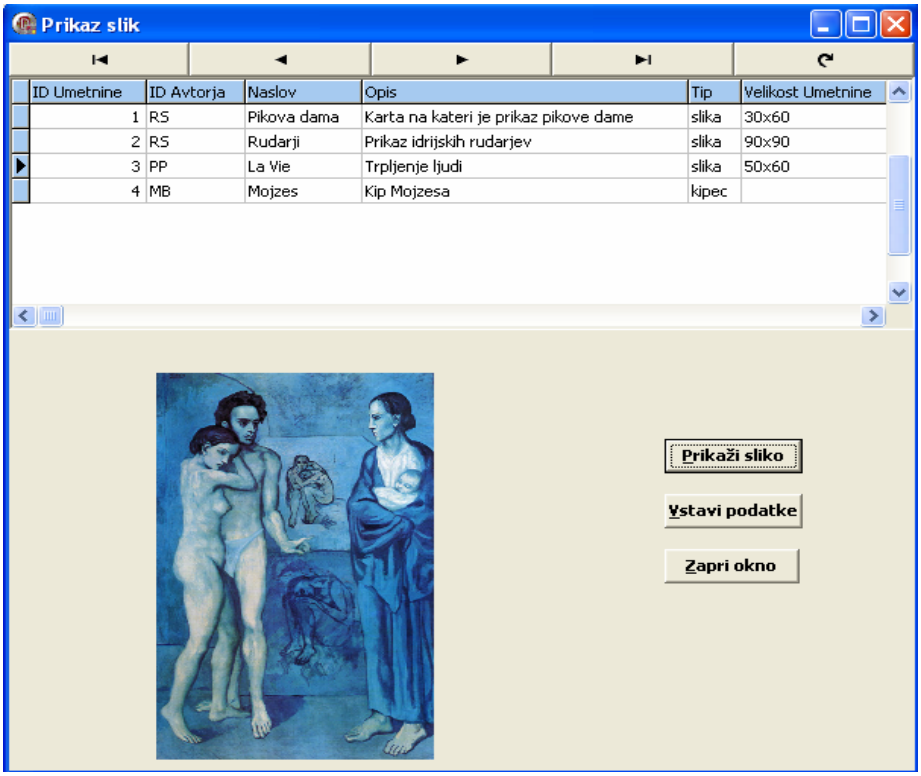
```
function JpegStartsInBlob
         (PicField:TBlobField):integer;
{ poišče, kje v polju PiCField,tipa BLOB, se nahaja oznaka
  SOI }
var
 bS     : TADOBlobStream;
 buffer : Word;
 hx     : string;
begin
 Result := -1;
 bS := TADOBlobStream.Create(PicField, bmRead);
 try
  while (Result = -1) and
        (bS.Position + 1 < bS.Size) do
  { dokler ne najdemo ali pa ne pridemo do konca toka }
  begin
   bS.ReadBuffer(buffer, 1); { preberemo zlog }
   hx := IntToHex(buffer, 2); { pretvorimo v 16. sistem }
   if hx = 'FF' then begin { našli smo 'sumljiv' začetek }
     bS.ReadBuffer(buffer, 1);
     hx := IntToHex(buffer, 2);
     if hx = 'D8' then { našli smo ustrezno oznako }
        Result := bS.Position - 2; { to je naš rezultat }
     else if hx = 'FF' then { ni ustrezen zaključek }
       bS.Position := bS.Position - 1;
   end; //if
  end; //while
 finally
  bS.Free;
 end;  //try
end;
```

Čeprav so že komentarji dovolj zgovorni, razložimo, kako funkcija *JpegStartsInBlob* deluje. Vrne nam položaj niza 'FFD8' v obliki celega števila (tipa integer). Če funkcija ne bo našla tega niza v ustreznem polju, nam bo kot rezultat vrnila število -1. To je tudi začetna vrednost spremenljivke Result (torej tiste, v kateri v funkciji v jeziku Delphi vrnemo rezultat). Nato iz polja PicField, kjer pričakujemo, da je slika v formatu jpeg, naredimo tok podatkov, ki ga bomo brali. Začnemo pregledovati binarne podatke v toku. Podatke pregledujemo, če niza še nismo našli (v spremenljivki Result je -1), ali pa dokler nismo prišli do konca datoteke. Iz toka prebiramo znak za znakom in po dva in dva zlagamo v spremenljivko Buffer. Vrednost te pretvorimo v šestnajstiški zapis. Če ta dva znaka tvorita niz 'FF', potem pogledamo naslednja dva znaka. Če tvorita niz 'D8', potem smo našli pravi niz in je začetek označbe SOI dva znaka nazaj, sicer se postavimo za en znak nazaj in pregledujemo naprej. Na koncu sprostimo objekt bs, ki smo ga naredili na začetku.

Sedaj se znamo v ustreznem podatkovnem toku postaviti na mesto, kjer se začnejo podatki o sliki (na mestu označbe SOI). Zato bomo podprogram za prikaz slike le malce spremenili. S pomočjo omenjene funkcije bomo enostavno preskočili za nas nepotrebne začetne znake. Spremembe so označene s temnejšo barvo. Tam uporabimo funkcijo *JpegStartsInBlob*, ki poišče SOI označbo.

```
procedure TfrmPrikazSlik.btn_prikaziClick(Sender: TObject);
var
  bS  : TADOBlobStream;
  Pic : TJpegImage;
  bb : integer;
begin
  { ustvarimo podatkovni tok iz vsebine ustreznega polja s
    sliko }
  bS := TADOBlobStream.Create
        (ADOPrikazSlikaUmetnine, bmRead);
  try
    { poiščemo oznako SIO }
    bb := JpegStartsInBlob(ADOPrikazSlikaUmetnine);
    { v toku preskočimo do SIO }
    bS.Seek(bb, soFromBeginning);
    { ustvarimo objekt za sliko }
    Pic := TJpegImage.Create;
    try
     { iz toka »poberemo« sliko }
     Pic.LoadFromStream(bS);
     ADOSlika.Picture.Graphic := Pic;
    finally
     Pic.Free;
    end;
  finally
    bS.Free
  end;
end;
```

Formo imamo oblikovano, kodo napisano, zato poženemo program, ki se uspešno prevede.
Ob kliku na gumb ***Prikaži sliko*** se prikaže slika iz vrstice, v kateri je kurzor.



Slika57. Po zagonu programa lahko vidimo tudi sliko

Že prej smo omenili, da bi lahko prikaz slike izvedli tudi drugače, brez tega, da bi moral uporabnik klikniti na poseben gumb. Sestavimo podprogram, ki nam brez klika na gumb prikaže sliko iz vrstice, v kateri je kurzor. Podprogram je sledeč:

```
procedure TfrmPrikazSlik.ADOPrikazAfterScroll(DataSet: TDataSet);
var
  Fld: TBlobField;
begin
  if ADOPrikaz.State = dsBrowse then
  begin
    Fld := ADOPrikaz.FieldByName('Slika Umetnine')
           as TBlobField;
    PrikaziSliko(Fld, ADOSlika);
  end;
end;
```

Dogodek, ki sproži to metodo, je *AfterScroll*, ki se zgodi ob kliku na vrstico tabele in pa takrat, ko se po tabeli premikamo s puščicama ↑ in ↓.
Koda nam pove naslednje: če je tabela v stanju pregledovanja, brskanja, potem poiščemo v vrstici, ki je aktivna, polje z imenom *Slika Umetnine*, ki je tipa `TBlobField`. Nato pa prikažemo sliko s proceduro *PrikaziSliko*, ki je naslednja.

```
procedure PrikaziSliko(Fld: TBlobField; Image1: TImage);
var
  bS  : TADOBlobStream;
  Pic : TJpegImage;
  bb : integer;
begin
  { ustvarimo podatkovni tok iz vsebine ustreznega polja s
    sliko }
```

```
  bS := TADOBlobStream.Create(Fld, bmRead);
  try
    bb := JpegStartsInBlob(Fld); { poiščemo oznako SIO }
    bS.Seek(bb, soFromBeginning); { v toku preskočimo do SIO }
    Pic:=TJpegImage.Create; { ustvarimo objekt za sliko }
    try
     Pic.LoadFromStream(bS); { iz toka »poberemo« sliko }
     Image1.Picture.Graphic:=Pic;
    finally
     Pic.Free;
    end;
  finally
    bS.Free
  end;
end;
```

Telo procedure je podobno proceduri `btn_prikaziClick`, ki smo jo zapisali in komentirali že zgoraj. Vendar se ta procedura ne zgodi ob nekem dogodku. To proceduro bomo uporabili, kadar bomo imeli namen samo prikazati sliko.

Sliko prikažemo v gradniku `Image`, ki smo ga poimenovali `ADOSlika`. Na formo postavimo še dva gumba tipa `TButton`, ki ju bomo uporabili kasneje. Enega poimenujemo `btn_uredi`. Uporabili ga bomo za urejanje in spreminjanje podatkov. Drugega, ki ga bomo uporabili za brisanje podatkov iz podatkovne baze, pa poimenujemo `btn_brisi`.

Vstavljanje podatkov v bazo

Do sedaj smo imeli v Delphiju samo eno formo. Z njo smo prikazovali podatke, ki smo jih imeli v bazi. Da bi bil program bolj pregleden, bomo takrat, ko bomo s programom vstavljali nove podatke, odprli novo okno – zgradili bomo novo formo.
Na formo bomo nanesli gradnike, ki bodo primerni za vstavljanje posameznih podatkov v tabelo ***Umetnine***. Najosnovnejše gradnike za vstavljanje podatkov sicer najdemo na zavihku `Standard`, bolj primerni za vstavljanje podatkov v podatkovno bazo pa so gradniki na zavihku `Data Controls` . Uporabili bomo le nekatere.

Gradniki za vstavljanje

Pri vstavljanju podatkov v bazo bi lahko uporabili kar gumb gradnika `DBNavigator` -  . Vendar bi mi radi vstavljali podatke tako, da bomo primerno oblikovali vnosni obrazec. Prva ideja je, da bi pri tem uporabili nekatere gradnike iz zavihka `Standard`. Ti gradniki so podobni tistim, ki jih pogosto srečamo pri programih v okolju Windows. Prav bi nam prišli gradniki kot so `Label`, `Edit`, `Button` in `ComboBox`. Te poiščemo na zavihku `Standard`, kliknemo nanje in jih prenesemo na ustrezno mesto na formi. S pomočjo označevalnih točk (mali črni kvadratki okrog gradnika) jim določimo ustrezno velikost. Opišimo jih na kratko.

Gradnik Label

Gradnik `Label` je tipa `TLabel.`Namenjen je prikazu teksta. Pogosto je postavljen poleg ostalih gradnikov, najpogosteje poleg gradnikov `Edit`, `ComboBox`, `Memo`, … in jih s tem označuje (uporabljen je kot nalepka z imenom). Osnovna lastnost tega gradnika je *`Caption`*. Če želimo v programu na primer gradniku `Napis` tipa `TLabel` spremeniti tekst, ki ga prikazuje, v »Napaka«, bi napisali

```
Napis.Caption := "Napaka";
```

Če želimo, da je napis že takoj drugačen, uporabimo kar pregledovalnik lastnosti. Poiščemo lastnost *`Caption`* in v vnosno polje vpišemo želeno besedilo.

Gradnik Edit

Gradnik `Edit` je tipa `TEdit.` To je vnosno polje za vnos ene vrstice besedila. Njegova glavna lastnost je *`Text`*, v katero vnesemo besedilo.

Namesto da bi uporabili kombinacijo gradnikov `Label` in `Edit`, bi lahko uporabili gradnik `LabeledEdit`. Z lastnostjo *`Caption`* spreminjamo napis (nalepko), vsebino vnosne vrstice pa nadzoruje lastnost *`Text`*.

Položaj napisa glede na vnosno polje nadzira lastnost `LabelPosition`. Privzeto ima gradnik napis nad vnosno vrstico.

```
LabelPosition = lpAbove;
```

Če želimo imeti napis levo ob vnosni vrstici, lastnost nastavimo na

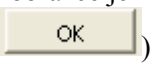
```
LabelPosition = lpLeft;
```

Podobna gradnika sta `Memo` in `RichEdit`. `Memo` je vnosno polje za prikaz večvrstičnega besedila, `RichEdit` pa vnosno polje za prikaz večvrstičnega oblikovanega besedila. Pri obeh je vnešeno besedilo v lastnosti *`Lines`*.

Gradnik Button

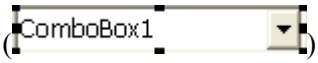
Gradnik `Button` je tipa `TButton` (

). Gumbe v programih največkrat uporabljamo za proženje različnih postopkov. Najpomembnejši dogodek gumba je klik nanj. Ta gradnik smo že uporabili in ga bomo še večkrat. Namesto gradnika `Button` bi lahko uporabili podobne gradnike, kot sta gradnika `SpeedButton` in `BitBtn`. Gradnik `BitBtn` (

) je gumb s sličico za modalno zapiranje obrazcev. Za modalno odprte obrazce je značilno, da jih mora uporabnik, preden jih zapusti, tudi zapreti. Gradnik `SpeedButton` (

) je orodni gumb, ki lahko ostane po kliku vklopljen.

Gradnik ComboBox

Gradnik `ComboBox` je tipa `TComboBox` (

). To je izbirni seznam. Prikaže se kot okno z besedilom v vrsticah, v katerem lahko vrstice izbiramo, podatek pa lahko tudi vtipkamo. Z lastnostjo *`Text`* spreminjamo vsebino vnosne vrstice, z lastnostjo *`Items`* pa vnašamo izbire. Te lahko izbiramo šele po zagonu programa, ko kliknemo na  .
Podoben gradnik je `ListBox`, le da pri njem lahko le izbiramo vrstice, ne moremo pa vtipkati podatka.

Gradniku `ListBox` izbire prav tako vnesemo v lastnost *`Items`*.

Gradniki za vstavljanje v podatkovno bazo

Omenjeni gradniki so sicer uporabni, a bi morali sami skrbeti, da bi iz baze dobili ustrezni podatek in ga prenesli v ustrezno lastnost gradnika (*`Caption`*, *`Text`*, ...) in obratno, da bi se tisto, kar bi vnesli na primer v vnosno vrstico ali izbrali v izbirnem seznamu, preneslo v ustrezni zapis v bazo.
V Delphiju imamo na voljo določene gradnike, ki jih lahko povežemo z gradnikom `DataSource`. Vsi ti gradniki prikazujejo vsebino ene celice iz tabele. Te gradnike najdemo na zavihku `Data Controls`. Ker delamo s podatkovno bazo in bodo ti gradniki prikazovali podatke iz baze, bomo namesto prej opisanih uporabili raje te gradnike.
Najbolj pogosto uporabljenih gradnikov bo gradnik tipa `TDBEdit`.

Ta prikazuje celico iz tabele, na katero smo povezani preko gradnika `DataSource`. Podobno kot gradnik `Edit` ima tudi ta osnovno lastnost *`Text`*, kjer je prikazana vsebina vrstice. Za primer opišimo, kako v gradniku `DBEdit1` prikažemo eno celico iz tabele ***Umetnine***. Prikazali bomo celico *IDAvtorja*.
V pregledovalniku lastnosti nastavimo:

```
DBEdit1.DataSource := frmPrikazSlik.IzvorPrikazSlike;
DBEdit1.DataSet := ADOPrikaz;
DBEdit1.TableName := Umetnine;
DBEdit1.DataField := ID Avtorja;
DBEdit1.DataSet.Active := True;
```

Najprej gradnik `DBEdit1` povežemo z izvorom podatkov in povemo, iz katere tabele bomo črpali podatke (***Umetnine***). Z lastnostjo *`DataField`* povemo, katero polje iz tabele ***Umetnine*** bomo prikazali v gradniku `DBEdit1` in izvor aktiviramo. S tem, ko izvore aktiviramo, se nam v gradniku prikažejo podatki, v bazi pa lahko spreminjamo podatke, jih vstavljamo, brišemo, …

Naslednji uporabljeni gradnik bo gradnik tipa `TDBLookupComboBox`.

Tudi ta gradnik je povezan preko gradnika `DataSource` in prikazuje v obliki izbirnega seznama podatke določenega polja v podatkovni bazi. V lastnosti *`Items`* dobimo naštete vse različne vrednosti, ki jih ima to polje v izbrani tabeli podatkov. Ko dodamo v podatkovno bazo nov tip umetnine, je to videti tudi v gradniku `DBLookupComboBox`. V tem gradniku bomo prikazali tipe umetnin. V pregledovalniku lastnosti nastavimo

```
DBLookupComboBox.DataSource := frmPrikazSlik.IzvorPrikazSlike;
DBLookupComboBox.DataSet := ADOTipi;
DBLookupComboBox.TableName := Tipi Umetnin;
DBLookupComboBox.DataField := Tip;
DBLookupComboBox.KeyField := Tip Umetnine;
DBLookupComboBox.DataSet.Active := True;
```

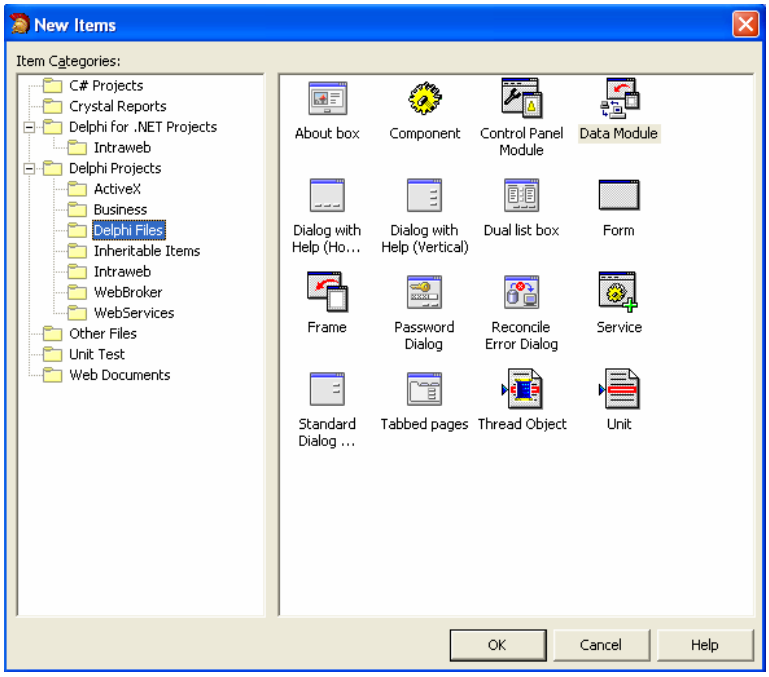
Slika58. Gradnik DBLookupComboBox po povezavi

Če bi uporabili navaden `ComboBox`, bi morali vse podatke vpisovati ročno. Kliknili bi poleg lastnosti *`Items`* v pregledovalniku lastnosti in odprlo bi se okno, v katerega bi vnesli vrstice. Če bi v bazo dodali nov podatek, bi morali programsko poskrbeti, da bi ga dodali v lastnost *`Items`* tega gradnika. V DBLookupComboBoxu pa gradnik »sam« poskrbi za to, da ima ustrezno napolnjene izbire v lastnosti *`Items`*.

Uporabili bomo še gradnik tipa `TOpenPictureDialog` (), s katerim bomo v gradnik `Image` vnesli ustrezno sliko. `OpenPictureDialog` najdemo na zavihku Dialogs. Uporablja se za izbiro slikovnih datotek, ki jih nameravamo odpreti. Njegova posebnost je okno za ogled izbrane slikovne datoteke. Omogoča prikazati datoteke vrst JPG, JPEG (slike), BMP (bitne slike), ICO (ikone), WMF in EMF (meta datoteke).

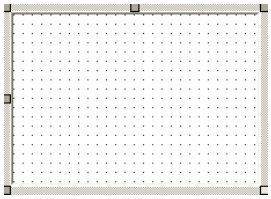
Uporaba gradnika DataModule

S pomočjo gradnika `DataModule` na enem mestu zberemo nekatere nevidne gradnike v aplikaciji. Najpogosteje so to gradniki `DataSet` in `DataSource` (`TADOConnection`, `TADOTable`, `TADOQuery`, `TADOCommand`, `TDataSource`, …). Na ta način se izognemu temu, da bi morali na različnih formah vsakič sproti nameščati in nastavljati te gradnike. Gradnik DataModule uporabimo tudi za združevanje drugih nevidnih gradnikov kot so `TTimer`, `TMainMenu`, `TSaveDialog`, … `DataModule` ne vsebuje nobene programske kode, ampak le napovedi (stavki **uses**). Vsebuje samo dve lastnosti (*Name* in *Tag*) in dva dogodka (*OnCreate* in *OnDestroy*).
Gradnik `DataModule` smo uporabili zato, da smo nanj nanesli tiste nevidne gradnike, ki jih bomo uporabljali za oblikovanje več form. Tiste nevidne gradnike, ki jih bomo uporabljali samo za eno formo, bomo nanesli kar na formo, kjer bo gradnik uporabljen.
Do gradnika `DataModule` pridemo tako, da v orodni vrstici izberemo: ***File/ New/Other/DelphiProject/DelphiFiles***. Odpre se naslednje okno:



Slika59. Odprtje gradnika DataModule

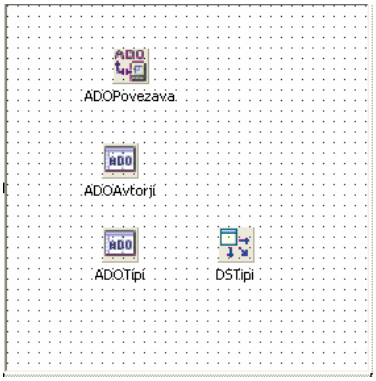
Kliknemo na ***Delphi Projects / Delphi Files***. V sosednjem oknu se nam pokažejo komponente. Med njimi izberemo `DataModule`, ter kliknemo ***OK***. Odpre se okno,



Slika60. Gradnik DataModule

kamor nanesemo naslednje gradnike:

```
ADOPovezava: TADOConnection;
ADOAvtorji: TADOTable;
ADOTipi: TADOTable;
DSTipi: TDataSource;
```



Slika61. Gradniki na DataModulu

Položaj gradnikov je povsem nepomemben, saj ne gre za vidne komponente programa. Nato s pregledovalnikom lastnosti nastavimo naslednje lastnosti:
Za gradnik `ADOPovezava`:

```
Connected = False;
```
določa status povezave. Pove, kdaj je povezava aktivna in kdaj ne, oziroma drugače, ali smo prikloplјeni na server ali ne. Privzeto je status povezave nastavljen na false.

```
LoginPrompt = False;
```
LoginPrompt je poziv za prjavo v sistem. Ko ga nastavimo na vrednost False pomeni, da za vstop v sistem ne potrebujemo prijave.

```
Name = ADOPovezava;
```
poimenujemo gradnik kot ADOPovezava

Za gradnik `ADOAvtorji` smo nastavili naslednje lastnosti:

```
ConnectionString = Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=baza\PodatkiUmetnine.mdb;Persist Security Info=False;
TableName = Avtorji;
Name = ADOAvtorji;
```

Ponovimo še enkrat: *ConnectionString* nam pove, kje imamo shranjeno bazo, ki smo jo naredili. Lastnost *TableName* nam pove, katere podatke tabele bomo uporabili – v našem primeru podatke iz tabele ***Avtorji***.

Za gradnik `ADOTipi` smo nastavili naslednje lastnosti:

```
ConnectionString = Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=baza\PodatkiUmetnine.mdb;Persist Security Info=False;
TableName = Tipi Umetnin;
Name = ADOTipi;
```

Gradniku `DSTipi` določimo naslednje lastnosti:

```
DataSet = ADOTipi;
Active = True;
TableName = TipiUmetnin;
Name = DSTipi;
```

Izvor smo povezali s tabelo, v kateri imamo trenutno shranjene podatke `ADOTipi`. Podatke bomo črpali iz tabele ***Tipi Umetnin***, gradnik smo preimenovali v `DSTipi` in izvor aktivirali, ker privzeto ni aktiven. V urejevalniku kode je nastala naslednja koda:

```
unit DM;

interface

uses
  SysUtils, Classes, DB, ADODB;

type
  TDataModule1 = class(TDataModule)
    ADOPovezava: TADOConnection;
    ADOAvtorji: TADOTable;
    ADOTipi: TADOTable;
    ADOTipiTipUmetnine: TWideStringField;
    DSTipi: TDataSource;
    ADOAvtorjiIDAvtorja: TWideStringField;
    ADOAvtorjiImeAvtorja: TWideStringField;
    ADOAvtorjiPriimekAvtorja: TWideStringField;
    ADOAvtorjiDatumRojstva: TDateTimeField;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  DataModule1: TDataModule1;

implementation

{$R *.dfm}
end.
```

Našemu gradniku `DataModule` smo pustili privzeto ime, ki ga je določil Delphi sam in to je `DataModule1`.Formo smo poimenovali **DM.pas**. Vse lastnosti, ki smo jih nastavili v pregledovalniku lastnosti, najdemo v datoteki **DM.dfm**. V tej datoteki v tem trenutku piše

```
object DataModule1: TDataModule1
  OldCreateOrder = False
  Left = 407
  Top = 253
  Height = 382
  Width = 502
  object ADOConnection1: TADOConnection
    ConnectionString =
      'Provider=Microsoft.Jet.OLEDB.4.0;Data Source= baza\
PodatkiUmetnine.mdb;Persist Security Info=False'
    LoginPrompt = False
    Mode = cmShareDenyNone
    Provider = 'Microsoft.Jet.OLEDB.4.0'
    Left = 72
    Top = 32
  end
```

Na `DataModule1` nanesemo gradnik tipa `TADOConnection`. Potrebovali bomo samo en gradnik tega tipa, ker delamo samo z eno podatkovno bazo. Nastavili smo lastnost *`ConnectionString`* in lastnost *`LoginPrompt`* nastavili na *`False`*.

```
  object Avtorji: TADOTable
    Connection = ADOConnection1
    CursorType = ctStatic
    TableName = 'Avtorji'
    Left = 72
    Top = 104
```

Na `DataModule1` postavimo tudi dva gradnika tipa `ADOTable`. Enega povežemo z gradnikom `ADOConnection` in povemo, s katero tabelo bomo delali. Ime tabele nastavimo na tabelo ***Avtorji***. V tej tabeli imamo naslednja polja: *ID Avtorja, Ime Avtorja, Priimek Avtorja in Datum Rojstva.*

```
    object AvtorjiIDAvtorja: TWideStringField
      FieldName = 'ID Avtorja'
      Size = 50
    end
    object AvtorjiImeAvtorja: TWideStringField
      FieldName = 'Ime Avtorja'
      Size = 50
    end
    object AvtorjiPriimekAvtorja: TWideStringField
      FieldName = 'Priimek Avtorja'
      Size = 50
    end
    object AvtorjiDatumRojstva: TDateTimeField
      FieldName = 'Datum Rojstva'
    end
  end
  object Tipi: TADOTable
    Connection = ADOConnection1
    CursorType = ctStatic
    TableName = 'Tipi Umetnin'
    Left = 72
    Top = 168
```

Drugi gradnik tipa `TADOTable` pa povežemo s tabelo ***Tip Umetnine***. V tej tabeli imamo eno polje z imenom *Tip Umetnine*.

```
    object TipiTipUmetnine: TWideStringField
      FieldName = 'Tip Umetnine'
      Size = 50
    end
  end
```

Potrebujemo še en gradnik tipa `TDataSource`. Povežemo ga z tabelo `ADOTipi`.

```
  object DSTipi: TDataSource
    DataSet = ADOTipi
    Left = 160
    Top = 168
  end
end
```

Forma za vstavljanje podatkov

Oglejmo si sedaj, kako bomo sestavili formo za vstavljanje podatkov. Najprej odpremo novo formo z ukazom ***File/New/VCL Forms Application***. Odpre se nova forma in tudi nov urejevalnik kode, ki se nanaša na to, na novo odprto formo. Formo smo poimenovali Vstavljanje podatkov, zato se ustrezna datoteka s kodo imenuje **VstavljanjePodatkov.pas**. Na formo postavimo ustrezne gradnike za vstavljanje podatkov in jih preimenujemo. Nato moramo gradnike iz zavihka `Data Controls` povezati z bazo, da bomo v gradnikih `DBEdit` prikazovali ustrezne podatke. V pregledovalniku lastnosti za vsak gradnik nastavimo ustrezne povezave do izvora podatkov (`DataSource`). Ker je teh povezav in lastnosti veliko, bomo to prikazali kar v tabeli. Za vse v tabeli navedene gradnike velja, da imajo naslednje lastnosti enake:

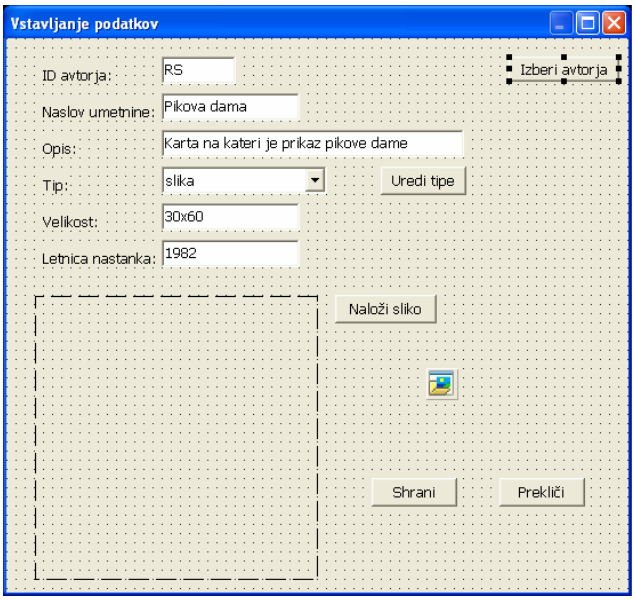
```
DataSource = frmPrikazSlik.IzvorPrikazSlike;
DataSet = ADOTipi;
Active = True;
TableName = Umetnine;
```

zato jih v tabeli sami ne bomo navajali

Ime gradnika	Pomen	Lastnosti		
		DataField	KeyField	ListSource
DBEAvtor	ID Avtorja	ID Avtorja		
DBENaslov	Naslov umetnine	Naslov		
DBEOpis	Opis	Opis		
DBCLBTip	Tip	Tip	Tip Umetnine	DataModule1.DSTipi
DBEVelikost	Velikost	Velikost Umetnine		
DBENastanek	Letnica nastanka	Letnica Nastanka		

In kaj pomenijo te lastnosti? *`DataField`* je lastnost, ki nam pove, katero polje iz tabele (v našem primeru iz tabele ***Umetnine***), bomo prikazali v gradniku za prikaz podatkov. Lastnost *`KeyField`* predstavlja polje iz druge tabele, s katerim se povežemo, da lahko prikažemo te podatke v gradniku `DBLookupComboBox`, *`ListSource`* pa je naš izvor, ki določa, od kje črpamo podatke. V našem primeru so podatki shranjeni v gradniku `DSTipi`, ta gradnik pa leži na `DataModule1`.

Izgled forme je naslednji:



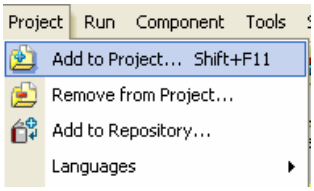
Slika62. Forma za vstavljanje podatkov

Kot vnosna polja smo v vseh primerih, razen za vnos tipa, uporabili gradnik tipa `TDBEdit`. Kot vnosno polje za tip smo uporabili gradnik `DBLookupComboBox`. Za napise poleg vnosnih polj smo uporabili gradnike tipa `TLabel`. Za prikaz slike smo uporabili gradnik tipa `TImage`. Poimenovali smo ga `Slika`. Za izbiro datoteke s sliko in njen prenos v gradnik `Image` smo uporabili gradnik `OpenPictureDialog`. Avtorja bomo izbrali s klikom na gradnik `SpeedButton`, vsi ostali gumbi pa so navadni gumbi `Button`.

Opišimo, kaj želimo, da se na tej formi dogaja:

- Podatke, ki jih bomo vnesli, kot so *Naslov umetnine, Opis, Velikost in Letnico nastanka*, bomo vpisovali direktno. *ID avtorja* bomo izbrali s klikom na gumb ***Izberi avtorja***, tipe pa s klikom na gumb ***Uredi tipe***. Ker ne želimo, da bi na tej formi dodajali novega avtorja (oziroma njegove inicialke), moramo preprečiti vpisovanje podatkov v gradnik `DBEAvtor`. To dosežemo tako, da v pregledovalniku lastnosti nastavimo `DBEAvtor.ReadOnly := True.` Tako bomo vrednost Text tega polja lahko le brali, ne pa tudi popravljali.
- *Tip umetnine* bomo izbrali iz spustnega seznama, tako da bomo kliknili v pravokotniček poleg seznama in izbrali eno od možnosti, ki so nam podane. Vse možne izbire se napolnijo avtomatično iz baze, kot smo povedali že prej. Če vpisujemo podatke o umetnini, katerega tipa še nimamo v ustreznem spustnem seznamu, moramo ta tip dodati. To dosežemo tako, da kliknemo na gumb ***Uredi tipe***. Odprla se bo nova forma, kjer s pomočjo gradnika `DBNavigator` dodamo nov tip umetnine. Formo bomo izoblikovali v nadaljevanju.
- Ustrezno datoteko, kjer je slika naše umetnine, bomo izbrali tako, da bomo kliknili na gumb ***Naloži sliko***, ki bo prikazal pogovorno okno za izbiro datoteke.
- Polje *ID avtorja* bomo izpolnili s pomočjo klika na gradnik `SpeedButton` z napisom ***Izberi avtorja***. Takrat se bo odprlo novo okno, kjer bomo imeli podatke o avtorjih. Izbrali bomo enega avtorja in s tem v gradnik `DBEAvtor` zapisali *ID Avtorja*. Tudi to formo bomo izoblikovali v nadaljevanju.
- Ko imamo vse podatke vnesene, kliknemo na gumb z napisom ***Shrani***. Ta povzroči, da se vsi podatki zapišejo v našo podatkovno bazo. Spremembe so vidne tudi v posameznih tabelah.
- Če smo se premislili in podatkov ne želimo vnesti, vse skupaj prekličemo s klikom na gumb ***Prekliči***.

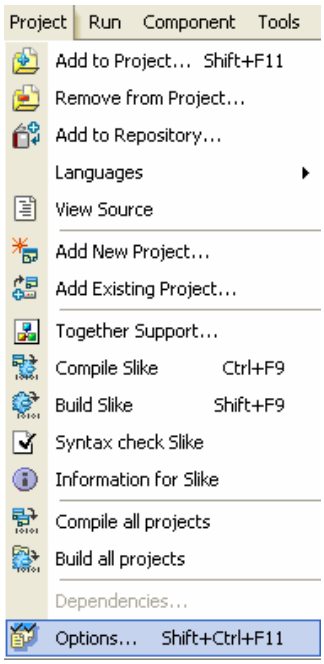
Tako, pripravljeno imamo vnosno formo. Želimo jo povezati z osnovno formo ***Prikaz slike***. To naredimo tako, da v menijski vrstici izberemo
Project/Add to Project:



Slika63. Sestava projekta v celoto

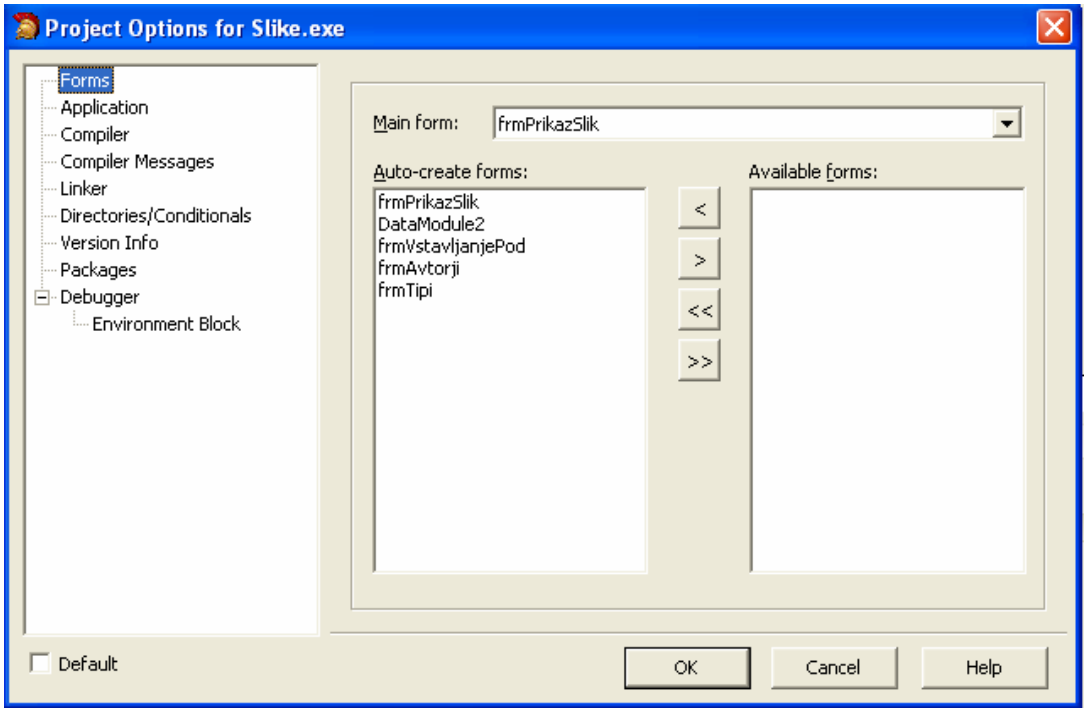
Na disku poiščemo datoteko **VstavljanjePod.pas** in kliknemo na gumb ***Odpri***. Tako bomo naredili tudi s preostalimi formami. Ker pa imamo sedaj dve formi, je potrebno določiti glavno formo. To naredimo tako, da kliknemo na

Project/Options:



Slika64. Nastavitev glavne forme

Odpre se okno:



Slika65. Za glavno formo izberemo frmPrikazSlik

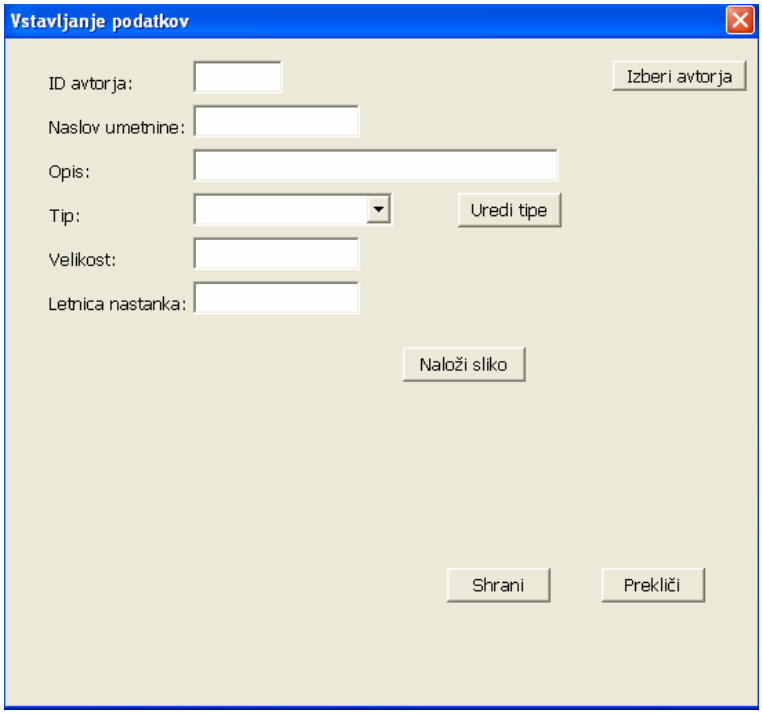
V okno poleg napisa ***Main form*** vnesemo formo, ki jo želimo imeti kot glavno formo – to je forma `frmPrikazSlik`.
Hočemo, da se pravkar narejena forma za vnos podatkov prikaže, ko po zagonu programa na formi ***Prikaz slik*** kliknemo na gumb ***Vstavi podatke***. Zato se preselimo na formo ***Prikaz slike***,

dvakrat kliknemo na gumb ***Vstavi podatke***, da zagledamo urejevalnik kode. Vanj vnesemo naslednjo kodo:

```
procedure TfrmPrikazSlik.btn_vstaviClick(Sender: TObject);
begin
  frmVstavljanjePod.Caption := 'Vstavljanje podatkov';
  frmVstavljanjePod.btn_shrani.Caption := 'Shrani';
  ADOPrikaz.Insert;
  if frmVstavljanjePod.ShowModal = mrOk then
  { če smo okno za vstavljanje zaprli s klikom na
    gumb Shrani (ki ima modalni rezultat mrOk) }
  begin
    ADOPrikaz.Post; {zapišemo podatke v tabelo}
    PrikaziSliko(ADOPrikazSlikaUmetnine, ADOSlika);
  end
  else
    ADOPrikaz.Cancel; {nobenih sprememb v bazi}
end;
```

Poglejmo, kaj ta koda počne. Najprej formi za vstavljanje določimo, da je napis na oknu ***"Vstavljanje podatkov".*** Napis smo oblikovali programsko, kajti to formo bomo uporabljali tudi za urejanje podatkov. Takrat bo napis drugačen. Spremenili smo tudi napis na gumbu `btn_shrani`. Nato z `ADOPrikaz.Insert` tabelo `AdoPrikaz` (tabelo, ki je povezana z bazo) postavimo v stanje vstavljanja. Novo formo prikažemo modalno z `frmVstavljanjePod.ShowModal`. Če je forma odprta modalno, pomeni, da ne moremo preklapljati med formami. Če želimo delati na kakšni drugi formi, moramo najprej modalno odprto formo zapreti. Tukaj je seveda nujno, da formo za vstavljanje prikažemo modalno, saj dokler z vstavljanjem podatkov v bazo ne končamo, forma, ki prikazuje podatke v bazi, ne more delati pravilno. In če na formi `frmVstavljanjePod` kliknemo na gumb, ki ima modalni rezultat enak *mrOk* (`frmVstavljanjePod.ShowModal = mrOk`), se sprememba zapiše v tabelo `ADOPrikaz` (`ADOPrikaz.Post`). Nato še gradniku `Slika` prikažemo sliko iz polja *ADOPrikazSlikaUmetnine*. Če okna nismo zaprli tako, da bi kliknili na gumb z modalnim rezultatom *mrOK*, se tabela ne spremeni.

Po zagonu programa se najprej odpre forma ***Prikaz Slik***. Ko kliknemo na gumb ***Vstavi podatke***, se odpre forma ***Vstavljanje podatkov***.



Slika66. Forma za vstavljanje podatkov po zagonu programa

Oblikovati moramo še formo za izbiro avtorja in formo za dodajanje novih tipov umetnin. Najprej sestavimo novo formo ***Avtorji***, ki jo bomo dobili ob kliku na gumb ***Izberi avtorja***. Nanjo postavimo naslednje gradnike:

- `DBNavigator`, ki ga uporabljamo za premikanje po tabeli, za dodajanje novega avtorja in za brisanje avtorja,
- `DBGrid`, ki ga uporabljamo za prikaz podatkov,
- `DataSource`, ki ga uporabljamo za prenos podatkov iz podatkovne baze do gradnika `DBGrid`, ki podatke prikaže,
- `Panel` za razdelitev forme in
- dva gumba `Button`. Na enega zapišemo ***Izberi***, s katerim izberemo želenega avtorja, na drugega pa ***Prekliči***, s katerim prekinemo izbiro avtorja.

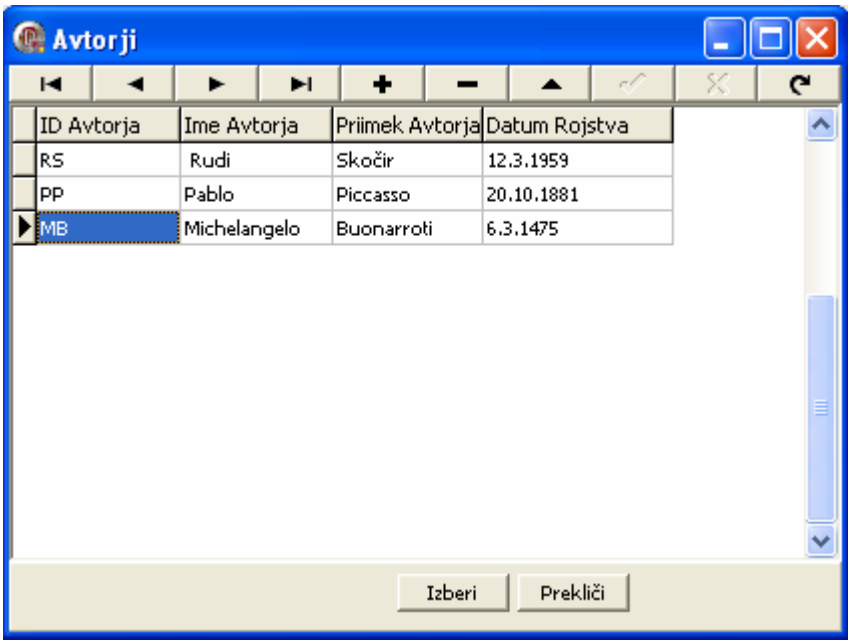
`Panel` je gradnik tipa `TPanel`. `Panel` uporabljamo zato, da združujemo gradnike v celoto – ko premikamo `Panel`, se sočasno premikajo vsi gradniki na `Panelu`. Če nastavimo, da je določen gradnik vezan na desni rob `Panela`, se ob raztegovanju `Panela` premakne tudi ta gradnik. Ko `Panel` zbrišemo, zbrišemo vse vsebovane gradnike, ... Skratka `Panel` je nekakšen »združevalec« gradnikov.

V pregledovalniku lastnosti nastavimo

```
btn_izberi.ModalResult = mrOk
btn_preklici.ModalResult = mrCancel
DataSource.Name = DSAvtorji
DSAvtorji.DataSet = DataModule1.ADOAvtorji
DBNavigator1.DataSource = DSAvtorji.
```

Gumboma nastavimo funkcijo delovanja. Gumbu `btn_izberi` njegov modalni rezultat nastavimo na *`mrOk`*, gumbu `btn_preklici` pa na *`mrCancel`*. V lastnost forme *`ShowModal`* se namreč zapiše, ali je bil kliknjen tisti gumb, ki »nosi« vrednost *`mrOk`* (okrajšava `mr` izvira iz lastnosti

`ModalResult`), ali tisti, ki »nosi« vrednost `mrCancel`. Ko okno zapremo, se zato ve, na kakšen način smo ga zaprli.
Izvoru podatkov spremenimo ime v `DSAvtorji` in izvore povežemo s tabelo, v kateri imamo podatke o avtorjih.
Poglejmo, kako se celotna zadeva obnaša. Denimo, da smo pognali program. Odpre se osnovna forma za prikaz podatkov. Ker želimo vnesti podatke o novi umetnini, kliknemo na gumb za vstavljanje podatkov. Odpre se nova forma, kamor bomo vnesli podatke o umetnini. *ID Avtorja* izberemo tako, da kliknemo na gumb ***Izberi avtorja***. Spet se odpre nova forma ***Avtorji***, kjer se prikaže tabela avtorjev.



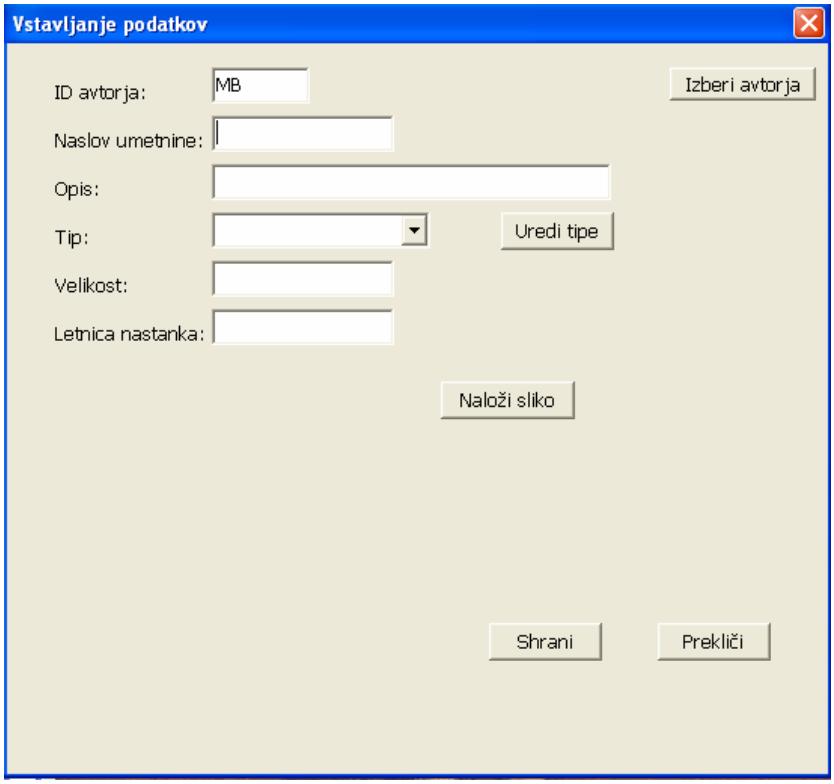
Slika67. Izbira Avtorja

Kliknemo na vrstico z ustreznim avtorjem – v našem primeru kliknemo na MB in kliknemo gumb ***Izberi***. S tem se vrnemo nazaj na formo Vstavljanje podatkov. V polju *ID Avtorja* je vpisana ustrezna oznaka. Gumb ***Izberi*** ima modalni rezultat `mrOK`. Zato bomo v formi, od koder smo to formo odprli, s pomočjo lastnosti `ShowModal` lahko preverili, če je bil za zapiranje okna kliknjen prav ta gumb. Če kliknemo na gumb ***Prekliči***, se prav tako vrnemo na formo ***Vstavljanje podatkov.*** Vendar polje *ID avtorja* ostane prazno.
Procedura, ki nam je izbiro avtorja omogočila, se seveda skriva v kodi, ki se izvede ob dogodku *Click* na gumb `sb_izberiAvtorja` in je naslednja:

```
procedure TfrmVstavljanjePod.sb_izberiAvtorjaClick
  (Sender: TObject);
begin
  if frmAvtorji.ShowModal = mrOk then
    { forma Avtorji je bila zaprta s klikom na OK }
    DBEAvtor.Field.AsString :=
    DataModule1.ADOAvtorji.FieldByName('ID Avtorja').AsString;
   { v DBEAvtor se zapiše podatek iz ID Avtorja }
end;
```

Razložimo jo. Ko kliknemo na formi `frmVstavljanjePod` na gumb `sb_izberiAvtorja`, se najprej modalno prikaže forma `frmAvtorji`. Če kliknemo na gumb, ki ima modalni rezultat enak `mrOk`, se v gradnik `DBEAvtor` zapiše podatek, ki je označen s kurzorjem v tabeli ***ADOAvtorji***, v polju z imenom *ID Avtorja*.

Izgled forme je sedaj:

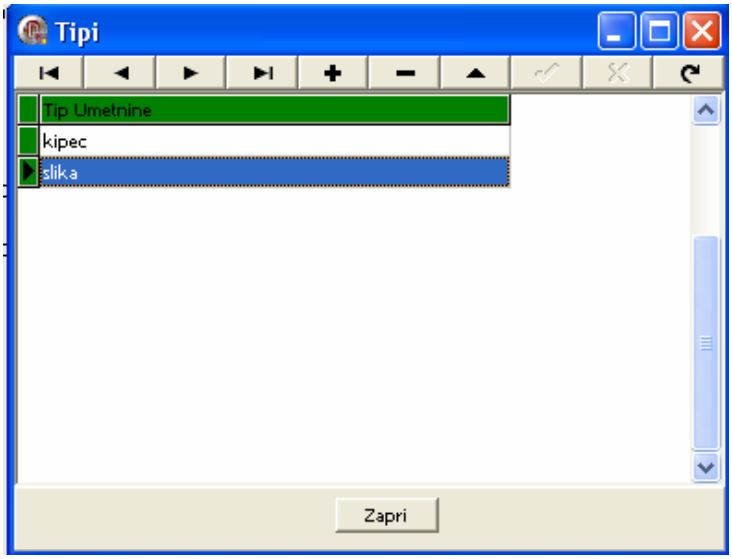


Slika68. Izbrali smo avtorja

Avtorja smo izbrali, vstaviti pa je potrebno še preostale podatke. *Naslov umetnine* in *Opis* vnašamo ročno preko tipkovnice. Tip umetnine izberemo iz izbirnega seznama. Če tipa umetnine ne najdemo v izbirnem seznamu, kliknemo na gumb `btn_urediTipe`, kjer preko gradnika `DBNavigator` vstavimo nov tip umetnine. Za gumb `btn_urediTipe` zapišemo naslednjo kodo:

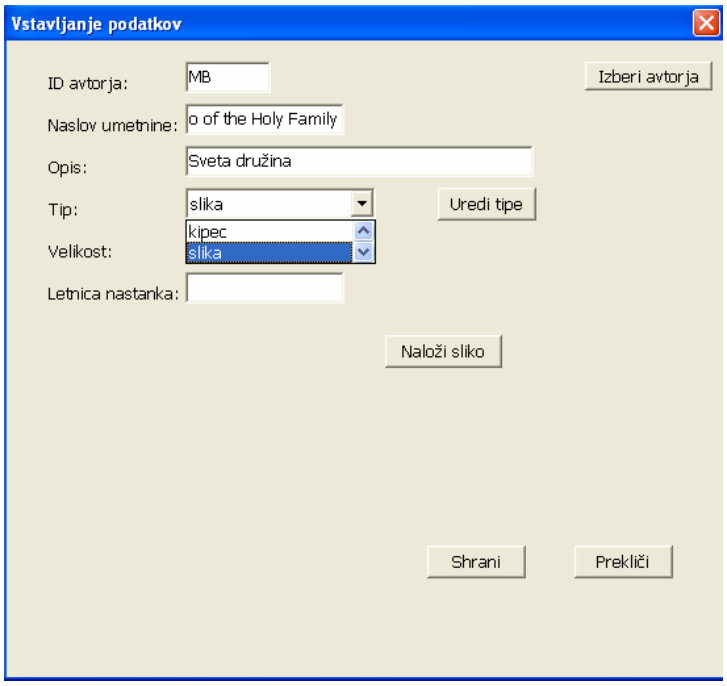
```
procedure TfrmVstavljanjePod.btn_urediTipeClick(Sender: TObject);
begin
  frmTipi.ShowModal; // modalno odpremo formo za vnos tipov
end;
```

Ta koda nam modalno prikaže formo `frmTipi`, kar pomeni, da moramo pred nadaljevanjem programa to formo obvezno zapreti. Ker bo ta forma taka, da bo zapiranje možno le preko enega gumba, nam modalnega rezultata ne bo potrebno preverjati. V vsakem primeru bo v oknu izbran neki tip umetnine, ki ga bomo s tem prenesli v ustrezno polje.
Kliknemo na gumb z oznako '+' in v prazno polje vpišemo nov tip.



Slika69. Nov tip umetnine lahko vstavimo preko te forme

Iz seznama izberemo tip slika. S klikom na gumb ***Zapri*** se vrnemo v prejšnje okno, kjer je v tipu umetnine že vpisana ustrezna (izbrana) vrednost.



Slika70. Izbira tipa umetnine

Vnesemo še *Velikost* umetnine 150x113 in *Letnico nastanka* 1540. Tudi ta dva podatka vnesemo preko tipkovnice. V gradnik `Image` moramo naložiti še sliko. To naj se zgodi ob kliku na gumb z napisom ***Naloži sliko***. Opišimo, kaj vse moramo v tej metodi postoriti:

- izbrati datoteko na disku, kjer je slika umetnine,
- v ustrezno polje v bazi shraniti vsebino te datoteke,
- poskrbeti za prikaz slike na ustrezni formi.

Najprej si oglejmo kodo, ki bo shranila podatke iz grafične datoteke v ustrezno polje tabele. Podprogram bomo poimenovali *ShraniSlikoVTabelo*. Koda tega podprograma je naslednja

```
procedure ShraniSlikoVTabelo(Tabela: TADOTable;
          Polje: TBlobField; PotSlike: String);
var
   fs: TFileStream;
begin
   fs := TFileStream.Create(PotSlike, fmOpenRead);
   try
      Tabela.Edit; {postavimo se v način urejanja}
      Polje.LoadFromStream(fs); {napolnimo ustrezno polje tabele}
      Tabela.Post; {zapišemo podatek v bazo}
   finally
      fs.Free;
   end;
end;
```

Procedura dobi za parametre spremenljivko tipa `TADOTable`, spremenljivko tipa `TBlobField` in spremenljivko tipa `String`. Prva, tipa `TADOTable`, pove, v katero tabelo bomo podatke shranjevali. Podatke bomo shranili v polje, ki je drugi parameter (in mora biti seveda tipa `TBlobField`, saj smo povedali, da vsebino grafičnih datotek hranimo kot nekaj vrste BLOB). Zadnji parameter nam pove pot, po kateri pridemo do shranjene slike umetnine. V kodi najprej ustvarimo objekt fs. To je podatkovni tok, ki predstavlja vsebino datoteke na disku s potjo `PotSlike`. Ker bomo tok le brali in ga ne bomo spreminjali, ga odpremo le za branje (uporabimo konstanto *`fmOpenRead`*). Tabelo postavimo v stanje urejanja. Polje napolnimo iz podatkovnega toka fs. Na koncu še v tabelo zapišemo spremembo in na koncu sprostimo objekt fs, ki je bil ustvarjen na začetku.
Sedaj navedimo še kodo, ki se zgodi ob kliku na gumb `btn_naloziSliko`:

```
procedure TfrmVstavljanjePod.btn_naloziSlikoClick(
          Sender: TObject);
begin
  if OPD_naloziSliko.Execute then {odpremo pogovorno okno in
     poiščemo ustrezno grafično datoteko}
  begin
    ShraniSlikoVTabelo(frmPrikazSlik.ADOPrikaz,
          frmPrikazSlik.ADOPrikazSlikaUmetnine,
          OPD_naloziSliko.FileName);
    PrikaziSliko(frmPrikazSlik.ADOPrikazSlikaUmetnine,
                 Slika); {sliko prikažemo v gradniku Slika}
  end;
end;
```

Razložimo kodo. Ko izvajamo metodo *`Execute`* nad gradnikom tipa `TOpenPictureDialog` (kot je naš gradnik z imenom `OPD_naloziSliko`), ta povzroči, da zagledamo pogovorno okno, v katerem izberemo ustrezno grafično datoteko. Metoda *`Execute`* ima rezultat tipa `Boolean`. *`Execute`* vrne rezultat *`True`* v primeru, če uporabnik klikne na gumb ***OK*** ali če dvakrat klikne na ime datoteke in jo s tem odpre, ali pa če pritisne gumb ***Enter*** na tipkovnici. Rezultat *`False`* vrne v primeru, ko uporabnik klikne na gumb ***Cancel***, pritisne gumb ***Esc*** na tipkovnici, ali pa v primeru, da zapre pogovorno okno. V našem primeru, ko metoda vrne rezultat *`True`*, sliko shranimo v tabelo s klicem podprograma *ShraniSlikoVTabelo*, kjer uporabimo parametre

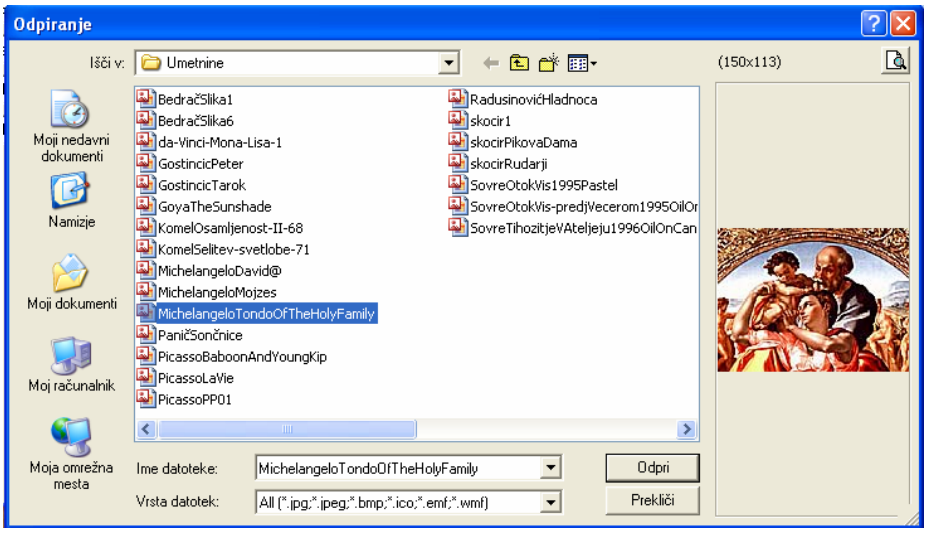
frmPrikazSlik.ADOPrikaz	ime tabele
frmPrikazSlik.ADOPrikazSlikaUmetnine	polje, v katero naj shrani vsebino grafične datoteke
OPD_naloziSliko.FileName	pot, kjer je shranjena slika

Kako pa smo prišli do tiste poti? Ko smo z `OPD_naloziSliko.Execute` odprli pogovorno okno za vnos slike in ko pri izbiri datoteke kliknemo na gumb ***OK***, se v lastnost *`FileName`* zapiše celotna pot in ime datoteke.

Nato s proceduro *PrikaziSliko(frmPrikazSlik.ADOPrikazSlikaUmetnine, Slika)* na vnosni formi prikažemo sliko, kjer je

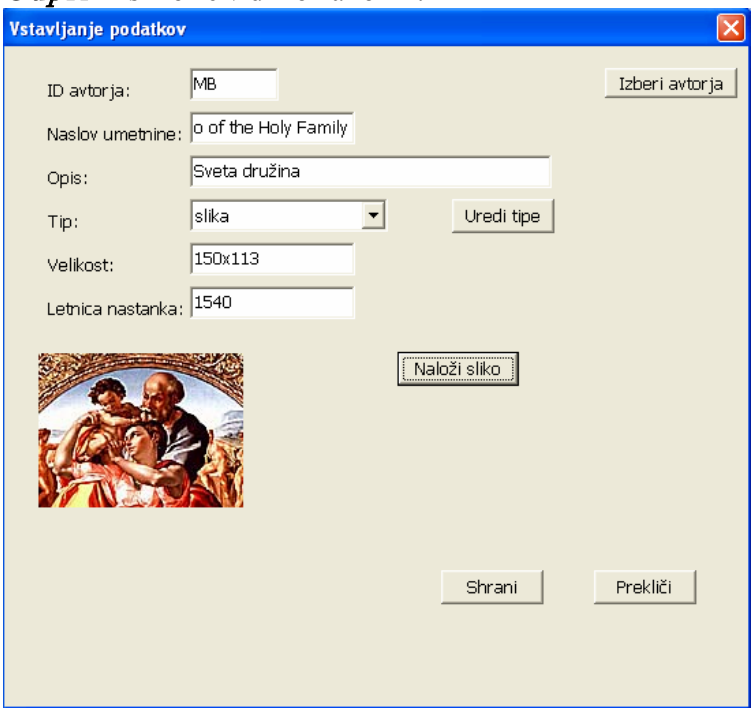
frmPrikazSlik.ADOPrikazSlikaUmetnine	polje, v katerem je shranjena slika
Slika	gradnik tipa `TImage`, ki naj prikaže sliko

Poglejmo, kaj se torej zgodi ob kliku na gumb `btn_naloziSliko`. Zagledamo pogovorno okno, kjer poiščemo na disku datoteko, kjer imamo shranjeno sliko ustrezne umetnine:



Slika71. Vstavljanje slike umetnine

Kliknemo na gumb ***Odpri*** in sliko že vidimo na formi.

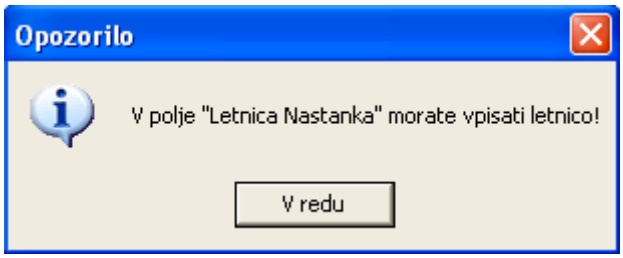


Slika72. Forma po vstavitvi vseh podatkov

Preden kliknemo na gumb ***Shrani***, moramo preveriti, če smo v polje *Letnica nastanka* res vpisali število. To bomo preverili s funkcijo `PreveriVnos`.

```
function TfrmVstavljanjePod.PreveriVnos (niz: String): Boolean;
var i: Integer;
begin
   i := StrToIntDef (niz, -1); {niz spremenimo v število in če
      se to uspešno pretvori (i ni -1), je OK}
   if (i = -1) or (i < 1000) or (i > 2100) then begin
      { ker pretvorba ni bila uspešna, izpišemo opozorilo }
      { enako, če je leto manjše kot 1000 ali večje kot 2100 }
      MessageBox(0,
        'V polje "Letnica Nastanka" morate vpisati letnico!',
        'Opozorilo', MB_OK + MB_ICONINFORMATION);
      Result = False; // ni bilo v redu
   end else
     Result = True; // v spremenljivki niz je res število!
end;
```

Funkcija `PreveriVnos` nam pregleda niz, ki je tipa besedilo. Če ta niz lahko pretvori v število, potem smo res vpisali število. Preverimo še, če je to število med 1000 in 2100, kar naj bi predstavljalo smisleno letnico nastanka umetnine.
Ko kliknemo na gumb ***Shrani,*** se podatki vpišejo v podatkovno bazo le, če smo jih pravilno vnesli. Če pa v polje *Letnica nastanka* nismo vnesli številke, se izpiše naslednje opozorilo:



Slika73. Opozorilo ob napačni vstavitvi podatka

Če kliknemo na gumb ***Prekliči***, se forma zapre, v tabeli pa ni sprememb. Za gumb `btn_shrani` nastavimo modalni rezultat na *mrOk*. Podatke dejansko shrani metoda klik na gumb `btn_izberiAvtorja`, ko zve, da se je tole okno zaprlo z modalnim rezultatom enakim *mrOK*.
Za gumb `btn_shrani` smo zapisali naslednjo proceduro:

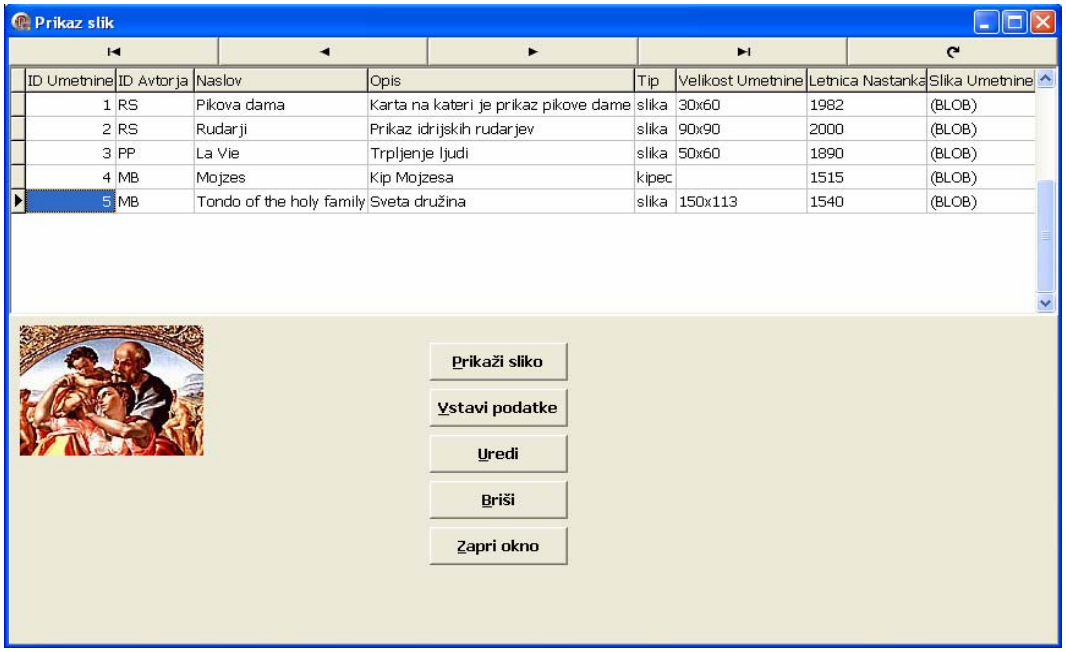
```
procedure TfrmVstavljanjePod.btn_shraniClick(Sender: TObject);
begin
   if (PreveriVnos(DBENastanek.Text) = True) then
      ModalResult := mrOk;
end;
```

Ta koda najprej preveri, če smo vpisali v polje *Letnica nastanka* res podatek v obliki številke, nato pa, če je temu tako, zapre formo z modalnim rezultatom mrOk. To povzroči, da v nadrejeni formi podatke zapišemo tudi v bazo.
Za gumb z napisom ***Prekliči*** v pregledovalniku lastnosti nastavimo

```
ModalResult := mrCancel;
```

Ponovimo še enkrat, da modalni rezultat *mrOk* pomeni, da je uporabnik zaprl formo s klikom na gumb ***OK***, modalni rezultat *mrCancel* pa pomeni, da je uporabnik zaprl formo s klikom na gumb ***Cancel***. Kako pa modalni rezultat uporabimo, je naša stvar. Seveda lahko pričakujemo, da bomo modalni

rezultat `mrOK` uporabili za to, da bomo signalizirali, da se je okno zaprlo s potrditvijo sprememb, modalni rezultat `mrCancel` pa takrat, ko uporabnik noče sprememb.
Ko kliknemo na gumb ***Shrani***, se forma ***Vstavljanje podatkov*** zapre in podatki se zapišejo v podatkovno bazo, kar je vidno na naslednji sliki:



Slika74. Po zaprtju forme za vstavljanje podatkov

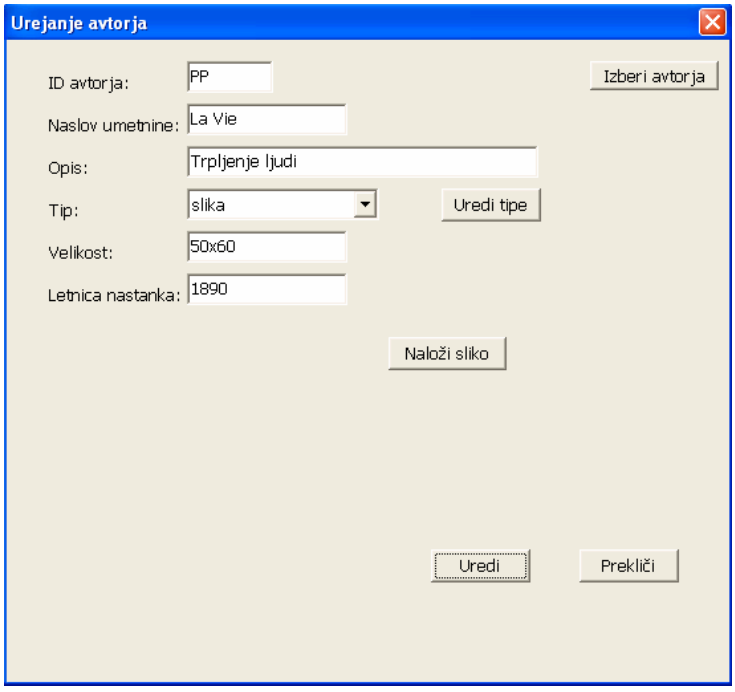
Urejanje podatkov v podatkovni bazi

Za urejanje podatkov v podatkovni bazi bomo uporabili kar isto formo kot za vstavljanje podatkov. Spremenili bomo samo naslov forme. Razlika bo še v tem, da bomo tabelo `ADOPrikaz` postavili v stanje urejanja. To bo povzročilo, da bomo v ustreznih gradnikih že imeli vnešene trenutne vrednosti polj.

Na formi ***Prikaz slik*** dodamo nov gumb `btn_uredi`. Pod ta gumb pa zapišemo naslednjo kodo:

```
procedure TfrmPrikazSlik.btn_urediClick(Sender: TObject);
begin
  frmVstavljanjePod.Caption := 'Urejanje avtorja';
     {spremenimo naslov forme}
  frmVstavljanjePod.btn_shrani.Caption := 'Uredi';
     {spremenimo napis na gumbu}
  ADOPrikaz.Edit; {tabelo postavimo v stanje urejanja}
  if frmVstavljanjePod.ShowModal = mrOk then
  begin
    ADOPrikaz.Post; {zapiše spremembo}
    PrikaziSliko(ADOPrikazSlikaUmetnine, ADOSlika);
        {prikažemo sliko s klicem procedure}
  end
  else
    ADOPrikaz.Cancel;
end;
```

Podprograma ni potrebno razlagati, saj je praktično enak, kot smo ga zapisali za gumb `btn_vstavi`. Spremenili smo le naslov forme v ***Urejanje avtorja*** in napis na gumbu `btn_shrani` v ***Uredi***. Tabelo postavimo v stanje urejanja `ADOPrikaz.Edit`, prejšnjo pa smo postavili v stanje vstavljanja.
Ko kliknemo na gumb z napisom ***Uredi***, se odpre forma:

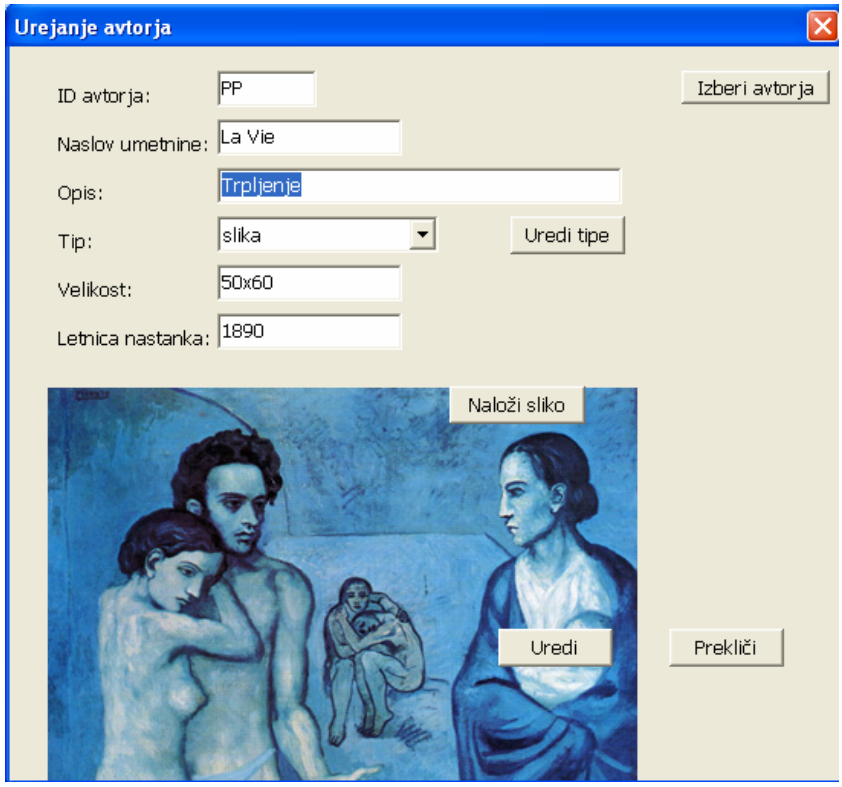


Slika75. Forma za urejanje podatkov

Prikažejo se podatki iz aktivnega zapisa (vrstice, v kateri je kurzor). Seveda lahko popravimo en ali več podatkov o umetnini. Radi bi videli, da se nam ob odprtju te forme prikaže tudi slika, ki pripada tem podatkom. To naredimo z naslednjo proceduro:

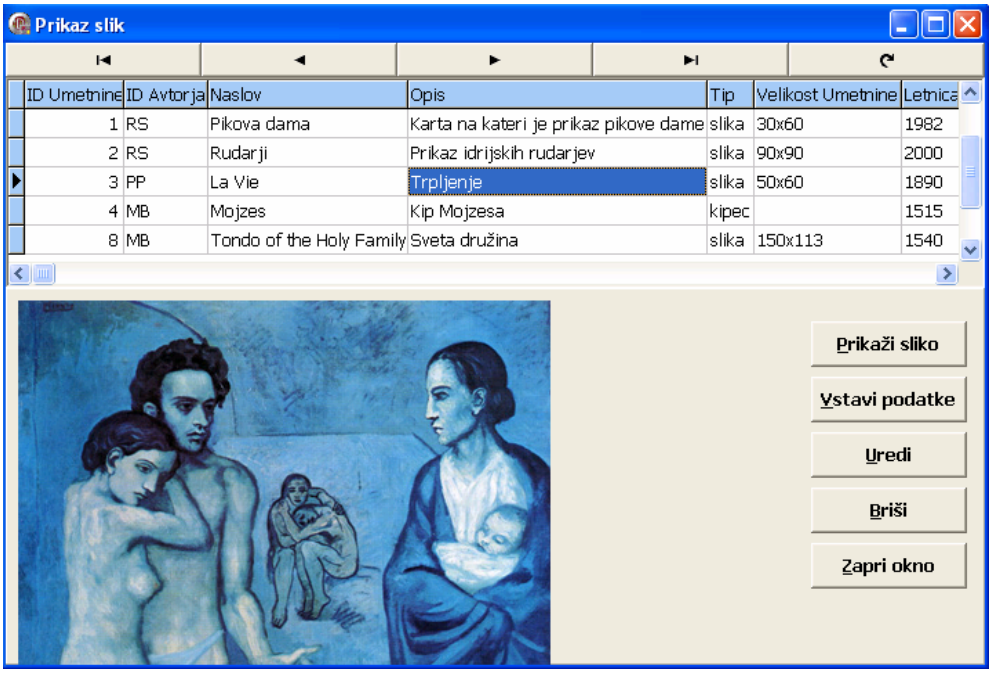
```
procedure TfrmVstavljanjePod.FormShow(Sender: TObject);
begin
  if frmPrikazSlik.ADOPrikaz.State = dsEdit then
     {če je tabela v stanju urejanja}
    PrikaziSliko(frmPrikazSlik.ADOPrikazSlikaUmetnine, Slika)
  else
    Slika.Picture.Assign(nil);
end;
```

Najprej pogledamo, če je `ADOPrikaz` v stanju urejanja. Če je, prikažemo sliko s proceduro `PrikaziSliko`, ki ima za prvi parameter polje, v katerem je shranjena slika, kot drugi parameter pa ima gradnik, v katerem se bo slika prikazala. To je gradnik `Slika`, tipa `TADOTable`. Če tabela ni v stanju urejanja ali pa če program slike ne najde, se v gradniku `Slika` prikaže »prazna slika«, oziroma slike ni.



Slika76. Sprememba podatka

Ko kliknemo na gumb ***Uredi***, je dogajanje tako, kot je bilo opisano v prejšnjem razdelku (saj gre za isto formo). Forma se zapre in vsi podatki se bodo spremenili tudi v podatkovni bazi, kar je vidno na naslednji sliki.



Slika77. Razlika je vidna tudi v tabeli

Brisanje podatkov iz podatkovne baze

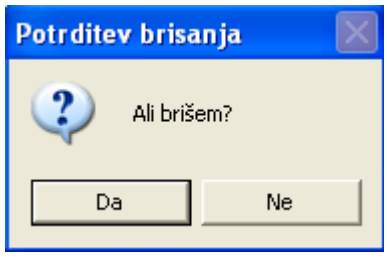
Za brisanje podatkov iz podatkovne baze nismo naredili nobene forme, saj ni potrebna. Pod gumb ***Briši*** smo zapisali naslednjo kodo:

```
procedure TfrmPrikazSlik.btn_brisiClick(Sender: TObject);
begin
  if MessageBox('Ali brišem?', 'Potrditev brisanja',
     MB_YESNO + MB_ICONQUESTION) = ID_YES
    { Če v sporočilnem oknu kliknemo na Yes/Da }
  then
    ADOPrikaz.Delete; {izbriše vrstico v bazi}
end;
```

MessageBox je funkcija, ki prikaže pogovorno okno, opremljeno z gumbi. Vrne podatek o tem, na kateri gumb je kliknil uporabnik. Če v našem primeru kliknemo na gumb ***Da***, nam izbriše vrstico, v kateri je kurzor (torej ustrezen zapis iz baze).

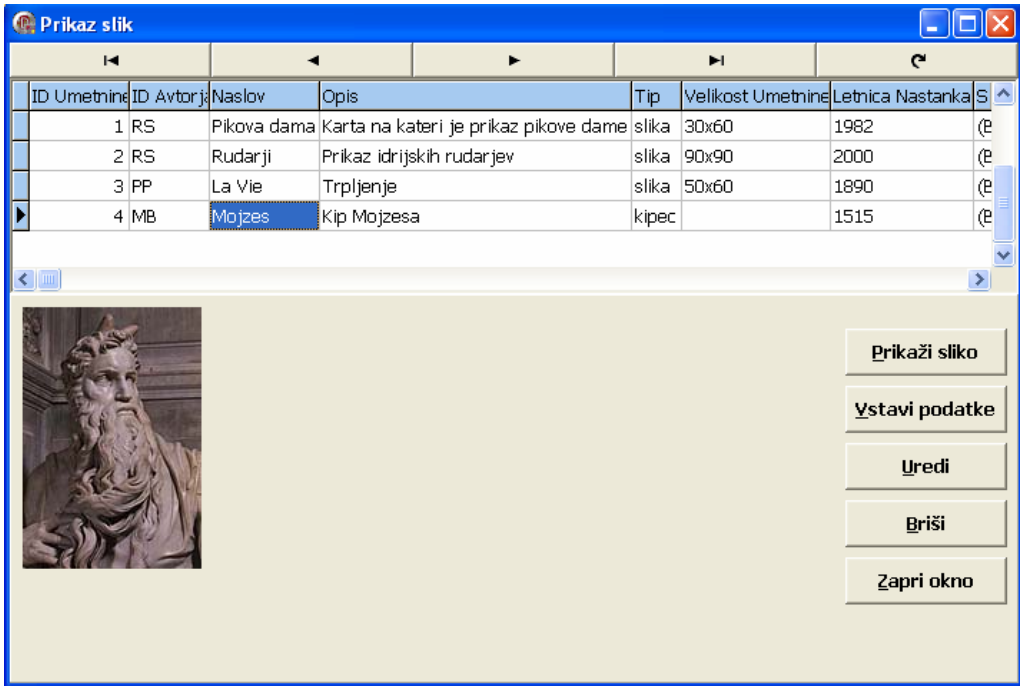
'Ali brišem?'	kaj želimo, da nas vpraša
'Potrditev brisanja'	naslov okna
MB_YESNO	kakšne gumbe nam ponudi na izbiro – v tem primeru nam ponudi gumba Yes in No (oziroma Da / Ne)
MB_ICONQUESTION	poleg vprašanja se izriše ikona v obliki ?

Vprašanje zgleda takole:



Slika78. Obrazec za potrditev brisanja podatkov iz baze

Ko kliknemo ***Da***, se okno z vprašanjem zapre, iz baze pa se izbriše celoten zapis, ki je bil pred tem aktiven. Tabela je naslednja:



Slika79. Končni izgled forme

Če kliknemo ***Ne***, se okno zapre, v tabeli pa se ne spremeni nič.

Zaključek

Pri delu na tej diplomski nalogi sem spoznala dva programa: Delphi in Microsoft Access. V sami nalogi sem najprej opisala Microsoft Access, v katerem smo najprej naredili podatkovno bazo PodatkiUmetnine.mdb. V njej smo naredili tri tabele (Avtorji, Tipi Umetnin in Umetnine), da smo kasneje z njimi operirali. V tabeli Avtorji hranimo podatke o avtorjih umetnin, ime in priimek in njihove rojstne podatke. V tabeli Tipi Umetnin hranimo podatke o tipih. V tabeli Umetnine pa hranimo podatke o posamezni umetnini, naslov, opis, velikost, tip, letnico nastanka in sliko umetnine. Te tabele smo povezali v relacije.Nato smo spoznali še programski jezik Delphi in ga na kratko opisali, in začeli sestavljati kodo, ki nam je prikazovala podatke iz podatkovne baze v Delphiju. Sestavili smo tudi program, ki nam vstavlja podatke v podatkovno bazo, ureja in briše podatke.
Ta program pa seveda ni končan. Lahko bi ga spreminjali in dograjevali brez konca. Z nekoliko več truda in znanja bi lahko podatke urejali tudi po vrstnem redu, po velikosti, abecednem redu, po letnicah, jih filtrirali in podobno.
Pri delu na diplomski nalogi sem se naučila predvsem dela s podatkovnimi bazami, osnov proigramiranja v Delphiju. Pomembno je tudi, da sem spoznala, da je zelo koristno uporabljati vgrajeno pomoč v programih Delphi in Microsoft Access. Čeprav je bilo delo zanimivo, mi je izdelava programa povzročila tudi veliko problemov. Nasploh mi je na prvi pogled enostaven problem povzročal kar nekaj preglavic. No s kar nekaj truda sem večino ovir uspešno premagala in se pri tem kar nekaj naučila.

Razviti program in podatkovna baza sta na priloženi zgoščenki.

Literatura:

1. CALVERT, CHARLES, *Delphi Unleashed*, USA, Macmillan Computer Publishing, 1995
2. CANTU, MARCO, *Mastering Borland Delphi 2005*, USA, Sybex, 2005
3. HRIBAR, PETER, *Spoznajmo Delphi*, Nova Gorica, Flamingo založba, 1999
4. KOSTREVEC, LJUBOMIR, *Uvod v programiranje v programskem jeziku Delphi*, Ljubljana, Pasadena, 2000
5. MATCHO, JON & FAULKNER, DAVID, *Special Edition Using Delphi*, Indianapolis, Que Corporation, 1995
6. MRHAR, PETER, *Uvod v SQL*, Nova Gorica, Flamingo založba, 2002
7. PAHOR, DAVID, *Leksikon računalništva in informatike*, Ljubljana, Pasadena, 2002

Internetne strani:

8. AMMARA, *Pictures in Access Forms & Reports – OLE Object Photo & Image Problems & Solutions*, 18.10.2005, 11.7.2006, <http://www.ammara.com/articles/accesspictureole.html>
9. BAUER, ANDREJ, *Arhiv 2004/2005: Računalništvo 1 (UNI)*, 15.11.2005 <http://haka.fmf.uni-lj.si/uni-racunalnistvo-1/arhiv-2004/lekcija21/index.html>
10. CHAPPLE, MIKE, *Databases, Microsoft ActiveX Data Objects (ADO),* 7.12.2005 <http://databases.about.com/od/development/a/ado.htm>
11. CZERNIK, JAMIE, *Jamie's Software - Access Articles Handling Images with Microsoft® Access*, 11.7.2006, < http://www.jamiessoftware.tk/articles/handlingimages.html>
12. DOMAČA STRAN CENTER ZA UPORABNIKE, *ActiveX Data Objects (ADO) Frequently Asked Questions*, 12.1.2004, 1.12.2005, <http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q183606&>
13. FOUCAULT, MICHEL, *Wikipedia*, 15.11.2005 <http://en.wikipedia.org/wiki/Main_Page>
14. GAJIČ, ŽARKO, *Communicating Between Forms*, 11.7.2006, <http://delphi.about.com/library/weekly/aa071503b.htm>
15. GAJIČ, ŽARKO, *Constructing the Database Connection String Dynamically at Run*, 13.7.2006, <http://delphi.about.com/library/weekly/aa101805a.htm>
16. GAJIČ, ŽARKO, *DataModules*, 13.7.2006, <http://delphi.about.com/od/database/l/aa101601a.htm>
17. GAJIČ, ŽARKO, *Delphi Programming*, 22.9.2005 <http://delphi.about.com/library/weekly/aa010101a.htm>
18. GAJIČ, ŽARKO, *Drop down list (DBLookupComboBox) inside a DBGrid - part2*, 13.7.2006, <http://delphi.about.com/od/usedbvcl/l/aa101403b.htm>
19. GAJIČ, ŽARKO, *Pictures inside a database*, 2.5.2006 <http://delphi.about.com/od/database/l/aa030601d.htm>
20. LYONS, MAX, ECKHARD, HENKEL AND BURREN, DAVID, *Description of Exif file format*, 3.2.2001, 7.12.2005 <http://park2.wakwak.com/~tsuruzoh/Computer/Digicams/exif-e.html>
21. SLOVENSKA DELPHI STRAN, *Forum*, 12.3.2006, <http://www.delphi-si.com/>