ФАКУЛТЕТ ИНЖЕЊЕРСКИХ НАУКА

Универзитета у Крагујевцу

Машинско инжењерство - Катедра за примењену механику и аутоматско управљање

Назив студијског програма: **Машинско инжењерство**

Ниво студија: **Основне академске студије**

Модул: **Информатика у инжењерству**

Предмет: **Софтверски инжењеринг**

Број индекса: **717/2013**

Н Е М А Њ А Д. К А Т А Н И Ћ

АПЛИКАТИВНИ МОДЕЛ ЗА ИЗДАВАЊЕ АВИО КАРАТА

-ЗАВРШНИ РАД-

Комисија за преглед и одбрану:

1. др. Ненад Филиповић - ментор

2. _____ **Датум одбране:_____**

3. _____ **Оцена: _____**

Крагујевац, октобар 2014;

У оквиру овог завршног рада кандидат треба да:

1. Анализира могућности објектно оријентисаног програмирања помоћу савремених софтверских алата,

2. Презентује основне карактеристике и принципе рада комплексних система за управљање базама података,

3. Креира извештај за штампање коначне принтабилне форме документације у бази података,

4. Примени Microsoft-ове софтверске пакете за израду апликације и омогући њену константу везу са базом података.

Препоручена литература:

[1] Michael Lee (2008) Microsoft SQL Server 2008, преведена књига;

[2] Jesse Liberty (2008), Microsoft Visual Studio 2008, преведена књига;

[3] Ненад Филиповић (2006), Објектно оријентисано програмирање, ФТН Чачак.

Крагујевац, датум

ментор:

Др Ненад Филиповић, ред.проф.

_____

Резиме:

У овом завршном раду су појашњени поступци израде комплексног апликативног софтвера.

Прво поглавље представља увод у објектно оријентисано програмирање. У следећем поглављу се говори о идеји самог завршног рада као и сам систем издавања авио-карата који је у србији заживео доста касније него у западном свету.

Треће поглавље говори уопштено о објектно оријентисаном програмирању и може важити за остале оопј као што су јава, c++ и visual basic. У четвртом поглављу су приказане просте команде C#, као и како само visual basic окружење функционише.

У петом поглављу се говори о појму базе података, објашњавају се структуре података, релација између веза, идентификациони кључеви. Објашњавају се погледи у SQL серверу као и потреба за мултиплим приказом исте табеле у циљу преноса различитих информација истог типа и извора у другу табелу.

Шесто и седмо поглавље приказује вишеслојну софтверску архитектуру и појам релације између Windwows (оперативни систем) – форми (NET framework) – база података (SQL server) –програмски језик ниског нивоа (C#).

Последњи део завршног рада говори о заокруживању рада у једну кратку целину, са благим појашњењима и закључцима о раду.

САДРЖАЈ

1. УВОД У ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

Од самог почетка коришћења рачунара, обрада различитих врста података, била је један од основних задатака. Подаци и информације су постали покретачка снага модерног пословања у целом свету. Када корисник жели да има квалитетне информације о свим сегментима пословног или и приватног живота најбоље је да на одређени начин организује све податке које могу да му пруже информације које су од велике важности у тренутку када су потребне. Поготово се то односи на ситуације када у кратком року мора донети неку квалитетну одлуку. Тада би било најбоље да подаци за сваки појединии елемент буду организовани тако да се могу сместити у табеле са истоврсним заглављем. Може, а веома често и мора да буде више табела које би обухватале све сегменте интересовања. Сви ти сегменти се неретко због своје природе морају организовати у посебне табеле, а те табеле се могу повезивати преко одређених заједничких елемената. Скуп више тих табела које служе једном заједничком циљу, скупа са њиховим везним елелментима назива се базом података. Корисничка апликација чије креирање ћемо представити у овом раду у знатној мери олаксава унос података у базу и смањује ризик од грешки које могу настати људским фактором

Њихов заједнички циљ се односи на свођење веома брзе и успешне инфрормације о свим догађајима који се дешавају унутар једне целине. Када се куца нешто у Word-у, или се врше нека табеларна израчунавања у Excel-у у више табела онда се има додира са базом података. То је у свари питање организације података.

Велику улогу у развоју науке игра развој Информационих Технологија које су најзаслужније за ефикасно чување и обраду података. За брз и ефикасан развој Информационих Технологија,последњих деценија, су заслужни су програми(софтвери), помоћу којих се врши обрада и чување података.

Самим тим развој Информационих Технологија прати и развој програма, чије карактеристике и примену одређују програмиери тако што их пишу у неком од програмском језику. Развој програмских језика почео је од машинских језика најразумљивих процесору (процесор разуме само 0 и 1-це) до објектно орјентисаних језика који сада имају најзнацајнију улогу у развоју програма, а самим тим и информационих технологија.

Објектно орјентисани програмски језици су тренутно најзаступљени програмски језици. Развој програма на овим програмским језицима заснива се на томе што се све представља као објекат који има унапред одређене особине и на основу њих врши се њихова међусобна комуникација.

C# (Ц-шарп) представља један од најмлађих објектно орјентисаних програмских језика. У овом дипломском раду описане су неке од основних теза и синтакси Ц# програмског језика, који прати развој апликације за куповину, издају и штампање карте.

2. МОДЕРНИ СИСТЕМИ ЗА ИЗДАВАЊЕ АВИО-КАРАТА

Промена развоја авио-карата која је овде уследила крајем 20.века чак и данас може збунити технички недовољно информисане људе. Уместо чекања да ћете примити традиционалну штампану карту на вашу кућну адресу данашње авио-карте се издају у електронском облику путем e-maila.

Данашњи тип електронких авио-карата има велику предност, авио-карта у електронској форми не може бити украдена нити изгубљена. Уколико би путник евентуално хаковао ваш e-mail налог, он не би ништа постигао тиме. Да би се укрцао на авион потребан му је и ваш пасош, а на карти има серијски број на основу којег узима купон тек на уласку на аеродром. То значи да и кад би хаковао e-mail, и кад би ручно променио податке на карти, на уласку у аеродром се издаје купон чији се серијски број подудара са вашом картом и који чува право име и број пасоша путника. То је прилично велика и сасвим довољна сигурност при куповини карте.

Ваша електронска авио-карта се налази у бази података авио компаније која вам издаје карту. Путници са собом на аеродром носе само пасош, а приликом чекирања, аеродромски службеник који врши чекирање ваше име и број авио карте проналази у бази података и на основу тога вам штампа бординг купоне.

2.1. КУПОВИНА И РЕЗЕРВАЦИЈА АВИО КАРАТА

За почетак је потребно да одете на неки од веб портала који се баве продајом авио карата те да помоћу њиховог информационог система (у Србији су то готово увек познати глобални резервациони системи „Амадеус“ или „Галилео“) извршите птетрагу летова. Углавном, потребно је попунити маску за онлине резервацију авио карте уношењем неколико параметара као што су датум поласка, датум повратка, полазиште (град из кога желите да полетите), дестинацију тј. одредиште, врсту путовања (уколико путујете у једном правцу, кликните на дугме “један правац”, уколико желите повратну авио карту, кликните на дугме “повратна карта” а уколико имате више полазака-повратака, кликните на линк “мултидестинација”. Такође, потребно је да попуните поља која се односе на путнике: „одрасли“, „деца“ и/или „беба“ као и класу која може бити нпр. економска или бизнис. Када сте попунили тражена поља обично је довољно притиснути тастер „ентер“ на тастатури или кликнити на дугме „пронађи“ (преузето са navidiku.rs).

Овде је описана десктоп апликација, какве се сусрећу на самим аерордромима, међутим други тип апликације је веб апликација коју би крајњи корисник покретао са свог веб прегледника. Модел десктоп апликације помоћу које корисник бира какву карту купује, једноставним избором полазне и одредишне дестинације, летилице, класе седишта и све остало што обухвата једну авио-карту. Авио-карте се могу платити уплатом на текући рачун, веб платном картицом (online куповина) и готовином.

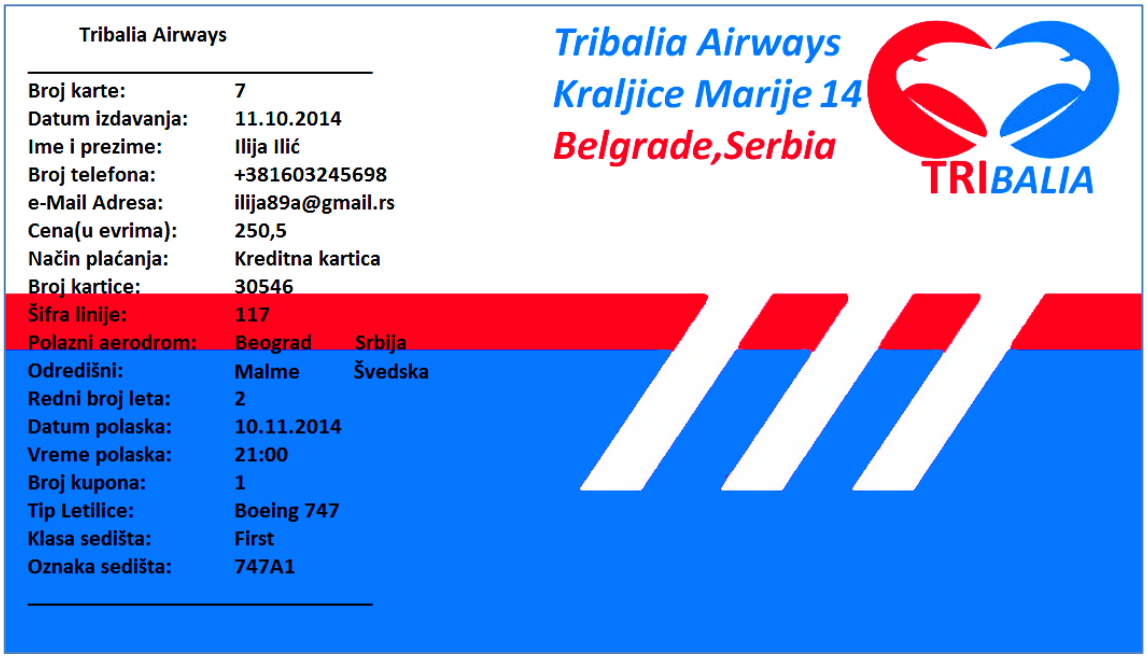
2.2. АПЛИКАТИВНИ МОДЕЛ ЗА ИЗДАВАЊЕ АВИО КАРАТА

би као такав морао садржати све неопходне информације о путнику, месту (полазном/ одредишном), линији, летовима и самој карти и купону. Код путника имамо име и презиме, број мобилног телефона, адресу е-поште на коју ће исти примити електронску карту и број пасоша. Место ће саджати информације о граду и држави аеродрома. Летилица саржи назив летилице као и конфигурацију седишта. Седишта су везана за класу, класа говори о квалитету самог седишта и утиче на цену. Виша класа-скупље седиште. Остале инфромације

су везане за линију, саме летове унутар линија, класе и броја седишта, датум и време поласка. Такође апликација мора садржати принтабилни извештај која преставља саму карту и садржи све потребне информације на једном месту. Извештај може бити и у формату word документе (*.doc(x)), али се препоручују слободни формати као што су *.pdf, *.odt, *.rtf или *.txt. Папир на коме се штампа не мора бити потпуно бланко, него је штавише пожељно да се користи образац на коме већ стоје неке основне информације о компанији која издаје авио-карту (генеричке информације) и обојен у бојама компаније.

Слика 1. *Пример обрасца за штампање авио-карте.*

Дакле овај или неки други сличан образац би се константно прештампавао са правом авио-картом у *.txt формату која би била постављена са леве стране као што је приказано на слици испод.



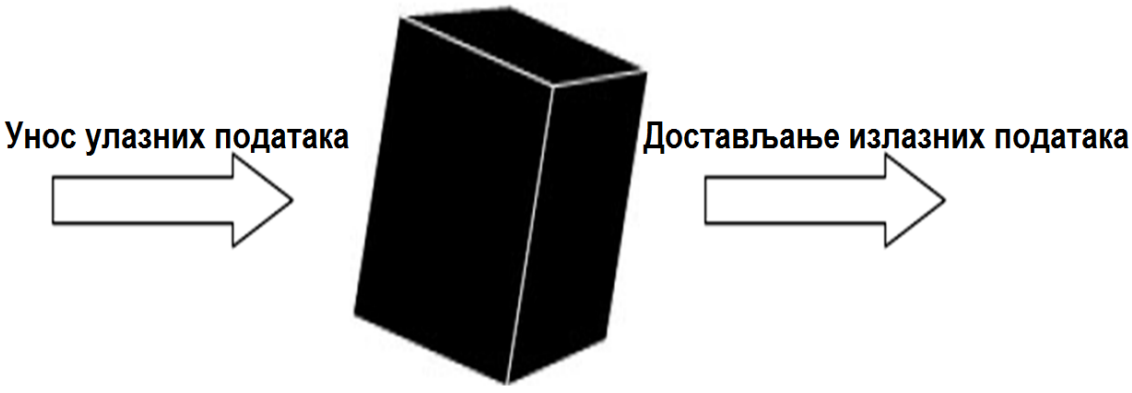
Слика 2. *Пример обрасца са отштампаном авио-картом.*

3. ОБЈЕКТНО ОРЈЕНТИСАНО ПРОГРАМИРАЊЕ

Данас се за програмирање најчешће везује појам софтверско инжењерство. Програмери су, у ствари, инжењери софтвера или популарније ”software девелоперс”. Да би неко могао да буде успешан програмер, морао би бити упознат са великим бројем програмских језика, већином објектно оријентисаних, затим базама података, основним структурама и алгоритмима за манипулацију истим. Међутим, на самом почетку, будући инжењер софтвера мора имати чврсту основу у актуелном начину и стилу програмирања то је објектино орјентисано програмирање. Развој софтверског инжењерства у свету трајао је годинама. Само програмирање софтвера прошло је кроз неколико фаза, од којих су неке фазе и даље у развоју, то јест развијају се паралелно са актуелним. Програмирање које је “освојило свет” своди се на низ програмских наредби које се секвенцијално извршавају (једна за другом). Ово и јесте једна од сувих дефиниција програмирања, која данас није универзално примењива.

Секвенцијално програмирање се изводило коришћењем јасно дефинисаних програмских команди–секвенци, које су се касније груписале у веће модуле, који су названи процедурама. Самим тим, овакво програмирање добија назив процедурално. Велики број данашњих програмских језика је процедуралан и високо употребљив у такозваном, системском програмирању (програмирање модула оперативних система ослања се на подршку брзих процедуралних језика попут програмског језика C). Приликом процедуралног програмирања, подаци које програм користи (обрађује) су одвојени од самог кода, то јест програмске логике, што је велика мана са обзиром на умањене могућности контроле приступа подацима или њеног потпуног одсуства. Ово је велики сигурносни ризик у програмирању.

У филозофији процедуралног програмирања, програм се посматра као black box (црна кутија), која, попут процеса, прима улазне податке, обрађује их и доставља излазне податке, то јест у случају коришћења персоналних компјутера, исписује на стандардном излазу или смешта у меморију рачунара. Сами подаци се најчешће чувају у некој глобалној бази података или на удаљеном серверу. Код, као извршни део програма, код процедуралног програмирања смештен је унутар функција и процедура, које су саме по себи блек бокс систем обраде улаза и добијања излаза.



Слика 3. *“Black Box” филозофија процедуралног програмирања*

Објектно оријентисано програмирање уводи један посебан термин објекат. Објектино орјентисање је, као што име каже, оријентисано ка објекту, који је нека основа оваквог програмирања. За почетак, за објекат је најбоље рећи да је једна од основних градивних јединица објектно орјентисаног програмирања. Ако програм посматрамо као организам, био би састављен из модула различитих функција, попут органа. Ови функционални модули сачињени су из сопствених делова, који су, у ствари, објекти, који међусобно комуницирају.

Дефинисањем једног оваквог ентитета, у објектима орјентисаној програмерској филозофији су сједињени подаци и понашање у једном потпуном пакету (full package). Обезбеђена је јединствена контрола приступа подацима, пошто објекат није прост тип података, попут целог броја или низа знакова, већ скуп примитивнијих врста података, којима је по потреби додато одређено понашање.

3.1. ОБЈЕКТИ

У програмирању, овај термин има веома јасно дефинисано значење. Но, ослонимо се тренутно на дефиницију објекта у логичном смислу. Објекат је, дакле, сваки ентитет који је потребно представити помоћу одређених стандардом дефинисаних метода описивања природних или вештачких ентитета, појава, догађаја. Дакле, филозофија је једноставна, све се представља објектом, помоћу објекта или помоћу више различитих објеката. Све је објекат.

Објекат је једнствена целина која садржи податке и понашање. Сам по себи, може садржати друге податке, целе бројеве, низове карактера, бројева, друге објекте. Подаци исказују стање објекта, и зовемо их атрибути. Дакле, објекат има стање и понашање.

Узмимо као пример објекта једну обичну канту за отпатке. Да би била објекат, потребно ју је представити одређеним стањем и понашањем. За почетак, одговоримо на питање шта је то стање једне канте за отпатке?То су њени атрибути попут димензија, боје, процентуалне попуњености и тако даље. На слици која следи можемо приметити да се у канти налазе згужвани папири. И они су објекти, само другачијег типа него што је канта.

Слика 4. *Објекти "канта" и "згужван папир"*

На слици 2. је дата скоро верна спецификација ова два објекта. Приметићемо да сваки атрибут има неку своју вредност. Ово су вредности које један објекат разликују од другог (неће сваки комад папира бити истог пречника, иако је појам пречник у овом случају банализација). Све ове вредности су одређеног типа. Можемо приметити да су пречници база ове канте приближне облику ваљка дати у центиметрима. У програму, овај податак био би записан као ненегативна целобројна вредност, док би име материјала израде представљао неки низ карактера. Процентуална попуњеност би била неки реалан број. Оно што на слици није наведено као део спецификације једне канте за отпатке јесте њена могућност да складишти објекте попут ових згужваних папира. У програму, ово би морало бити имплементирано, то јест као податак/ атрибут/показивач стања постојао би неки низ објеката који представљају отпатке у канти.

Сада је потребно одговорити на питање шта је то понашање једне канте за отпадке? Иако нам здрав разум говори да се једна канта за отпадке не "понаша", осим ако није подпомогнута добром технологијом из сфере роботике, ми као програмери можемо ово питање сагледати из једног другачијег угла. Само понашање не мора значити да сама канта нешто ради, већ и оно што се ради са кантом може представљати понашање. Тако, процес убацивања отпатка у канту можемо назвати "убаци()", а процес избацивања свих отпадака из канте "избаци()". Заграде нам омогућују да разликујемо атрибуте од метода, јер су ово заиста методе понашања канте, то јест методе контроле над њом.

3.2. КЛАСЕ

У природи се може срести велики број сличних ентитета. Неки су идентични, неки имају јако мале разлике, а неки су потпуно различити. Међутим, човек је, захваљујући својој интелигиенцији, извршио неке основне класификације ствари, бића и појава које је сретао током живота. Тако је, на пример, настала класификација живих бића на људе, животиње и биљке. Пре свега је логично да неке појмове зовемо заједничким именима, иако се они у неким детаљима разликују. Дакле, класа у природном поретку дефинише неке заједничке особине и понашања ентитета које се убрајају под њену класификацију, а који се разликују у детаљима везаних за одређене особине.

У програмирању, класа се третира као шаблон објекта, шематски план који се користи приликом креирања објеката. Класа дефинише које ће атрибуте и понашања поседовати сваки објекат те класе, све почетне вредности и захтевана понашања реализована методама. Дакле, сваки објекат је једна инстанца своје класе. Узмимо као пример нашу канту за отпадке.За почетак, постоји више различитих канти за одпадке. Свака од њих имаће неке своје димензије, биће одређене боје, а током њиховог коришћења њихова процентуална попуњеност ће се прилично динамично мењати. Све оне су канте, тако да би класа "Канта" дефинисала све објекте тог типа, обавезујући сваки од њих да поседују наведене атрибуте да искажу њихово стање.

Сама по себи, класа је тип податка вишег нивоа (целобројне вредности, карактери и остали су прости – примитивни типови података). Уколико би, на пример, поменута канта за отпадке могла да прима само згужвани папир као своје отпадке, као што би за атрибут процентуалне попуњености било назначено да је податак типа реалан број, тако би за низ отпадака као атрибут било назначено да је типа "Згужвани папир".

У објектно орјентисаном концепту, појам када објекат садржи објекте (мобилни телефон садржи екран, тастатуру, батерије...) назива се композиција. Релација композиције између два објекта назива се релација ИМА.Дакле, мобилни телефон ИМА батерију.

3.3. МЕТОДЕ И КОМУНИКАЦИЈА МЕЂУ ОБЈЕКТИМА

Као што је већ речено, методе реализују захтевано понашање класе. Сви објекти исте класе имају исте методе, то јест методе које врше исту функцију или низ функција. Оне реализују активности објеката и обезбеђују понашање. Неке методе омогућавају измену вредности атрибута. Ово је веома уобичајена пракса у објектно орјентисаном програмирању. У већини случајева, приступ подацима унутар неког објекта би требало да контролише сам објекат – ни један објекат не би требало да може да мења вредности атрибута другог објекта, уколико то није експлицитно дозвољено или не постоји метода која је задужена за то.

Свака метода се мора дефинисати то јест поседовати:

1. име методе(описује шта метода ради),
2. аргументи(вредности које јој се прослеђују),
3. повратна вредност(тип повратне вредности).

Методе служе и за комуникацију међу објектима. Комуникациони механизам између објеката су поруке. Да би неки објекат позвао неку методу другог објекта, он му шаље поруку, а одговор другог објекта дефинисан је повратном вредношћу која је враћена након иницијалне поруке. На пример, уколико би објекат “Пера” желео да сазна име објекта “Милена”, он би позвао методу објекта “Милена”, под именом “getIme()”, која је задужена да као своју повратну вредност врати низ карактера који представљају име.

3.4. ЕНКАПСУЛАЦИЈА И ИНТЕРФЕЈСИ

У току програмирања, увек је битно “ко коси, а ко воду носи”, то јест од великог је значаја верно контролисати приступ подацима неког објекта. Велики је пропуст уколико програмер у сваком моменту није свестан ко све може нехотице променити интерне податке неког објекта. Ово се највише изражава приликом тестирања софтвера, без којег ни један софтвер неће изаћи на тржиште.

Енкапсулација је способност објекта да сакрије своје податке или да их учини селективно доступним другим ентитетима. Одређене детаље, који нису важни за коришћење објекта, потребно је сакрити од осталих објеката, како они директним приступом не би мењали битне податке. На пример, уколико неки објекат има могућност рачунања неке сложене математичке операције, кориснику је потребно пружити интерфејс који ће примити параметре који су потребни за рачунање, а вратити само резултат. Није потребно да корисник познаје алгоритме који се користе за само израчунавање, као ни међу-податке (пр: вредност константе π или константе e). Енкапсулација се обезбеђује интерфејсима, то јест имплементацијом самог објекта.

Интерфејс је основно средство комуникације међу објектима. Било које понашање објекта мора бити позвано поруком преко интерфејса, који мора потпуно да опише како корисник комуницира са објектом класе. У већини случајева, интерфејси су јавни, што је логично, обзиром да су посредници у комуникацији међу објектима и са корисником (с тим што је, у програму, сам корисник један објекат, представљен у коду).

На познатом порталу Википедија стоји: “Интерфејс је обично апстракција коју за себе пружа ентитет спољашњој средини. Он раздваја методе екстерне комуникације од интерних операција, што омогућава ентитету да буде модификован изнутра, а да то не утиче на његову комуникацију са средином”.

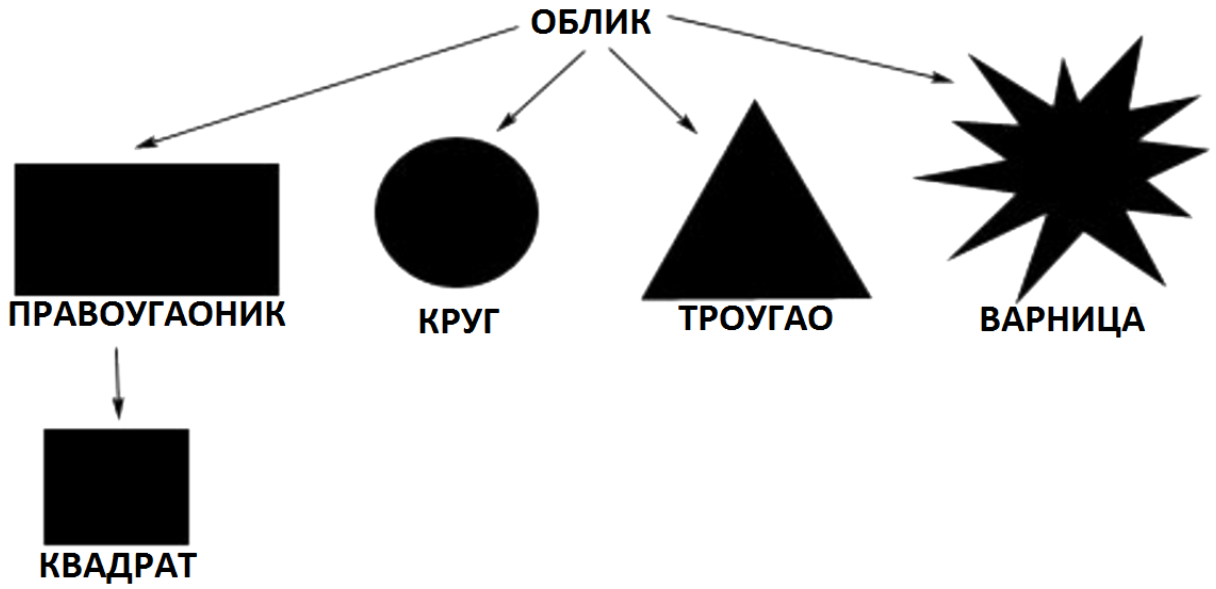
Дакле, интерфејс је апстракција објекта, његов начин презентовања спољашности, преко којег ће други објекти комуницирати са њиме. У основи, интерфејси не садрже атрибуте, већ само методе. Битно је напоменути да би, уколико објекат поседује неки интерфејс, требало да интерно имплементира понашање које тај интерфејс описује (не би било добро да објекат својим интерфејсом тврди како може да израчуна интеграл, а да интерно алгоритми за овај рачун не постоје).

3.5. НАСЛЕЂИВАЊЕ

Приликом дефинисања класе, поменули смо класификацију живих бића. Када бисмо дефинисали класу “Живо биће”, могли би дефинисати неке особине свих живих бића које су им заједничке, попут атрибута “старост” (сва жива бића имају одређену старост). Такође, када би дефинисали класу “Сисар”, могли би као особину свих сисара навести атрибут “боја очију” (сви сисари имају очи и то одређене боје). Дефинисањем класе “Човек”, у могућности смо да дефинишемо атрибут “име” и тако даље. Међутим, када би у класи “Сисар” дефинисали атрибут “старост”, постојало би преклапање атрибута, јер и класа “Живо биће” има овај атрибут. Наиме, сви сисари су жива бића, као што су и сви људи сисари и жива бића. Овде се може приметити одређен однос између класа. Овакве односе дефинише наслеђивање.

За почетак, потребно је поменути још једну релацију међу објектима. То је релација ЈЕ или ЈЕСТЕ“. Наиме, човек ЈЕ сисар, сисар ЈЕ живо биће (наравно, из овога следи: човек ЈЕ живо биће).

Наслеђивање се може дефинисати као разврставање класа дефинисањем истих особина различитих класа (“старост” је особина свих живих бића, па и сисара, а тиме и људи). Ово омогућава некој класи да наследи атрибуте и методе (стање и понашање) неке класе, чиме се дефинише однос између њих, где је класа која наслеђује подкласа, а класа од које наслеђује суперкласа.



Слика 5. *Наслеђивање и релација ЈЕСТЕ*

На слици 3. може се приметити да је главна суперкласа “ОБЛИК”, а да су њене подкласе “ПРАВОУГАОНИК”, “КРУГ”, “ТРОУГАО” и “ВАРНИЦА”.Ово значи да важи релација јесте, то јест правоугаоник ЈЕСТЕ облик, круг ЈЕСТЕ облик и тако даље. У програмирању, ова релација говори о томе да је класа “КВАДРАТ” наследила све атрибуте и методе од класе “ПРАВОУГАОНИК”, наравно и оне које је она наследила од класе “ОБЛИК”, зато што квадрат ЈЕСТЕ облик.

Из претходног примера се види да је стабло наслеђивања потенцијално велико. Када би покушали да хијерархијски дођемо од жибог бића до конкретне особе, морали би проћи неки овакав ланац: Живо биће – Сисар – Човек – Србин – Станимир... Потенцијал механизма наслеђивања лежи у апстракцији и техници организације суперкласа и подкласа.

У једном хијерархијском ланцу класа, свака подкласа је наследила атрибуте и методе своје суперкласе. Такође, наследила је све интерфејсе. Међутим, свака класа за себе имплементира своје интерфејсе, то јест свака класа на свој начин реагује на побуде из спољашње средине, то јест даје свој сопствени одговор, који може бити, али најчешће није исти као одговор који би дала суперкласа, од које је интерфејс наслеђен (поента интерфејса јесте да га свака класа имплементира на себи својствен начин). Оваква појава, где се начини имплементације одређених понашања класе, то јест њених објеката разликују у ланцу хијерархије наслеђивања, зове се полиморфизам.

4. MICROSOFT .NET РАЗВОЈНО ОКРУЖЕЊЕ

Microsoft .NET Framework је софтверска платформа која може бити инсталирана на рачунару који покреће Microsoft Windows оперативни систем. Она укључује велики број готових библиотека кодова за уобичајне проблеме у програмирању. Базе класа пружају широк спектар могућности укључујући кориснички интерфејс, приступ, податцима, базама, криптографијама, развој веб апликација, нумеричке алгоритме и мрезне комуникације. Библиотеке класа се користи од стране програмера који га комбинују са кодом за израду апликација. .NET подржава више програмских језика(C++, Visual Basic, Javu, C#), на тај начин му омогућава интерпортабилност при чему сваки језик може бити написан на другом. Доступан је свим језицима које .NET Framework обухвата. Како би могле да се пишу апликације није само довољно имати инсталиран .NET Framework, потребан је и Microsoft SDK и Visual Studio.

Microsoft је почео да развија .NET крајем 90-тих година под називом "Next Generation Windows Service" (Следећа генерација windows сервиса), а прва верзија настала је крајем 2000. године бета верзија .NET Framework 1.0. Укључујући прву верзију .NET Framework-а изашло је још неколико верзија написаних у табели:

Верзија .NET Framework	Датум	Visual Studio
1.0	13.02.2002.	Visual Studio .NET
1.1	24.04.2003.	Visual Studio 2003
2.0	07.11.2005.	Visual Studio 2005
3.0	06.11.2006.	Visual Studio 2005
3.5	19.11.2007.	Visual Studio 2008
4.0	12.04.2010.	Visual Studio 2010
4.5	12.09.2012.	Visual Studio 2012

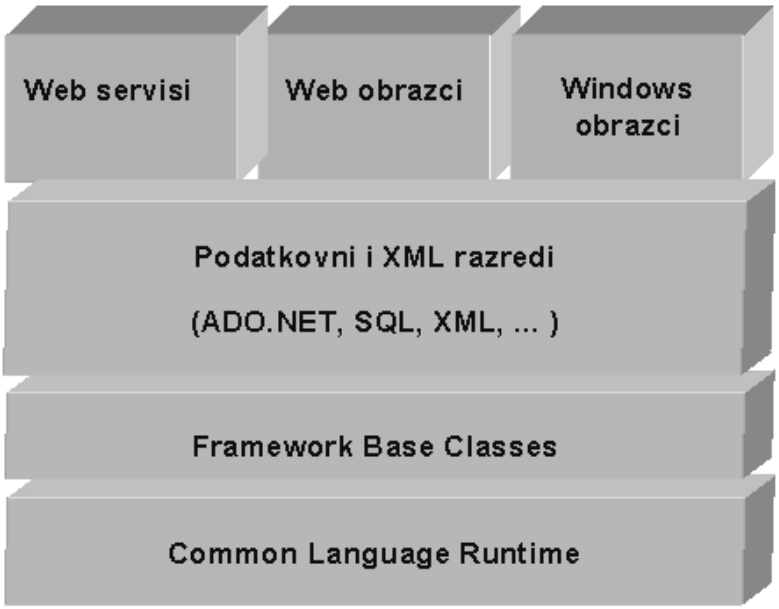
Табела 1. *Развој .NET Framework-а*

4.1. АРХИТЕКТУРА .NET-А

.NET је развојна платформа са новим интерфејсом за програмирање апликација (енгл. application programming interface, API) наследио је функционалност и могућности окружења за програмирање класичних оперативних система Windows, али и усвојио бројне, различите технологије које Microsoft развија од краја деведесетих година прошлог века. То обухвата рад са компонентама COM и XML, објектно оријентисан дизајн, подршку за нове протоколе за веб сервисе какви су SOAP, WSDL i UDDI, и оријентисаност ка интернету. Све је то интегрисано у оквиру DNA архитектуре (Distrubuted Net Applications). Microsoft је много ресурса ставио у службу развоја платформе .NET и сродних технологија. Опсег технологија које обухвата .NET је огроман. Платформу чине следеће три групе технологија:

- Скуп језика – указујући C# i VB, скуп алатки за развој програма (између осталог, Visual Studio .NET), опсежна библиотека класа за превлачење веб сервиса и апликација за веб и Windows, заједничко извршно кружење (Common Language Runtime, CLR) за извршавање кода објеката направљених u .NET Framework-у.
- Две генерације сервера .NET Enterprise Server постојећи, и они који ће постати доступни у наредне две до три године

- .NET подршка која није само за PC рачунаре, већ и за нове уређаје – од мобилних телефона, до платформи за компјутерске игрице .NET Framework.



Слика 6. *Архитектура .NET-а*

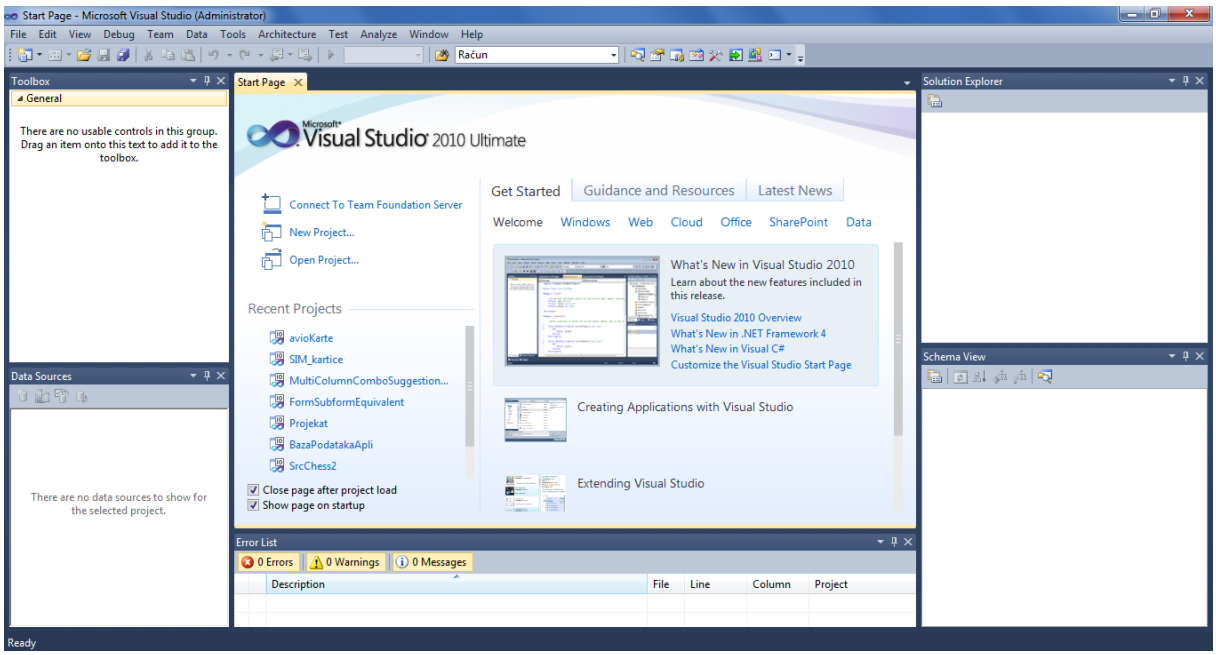
4.3. VISUAL STUDIO 2010 ULTIMATE EDITION

Visual Studio 2010 Express Edition је програмско интегрисано окружење богато алатима који садрже сву функционалност за израду малих и великих програма на свим језицима .NET-а, па чак могућа је комбинација модула различитих језика.

Након се преузме инсталациона датотека (постоји више варијанти преузимања зависно да ли желите да преузмете комплетан пакет свих доступних програмских језика или желите само део софтвера за одређени језик на пример C#, Java…), софтверског пакета Microsoft Visual Studia 2010. Потребно је неколико корака у инсталацији софтвера:

1. Два пута притиснути vcssetup.exe датотеку, која се налази на радној површини. Сачекати неколико секунди док инсталациони програм не учита све неопходне компоненте.
2. На инцијалном екрану притиснути тастер Next.
3. Појављују се серија дијалога. Не мењати подразумеване вредности, већ само притиснути Next тастер, како би се наставило са поступком инсталирања.
4. Након што се преузму и инсталирају сви неопходни елементи, можда ће бити неопходно да се рестарује рачунар.

На слици 7. Приказан је први прозор који се отвара приликом стартовања Microsoft Visual Studia 2010. Sledeći кораци у раду са овоим софтвером било би бирање на ком језику желите да писете програм или да неки од ранијих да отворите.



Слика 7. *изглед Visual Studia 2010*

Visual Studio подржава велики број најразличитих апликација,које јој омогућава .NET IDE (Integrated Development Environment) интерфејс за стварање и тестирање, сходно томе битно је сваку нову апликацију придружити некој од група.

4.4. ОСНОВЕ C# ПРОГРАМСКОГ ЈЕЗИКА

C# (C Sharp) је један од млађих програмских језика настао 2002. како саставни део Microsoft .NET Framework-a 1.0. C# је једноставан објектно оријентисан програмски језик опште намене. Развио га је Microsof тим који је водио Андрес Хејлсберг. Последња верзија .NET-a је 4.5.2 која је завршена 12.07.2014. Преставља неку напредну верзију мешавине C++ i Jave, мада има и сличности са Visual Basic-ом. У сустини је најсличнији C++.

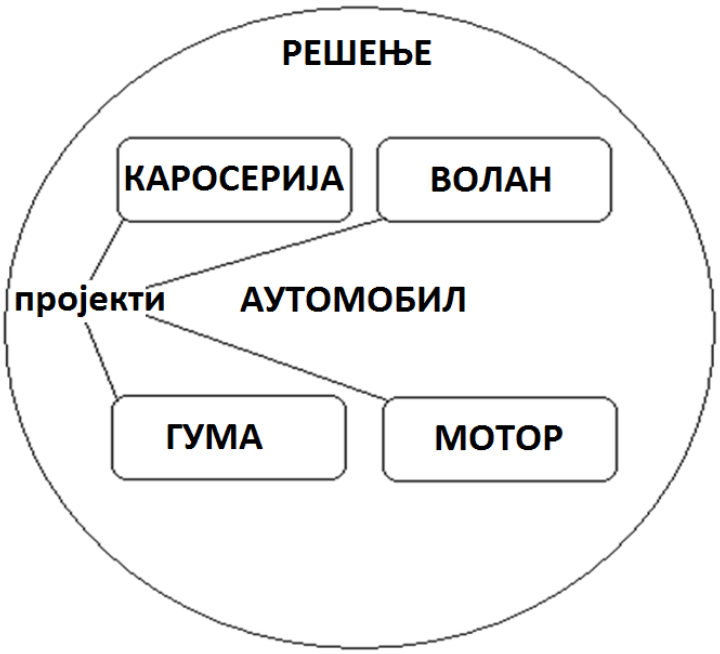
Када стартујете Visual C# Express(deo Visual Studia за писање апликација C# кодом) развојно окружење прво је неопходно да дефинишете који ће те тип апликације да развијете од основна 3 типа .NET апликација:

1. Конзолна апликација, која је пројектована тако да се изврсава на командној линији и која не поседује одговарајући кориснички интерфејс.
2. Windows апликација,која је пројектована да се извршава на корисничкој радној површини и која поседује одговарајући кориснички интерфејс.
3. Библиотека класа, која садржи функционалност, које се могу вишеструко користити и које се могу примењивати у конзолним и wиндowс апликацијама. Библиотека класа се неможе самостално извршавати.

Независно који тип апликације желите да развијате, онда када користите C# u Visual Studi-u integrisanom razvojnom okruženju, neophodno je da kreirate projekte i rešenja:

- Пројекат(project) је класификација која се користи за описивање .NET апликације.
- Решење(solution) је класификација која се користи за описивање већег броја .HET аплкација, које су на неки начин међусобно повезане једне са другима.

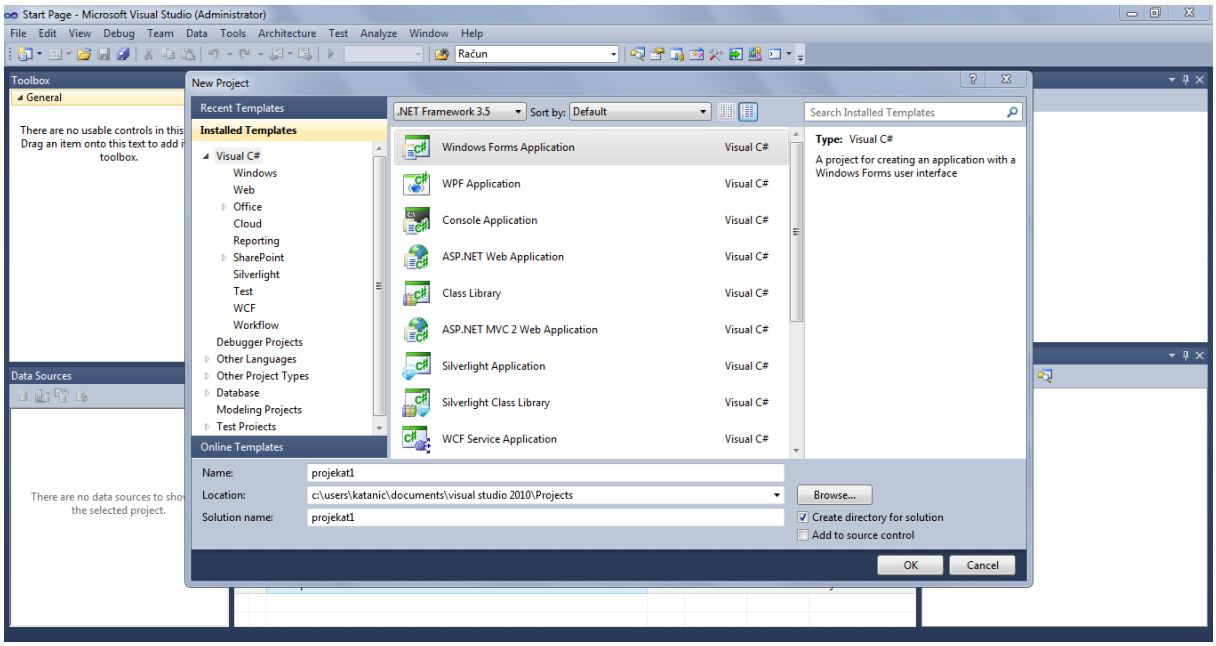
Када би сте креирали на пример аутомобил, пројекти би могли да вам буду волан, гуме, мотор или кароseриа аутомобила. Све ове пројекте можете да ставите у решење, чији је назив аутомобил (шематски приказано на слици 8.). Што значи да једно решење може садржи више пројеката.



Слика 8. *Пројекат и Решење*

4.5. КРЕИРАЊЕ И ПОКРЕТАЊЕ АПЛИКАЦИЈА У C#

Креирање Windows Form апликације почиње када стартујете Visual Studio или Visual C# Expres Edition. На слици 9.приказано је како изгледа прозор за одабир типа новог пројекта желите да направите. До приказаног прозора са слике долази се притиском на опцију File>>New>>Project затим у оквиру Templates треба одабрати једну од випше типова апликација, а у овом случају је то Windows Form, дате назив апликацији и одредите локацију за смештање пројекта. Наравно постоји увек могућност отварања већ постојећих пројеката.

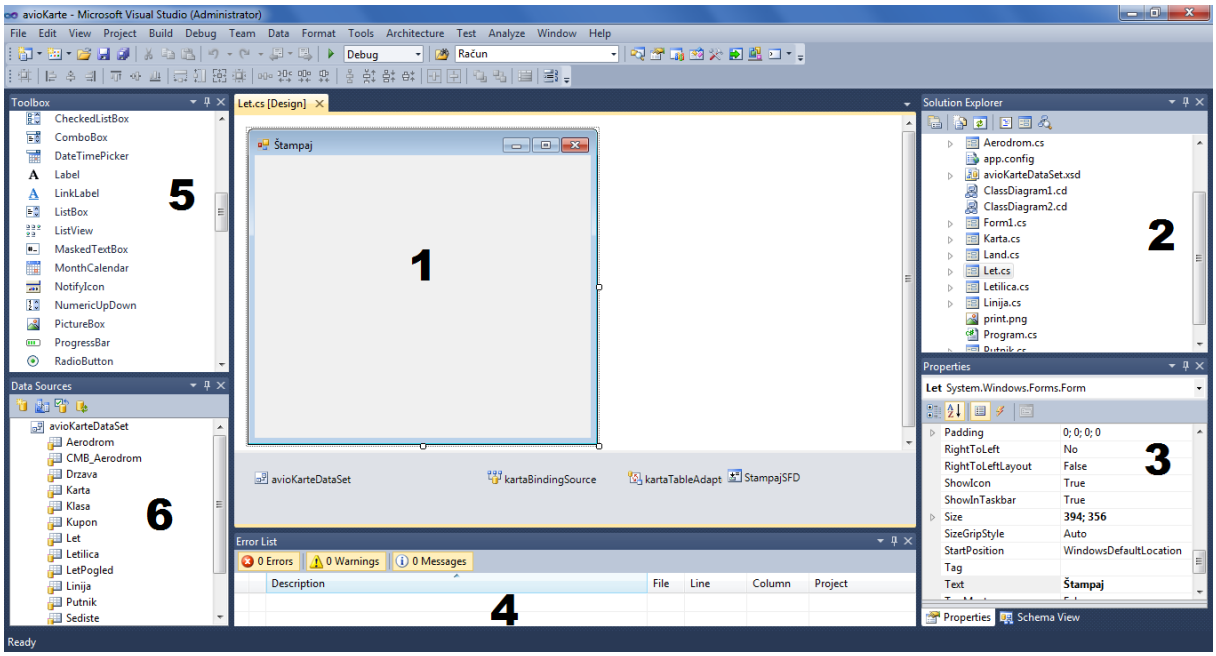


Слика 9. *Креирање windows форм-апликације "пројекат1"*

Приликом давања имена windows апликацији, као у нашем примеру "пројекат1", треба нагласити да када се креира нова апликација Visual C# Express развојно окружење име којим сте именовали апликацију додељује и у оквиру именовања целокупног решења. Сваки следећи пројекат који се додаје у решење неће променити назив решења. Промена назива

решења се ради тако што се десним тастером миша, у делу Solution Explorer, притисне на Solution затим се у падајућем менију одабере опција Ренаме и напише жељено име.

Додавање још пројеката у решење ради се притиском десног тастера миша на Солутион,у Solution Explorer-у, и одабере жељена апликација за пројекат потребна у решењу.Најчешћа је пракса да се додају бибилиотеке класа у одговарајуће решење као један од пројеката. Колико једно решење има пројеката пише u Solution Explorer-у уоквирено заградама у првој ставци тог менија.



Слика 10. *Почетак креирања Windows form-е у VS Ultimate-у*

На слици 10. виде се најчешће коришћени делови Visual Studia, приликом креирања једне windows форме C# језиком. На слици су назначени неки делови прозора који представљају:

1. Део у коме се дизајнирају форме, писање кода зависно на ком, од креираних делова апикација понуђених у Solution Exploreru, ако желимо да вршимо измене. Windows форме имају кориснички интерфејс тако да је њихов основни приказ у Visual Studiu у облику који не подразумева код, такав случај није код конзолних апликација и библиотека класа, њихов приказ је искључиво у облику линија кода приликом њиховог дизајна.
2. Solution Explorer који служи за лакшу прегледност целокупног решење. Подразумева падајући мени у коме се налазе сви пројекти везани за жељено решење. У примеру горе битно је нагласити да се "Решење1" састоји од два пројекта, први са називом "Библиотека" у којој ћемо сместити неке класе које ће бити потребне у изради windows форме "Пројекат1".
3. Properties служи за преглед својстава која се користе приликом израде windows форми. Неке од често коришћених својстава су: Name, Text, Hight, Wight, Minimun size, Maximum size, Left и Top.
4. Error list представља део кода који ако дође до евентуалне грешке током израде пројекта упозорава и објашњава где је и зашто дошло до грешке па програм неће да ради.

5. Toolbox је палета са алатима за дизајн windows форми и контроли.
6. Data Sources приказује шему базе података и служи за одржавање константне везе између апликације и сервера базе података.

Један од начина покретања програма у Visual Studiu је одабиром опције Build на Toolbary и притикањем левог тастера миша на падајућем менију, где треба одабрати опцију Build Solution, а VS ће само направити ваш програм који касније покрећете одабиром опције Debug из toolbara, па на падајућем менију одабрати Start Debugging. Други начин је притиском тастера F5 на тастатури VS аутоматски покрене опцију Start Debugging која покрене програм али притом и направи пројекат што се ради опцином Build Solution. Приликом израде било каквог програма (решења) најбоља пракса је да се одмах након креирања решења покренути опција Build Solution, а касније измене програма да се проверавају само притиском тастера F5 или покретањем опције Start Debugging.

4.6. ПОЧЕТНИ ДЕЛОВИ КОДА ПРОГРАМА

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Primer1
{
            class Program
            {
                  static void Main(string[] args)
{     /*komentar sluzi da objasni sta rade pojedini delovi koda
a u toku prevodjenja kompajler ih ignorise*/
                        Console.WriteLine("prvi program u C#");
                              Console.ReadKey();
                  }
}
}
```

Код који је написан представља код једне конзолне апликације која исписује на екрану поруку “први програм у C#”.

using-је део кода који укључује коришћење неког од System namespac-а из.Net Frameworka или неког креираног namespace-а,

namespace-представља назв како се неки садржај представља у .Net-у на пример нека библиотека класа(у претходном примеру наме када смо креирали “Решење1” имали смо библиотеку класа “библиотека” и то је њен namespace преко кога се референцира на тај део решења),

class-представља класу која се ограничава витичастим заградама { тело класе },

static void main() - је метода(функција) у којој се извршава програм, јединствена је због кључне речи

static која означава да непостоји дупликат ове функције. Без ње програм се неби извршавао, коментари у C# слузе као подсетник на одређене делове кода и њих компајлер игнорише могу бити писани у једној линијји "//коментар" или у више линија као горе на примеру,

WriteLine i ReadKey-су функције које припадају класи конзоле на коју се референцирају преко "**.**"(тачке).

4.7. ПОЈАШЊЕЊЕ ОСНОВНИХ КОМАНДИ TOOLBOX-а У C#

Почетни изглед, након именовања и креирања виндовс форм апликације, је као на слици 10. (у поглављу 5.1.) и обухвата углавном делове описане у оквиру те слике. Нај битнији део Visual Studio за креирање виндовс форми је Toolbox који у себи садржи све контроле које се користе у креирању виндовс форми кориштењем .Net-а, а самим тим и C#. Класе за управљање контролама налазе се у оквиру именског простора System унутар овог простора налази се Windows а у оквиру њега именски простор Forms у коме се налазе контроле за .Net програмирање windows форми, које обухвата и C#, налазе се у оквиру System.Windows.Forms. Ако желимо да користимо било коју контролу у програмирању windows форми потребно је навести овај именски простор као референцу (приликом креирања windows форми овај простор се аутоматски додаје на почетку кода).Свака контрола представља променљиву (објекат). Неке од најчешће коришћених контрола су:

Button - контрола која обезбеђује покретање одређених акција.
Label - контрола за приказ текста и користи се обично у комбинацији уз
друге контроле да би означила њихову намену.
LinkLabel (hiperlink) за повезивање са другим објектима.
TextBox - поље за унос текста (карактеристичан је јединствен формат за сваки знак у тексту).
RichTextBox - поље за унос форматрираног теста (могуће је променити формат сваком појединачном знаку у тексту).
RadioButton - контрола избора (од више RadioButtona у групи, у једном моменту може бити означен највише један).
CheckBox - контрола потврде која може бити означена или не (употребом више CheckBoxova омогућава се избор произвољног броја наведених опција).
ListBox - контрола за приказ листе елеменат са које корисник може изабрати једу или више ставки (дефинишемо коришћењем својства SelectedMode).
ComboBox - контрола која уз себе обједињује неке могућности контрола TextBox i ListBox, попут ListBox-а садржи листу ставки, али се може изабрати само једна, омогућава уношења текста као и TextBox.
GroupBox - контрола која омогућава груписање више контрола у једну целину, најчешће се користи у комбинацији са RadioButton, у оквиру једног GroupBox-а може бити означен највише један RadioButton, GroupBox се често користи за обједињавање контрола које истовремено треба (Enabled) или онемогућити, односно које истовремено треба приказати (Visible), или скратити на форми.
PictureBox - контрола за приказивање слике.

5. РЕАЛИЗАЦИЈА БАЗЕ ПОДАТАКА ЗА ИЗДАВАЊЕ АВИО-КАРАТА

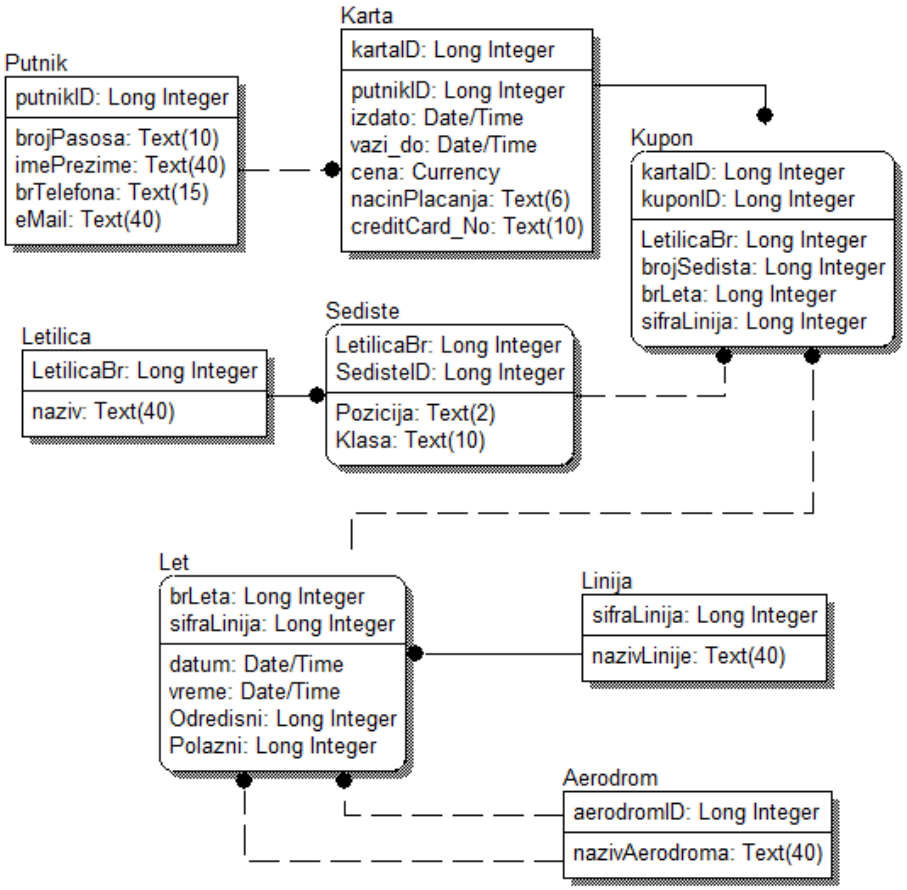
База података представља колекцију међусобно повезаних података који су организовани у табеле и друге структуре података, а користе се за једну или више апликација. Основна намена базе података је да буде складиште за податке. Из „дефиниције" базе података види се да је она колекција међусобно повезаних података организованих у табеле. Подаци у базама података су организовани у дводимензионалне табеле. Табела може да има више колона, где свака колона представља неку особину или атрибут. Врсте табеле чине конкретни подаци, односно конкренте вредности особина/атрибута неког објекта. Које ће табеле да садржи база података зависи од проблема за који треба реализовати базу података. Међусобна повезаност података је оно по чему се база података разликује у односу на фајл системе (датотеке) и програме за унакрсна израчунавања ко што је Excel.

Поступак избора и дефинисања табела за базу података је део процеса моделирања односно изградње модела података. Релациони модел података је најпопуларнији и најраспрострањенији модел података данас. Релација, као основни концепт релационог модела је заправо математичка релација, и има једноставну репрезентацију у облику дводимензионалне табеле са подацима. Овај модел даје основу за језик високог нивоа за приступ подацима. У фази пројектовања базе података треба најпре препознати објекте реалног света (ентитети) за које треба чувати податке и препознати њихове атрибуте. Сваки објекат, односно ентитет, поседује нека својства. Својства или атрибути објекта ће бити представљени колонама у одговарајућој табели. Објекти међусобно могу бити повезани различитим односима односно релацијама.

Реализација базе података за издавање авио карата захтева дефинисање модела података (слика 11), док су за генерисање базе података неопходни следећи подаци:

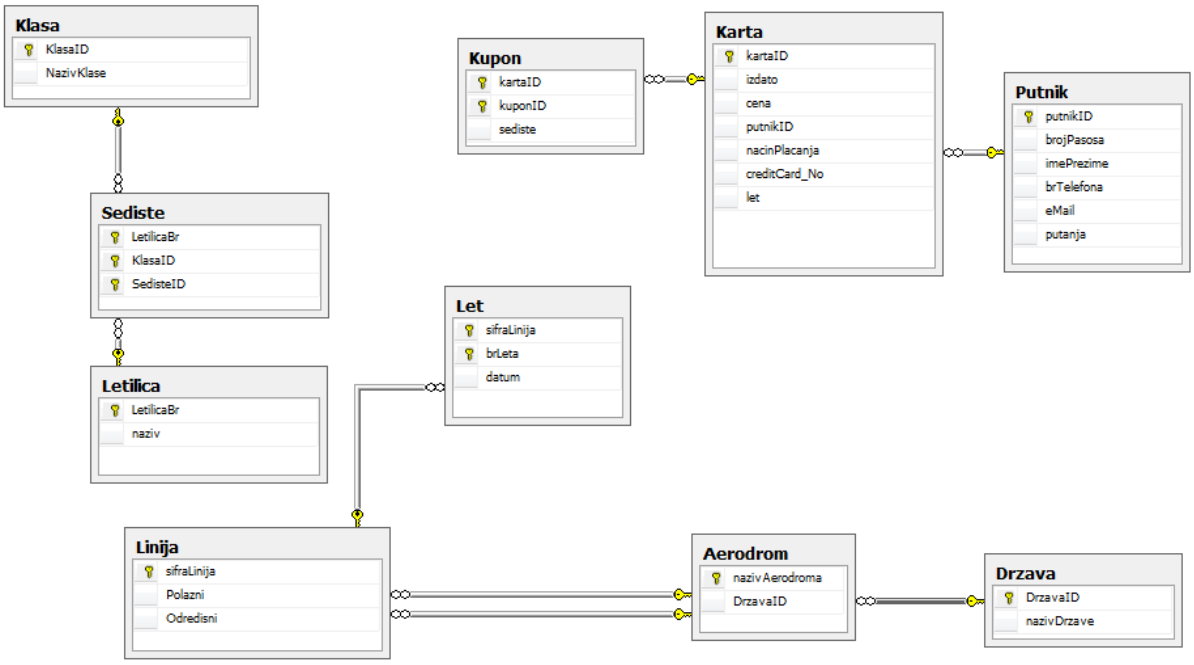
1. Подаци о путнику – број пасоша, име, презиме, број мобилног, адреса е-поште, слика;
2. Седиште, састојано од назива летелице, класе седишта и нумерације седишта. Комплексни кључ састојан од три примарна кључа, пренос кључа се изводи SQL View-ом;
3. Летилица – примарни кључ (идентификујући и асоцијативни) и назив ваздухоплова;
4. Линија - шифра линије (природни број) и назив локације полазног/одредишног аеродрома;
5. Аеродром – називАеродрома (текстуални примарни кључ представља локацију аеродрома) и Држава_ИД(трословна регистарска ознака државе где је тај аеродром);
6. Лет – иста линија може имати више летова, овде имамо идентификујућу везу. Кључ шифра_Линија се преноси из табеле Линија, број лета и датум/време;
7. Држава – Држава_ИД (тросолвна регистарска ознака државе) и назив_Државе (пун назив);
8. Карта – редни број карте,датум издавања,цена карте,коме се продаје(Путник_ИД), начин плаћања, бројк картице (необавезан унос) и лет;
9. Купон – зависни ентитет који преноси примарни кључ карте. Једна карта може имати више купона (породична карта). Означено број купона за карта_ИД и нумерација седишта.

На следећој страни су приказане слике 11 и 12. Слика 11 представља почетну замишљену форму релационог модела у ERwin-у, а слика 12 битно измењену и коначну шему базе података која је применљива и која ће бити накнадно кориштена у самој апликацији. Коначни дијаграм није у потпуности повезан, то се изводи путем SQL погледа (views);



Слика 11. *Почетна форма дијаграма ентитета*

Релациони модел података представља теоријску основу за базе података релационог типа, али је у току израде базе могуће његово проширење у зависности од потреба клијента.



Слика 12. *Крајња форма дијаграма ентитета*

5.1. КРЕИРАЊЕ БАЗЕ У MICROSOFT SQL СЕРВЕРУ 2008

Microsoft SQL Сервер је систем за управљање релационим базама података развијен од стране Microsoft-a. Као база података, то је софтверски производ чија је примарна функција чување и преузимање података по налогу других софтверских апликација, било да је то на истом рачунару, или на другом рачунару који ради преко мреже. Постоји најмање десетак различитих издања система Microsoft SQL Сервер за различите клијенте и за различита радна оптерећења (у распону од малих апликација које чувају и преузимају податке на истом рачунару, до милиона корисника и рачунара који приступају огромним количинама података на интернету у исто време). Примарни упитни језици су T-SQL i ANSI SQL.

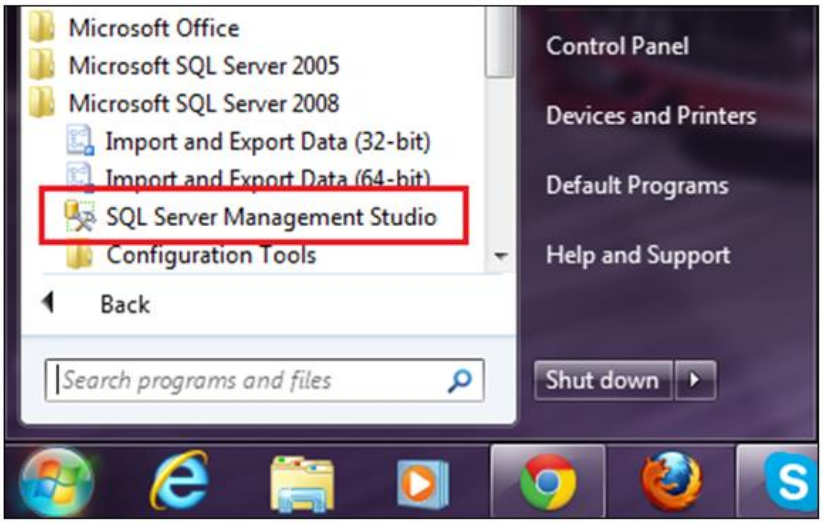
Битно је напоменути да се под термином подаци не сматрају само нумерчке и текстуалне информације. Потребе су порасле, тако да сада у базу можете сместити и друге информације као што су слике, аудио и видео информације, географске податке, документе и генерално било који вид информације који се појављује у свакодневном раду.

SQL сервер је робустан и поуздан систем за управљање базом података. Дизајниран је да има на стотине, или чак хиљаде корисника који приступају бази у било ком тренутку. Конкуренти са друге стране, не могу поднети ову врсту оптерећења добро. То чини SQL сервер савршено погодним за базу података сајтова. SQL сервер садржи и неке напредне административне алатке базе података које омогућавају организацијама да расподеле задатке, издају одређена ограничења, оптимизују базу података, подесе безбедоносне рачуне, врше пренос података са других извора, и још много тога.

5.2. ПОКРЕТАЊЕ SQL SERVERA 2008 И ПРОГРАМА MANAGMENT STUDIO

Да би се покренуо Microsoft-ов SQL Server 2008 и програм SQL Server 2008 Management Studio, потребно је урадити следеће:

1. У менију Start, прећи у одељак All Programs, изабрати опцију Microsoft SQL Server 2008, а затим опцију SQL Server Management Studio (слика 13).

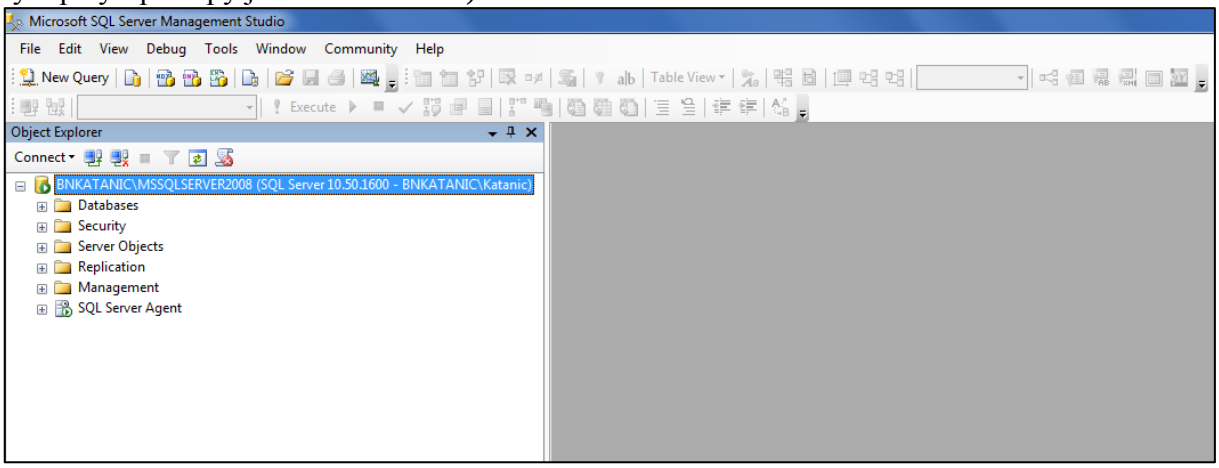


Слика 13. *Покретање SQL Servera 2008 и програма Managment Studio*

2. Појавиће се екран приказан на слици 13. који омогућава да се успостави веза с Microsoftovim SQL Serverom 2008. Ако се тип и име сервера разликују од оних понуђених на екрану, потребно је уписати одговарајући тип и име сервера, а затим бира опција Windows Authentication. Након тога се притисне дугме Connect.

Слика 14. *Успостављање везе са SQL Serverом 2008*

Пошто се успостави веза са сервером који је задат, појавиће се екран програма SQL Server Management Studio који ће се користити при раду на било којој бази података. Ако је све у реду, после пар секунди ће бити приказан *Studio* (слика 15) где са леве стране у *Object Explorer* оквиру можете видети да је извршена конекција на жељени SQL Server (име рачунара у примеру је BNKATANIC).

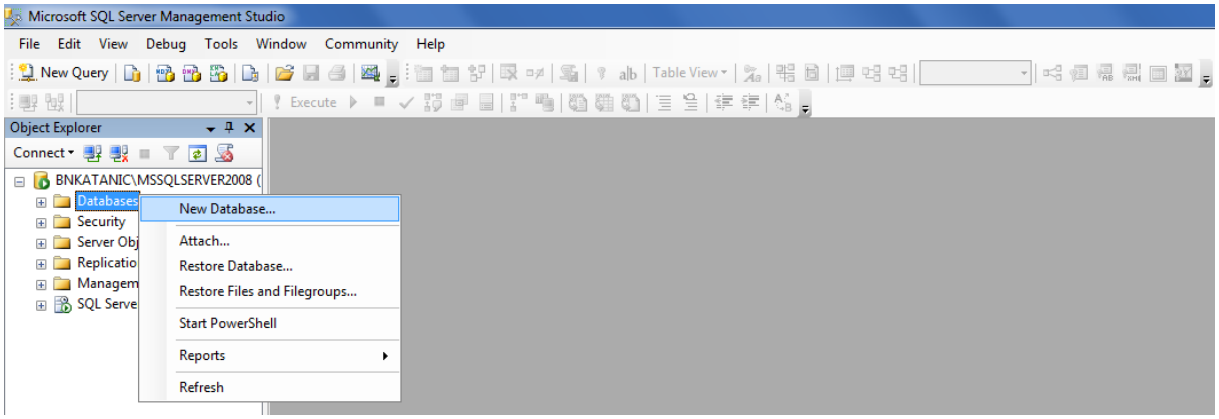


Слика 15. *Почетни екран програма*

На левој страни екрана програма SQL Server Management Studio стоји одељак *Object Explorer*. Одељак *Object Explorer* приказује објекте на серверу у облику хијерархијског стабла. На пример, то омогућава да се пронађе одређена база података, табела, колона или друга врста објекта.

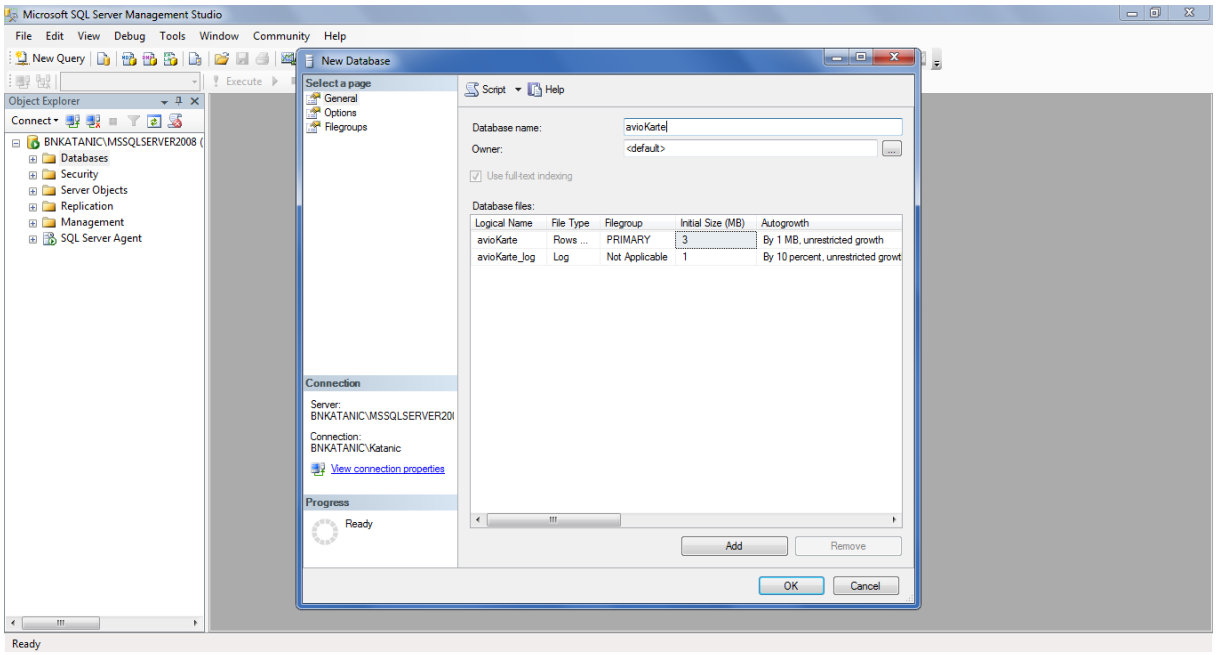
5.3. ИЗРАДА НОВЕ БАЗЕ ПОДАТАКА

Пре почетка рада с Microsoft – овим SQL сервером 2008, мора се направити база података. Да би се направила нова база података (слика 16), десним тастером миша се притисне у *Object Exploreru* чвор *Databases* и из приручног менија бира се опција *New Database*.



Слика 16. *Креирање нове базе података*

Појавиће се оквир за дијалог *New Database* (слика 17). Направиће се база података чије ће име бити авиоКарте.

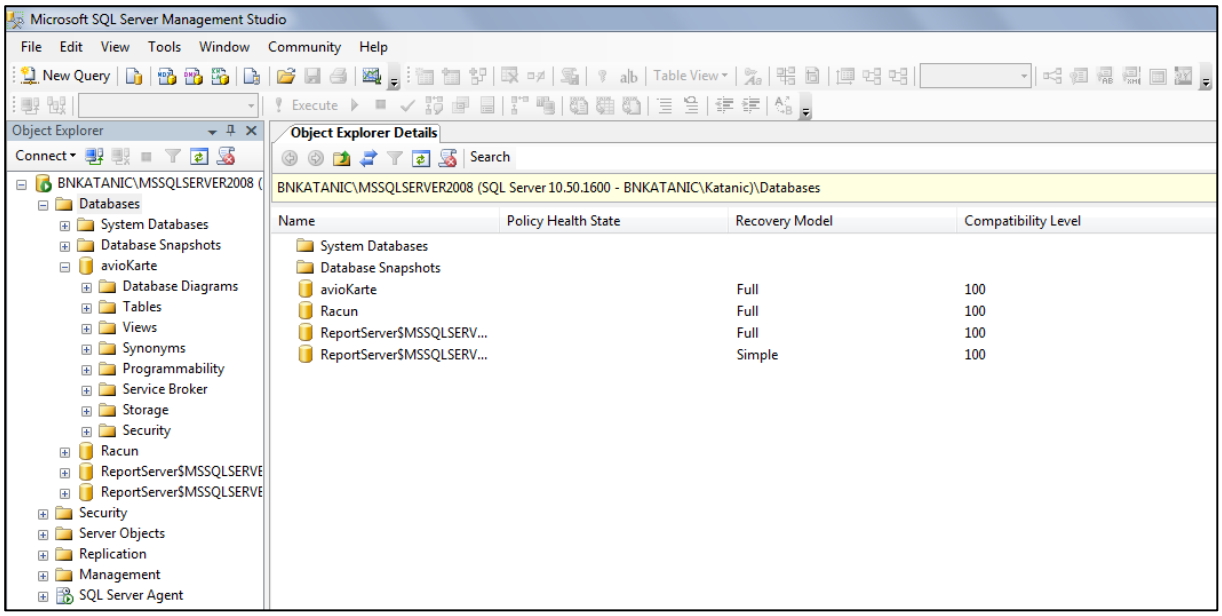


Слика 17. *Задавање имена нове базе података*

Сада је направљена база података авиоКарте. У одељку испод натписа Databases на картици Object Explorer Details, на десној страни екрана, се налази нова иконица авиоКарте која представља ту базу података.

Да би се одмах видела база података авиоКарте и у одељку Object Explorer на левој страни екрана, може бити потребно да се десним тастером миша притисне чвор Databases и да затим изабере опција Refresh.

Потом, може се отворити чвор Databases тако што ће се у Object Exploreru мишем притиснути знак + поред иконице Databases, чвор који представља базу података авиоКарте видеће се испод чвора Databases (у одељку Object Explorer на левој страни екрана). SQL Serverova база података је збирка објеката, као што су табеле, прикази и синоними, дефинисани ради подржавања активности које се одвијају над подацима у бази.



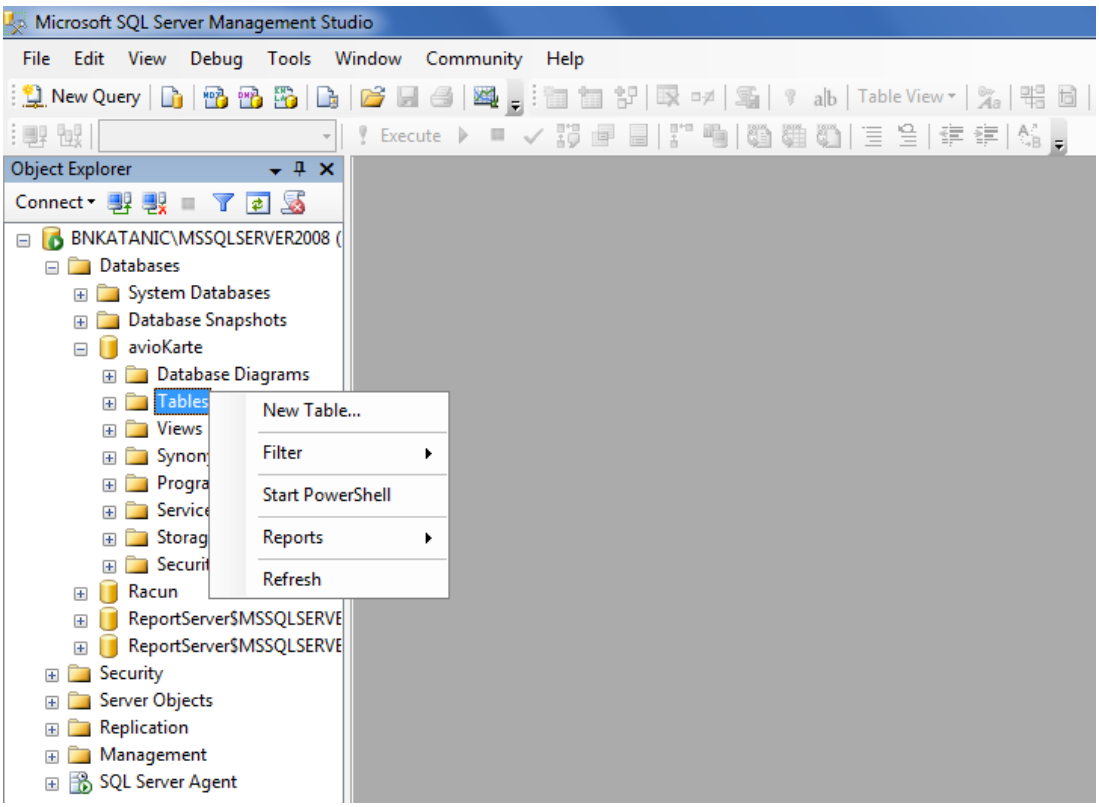
Слика 18. *Приказ објеката у нашој бази података*

5.4. ИЗРАДА ТАБЕЛА У БАЗИ ПОДАТАКА

Може се слободно рећи да је табела (енг. *Table*) основни објекат SQL Servera, као и сваке друге базе података. Табела је носилац информација и њу креира корисник у складу са својим потребама. Она представља једну логичку целину – ентитет у коме се чувају подаци.

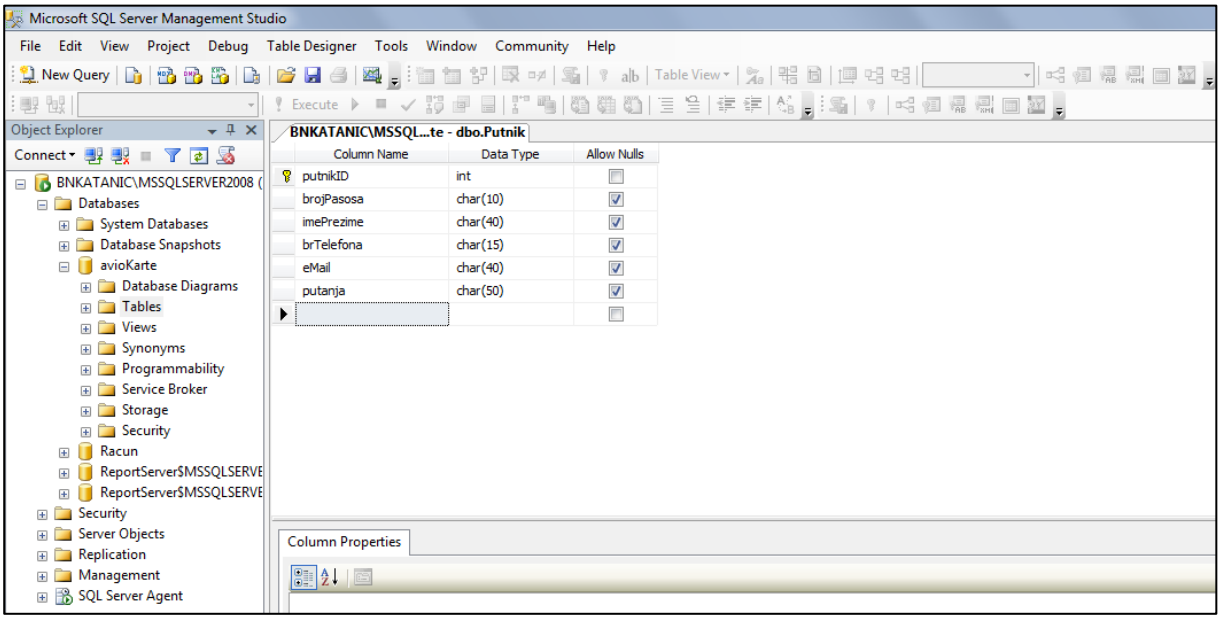
Креирање се врши тако што се десним кликом миша на чвор Tables појави опција New Table (слика 10). Кликом на ту опцију отвара се нова табела, која се састоји из три колоне: *Column Name, Data type* и *Allow Nulls*. У прву колону *Column name* уписују се називи атрибута који описују одредјени ентитет. Такође потребно је водити рачуна да бар један од атрибута мора бити кандидат за функцију примарног кључа, којим се контролише начин на који се информације повезују са другим табелама. Примарни кључ на јединствен начин дефинише запис, они су најчешће ID, типа *AutoNumber*.

Примарни кључ се поставља на једно или више поља у табели. Њихова улога је да онемогућавају понављање података у тим пољима. Њима осигуравамо јединственост свих података у пољима на која смо их поставили (један кључ = нема дупликата у закључаном пољу). Основно правило које дефинишемо приликом креирања табеле је тип атрибута. Податак који желимо да чувамо у атрибуту дефинише и његов тип. Постоје текстуални, датумски, целобројни и реални (број са децималним зарезом) типови атрибута. Заправо има их још много више, али ово су основни. Ако је за атрибут изабран датумски тип, за вредност овог атрибута је могуће искључиво уписати валидни датум. Било шта друго ће за последицу имати пријаву грешке и одбијање уноса или измене слога. Поље *Allow Nulls* nam daje могућност да одредимо који од атрибута су обавезни за унос а који не.



Слика 19. *Креирање нове табеле*

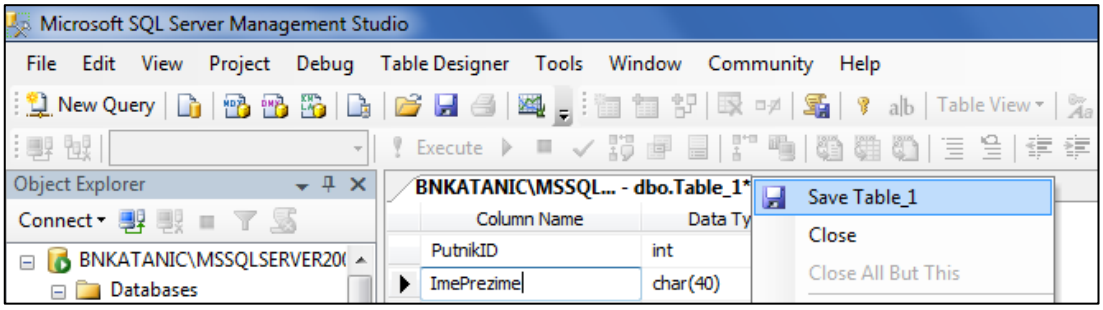
Након креирања табеле приступа се уносу података. Као основа може се користити релациони модел података, који је претходно израђен. На бази модела треба дефинисати структуру табеле и исправно изабрати тип и дужину текстуалних поља. Називи колона могу имати до 128 карактера, али то не треба злоупотребљавати јер ће каснији рад бити компликованији. Пожељно је изабрати имена која су разумљива и довољно кратка. На основу типова и дужина очекиваних података формирали смо структуру и дефинисали који подаци су обавезни за унос, а који не (слика 20).



Слика 20. *Декларација нове табеле*

Свака табела треба да има примарни кључ, односно колону чија вредност јединствено идентификује сваки слог табеле. У овом случају колона путник_ИД ће бити примарни кључ, па га треба и направити над њом. Потребно је урадити десни клик на назив колоне путник_ИД и из падајућег менија изабрати ставку *Set Primary Key*. Колона путник_ИД се од осталих разликује јер је с њене леве стране златни кључ.

Остаје још снимање ове табеле и при томе давање назива, у нашем случају Путник. У тоолбару кликом на икону дискете (слика 21), приказује се дијалог за давање имена табели. Уносимо Путник а затим кликнемо на дугме ОК. Овим је табела креирана и може да се користи за унос података. Исти поступак поновити прилликом израде осталих табела.

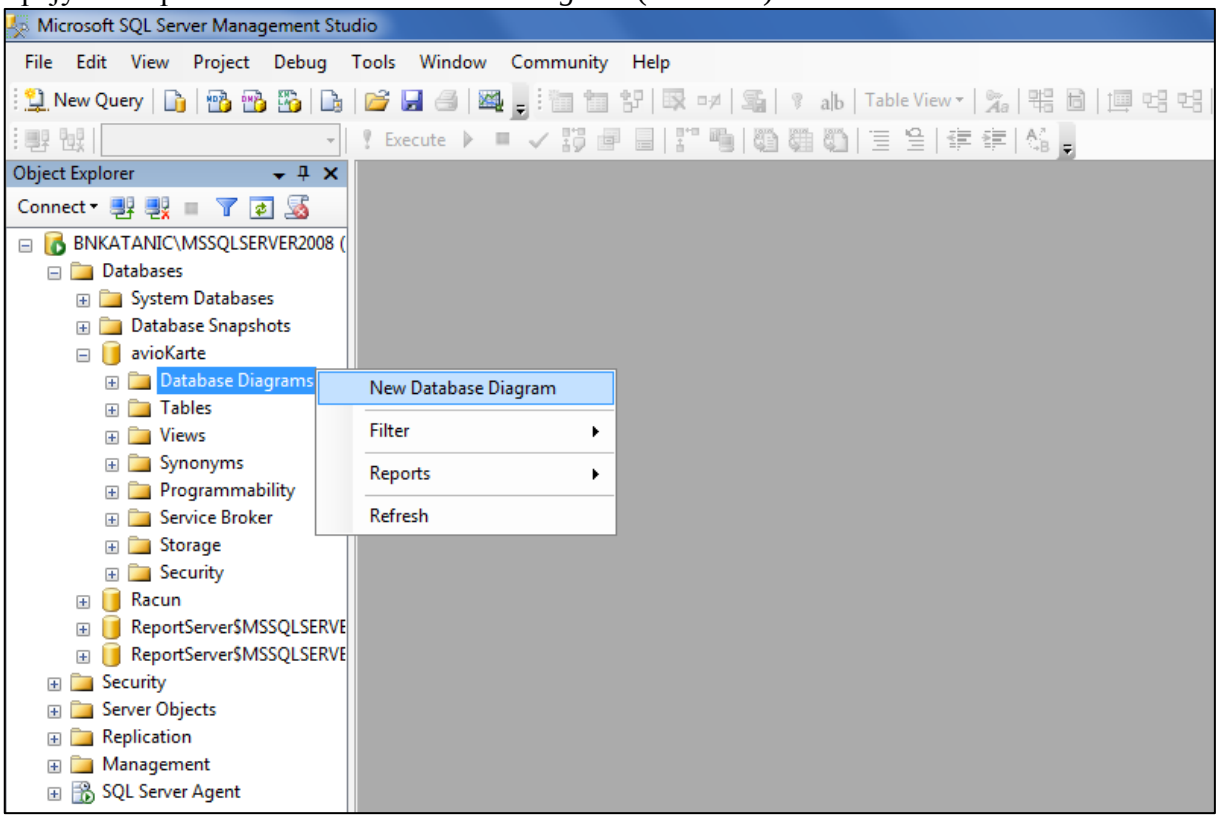


Слика 21. *Чување нове табеле*

5.5. ПОВЕЗИВАЊЕ ТАБЕЛА

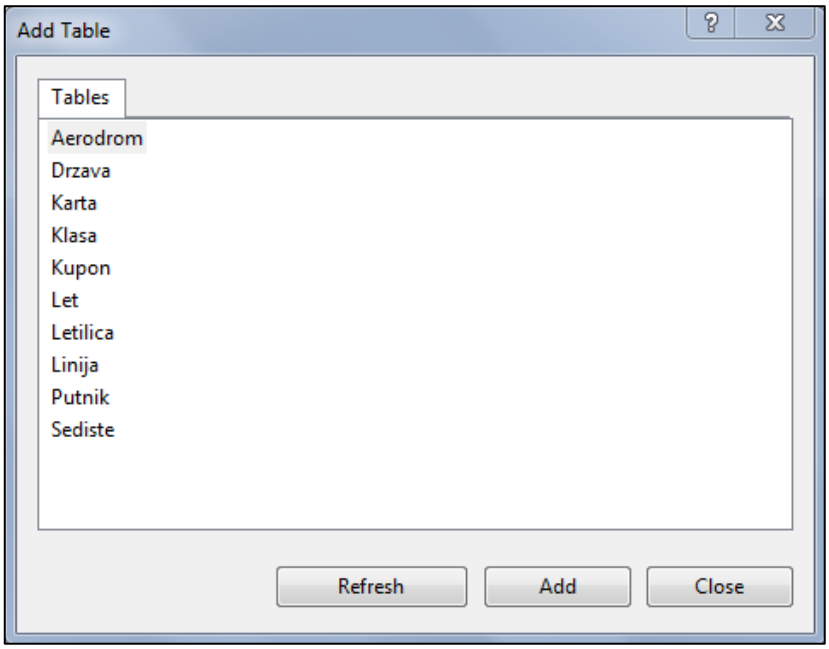
Након израде табела потребно их је повезати везом 1: Н, односно Putnik. putnikID : karta.putnikID.

Дијаграм веза креира се тако што у *Object Explorer*-у кликом на чвор *Database*, потом на базу података „авиоКарте“, а затим десним кликом изабрати ставку *Database Diagrams* и на крају изабором ставке *New Database Diagram* (слика 22).



Слика 22. *Креирање новог дијаграма за нашу базу података*

У наредном дијалог прозору треба изабрати табеле које учествују у релацији, затим кликом на дугме Add и потом на дугме Close затвара се дијалог за избор табела (слика 23).



Слика 23. *Одабир табела за дијаграм ентитета базе података*

Отворен је прозор за креирање релација између табела. У њему се налазе наше табеле које сада треба повезати помоћу *Drag and drop* начина. Релација се у бази података представља дводимензионалном табелом, где врсте одговарају појединим слоговима, а колоне атрибутима. Атрибути не морају имати тачан редослед у табели.

5.6. КРЕИРАЊЕ ПРОЦЕДУРА

У случају табеле аеродром примарни кључ се преноси два пута, што није практично за коришћење у Visual Studiju, самим тим је потребно исту табелу приказати два пута под другим именом. То се може извести десним кликом programmability->stored procedures->new stored procedures и уносом следећег кода (CREATE је приказано курзивно јер се после извршавања мења у ALTER):

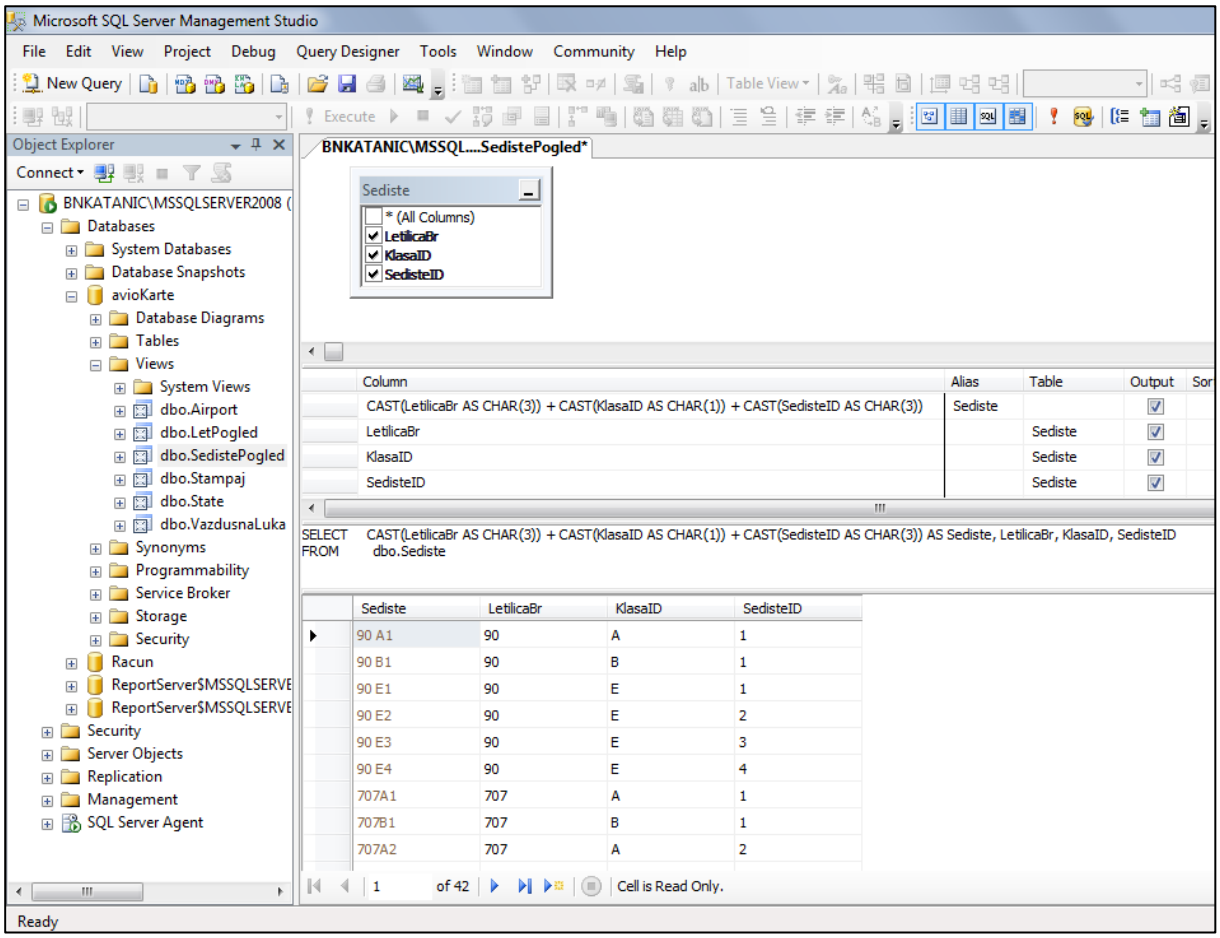
```
USE [avioKarte]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE procedure [dbo].[CMB_Aerodrom]
as
select nazivAerodroma, DrzavaID from dbo.Aerodrom
```

Ово је базични код који исписује исто што и класични начин отварања ове табеле. Он се примењује тамо где није могуће истовремено два пута учитавати исту табелу. Пример је полазни аеродром и одредисни аеродром у табели линија, не можете из исте табеле везати два comboboxa u *Visual Studiu* pa se позеже за овим принципом да би вам то било омогућено.

5.7. КРЕИРАЊЕ ПОГЛЕДА (SQL VIEW)

Када имамо ситуацију преноса комплексног кључа сачињеног од више примарних кључева у *Visual Studiu* потребно је користити SQL *Поглед* (*View*). Наиме сам combobox у *Visual Studiu* не дозвољава управљање више од две колоне и то прва за приказ и друга за вредност. Путем SQL *погледа* ви можете омогућити да две или више колона постану једно, а да у другом SQL погледу се оне врате и трансформишу у више колона.

Као пример користимо табелу седисте чији се примарни кључ састоји из три примарна кључа (назив авиона, класа седиста и број седишта)



Слика 24. *Креирање погледа за табелу Седисте*

Табела седиште (слика 24.) има три колоне све три су примарни кључ, ниједна није идентификатор, већ све три колоне заједно чине идентификатора. Нама SQL поглед омогућује да уз све три колоне имамо четврту која обједињује све три. Све колоне су из исте табеле самим тим `FROM dbo.Sediste.`

```
SELECT CAST(LetilicaBr AS CHAR(3))+CAST(KlasaID AS CHAR(1))+CAST(SedisteID AS CHAR(3)) AS Sediste,LetilicaBr,KlasaID,SedisteID
FROM dbo.Sediste
```

SELECT CAST(LetilicaBr AS CHAR(3))+CAST(KlasaID AS CHAR (1)) AS Sediste се односи на спајање у овом случају колоне LetilicaBr (целобројна вредност) конвертовану као реч од три слова и колоне KlasaID конвертовану као карактер у једну колону која ће се звати седиште и имати 4 карактера. Знак + је тај који их спаја и при спајању они морају имати дефинисано да постају иста променљива (било *CHAR, INT, BLOB*...).

6. КОНТРОЛА ТОКА ПРОГРАМА

Колико год је могуће упростити сложен апликативни модел који обједињује базу података, слојеве windows платформе и c# програмски језик (директни наследник c++), биће учињено у овом поглављу.

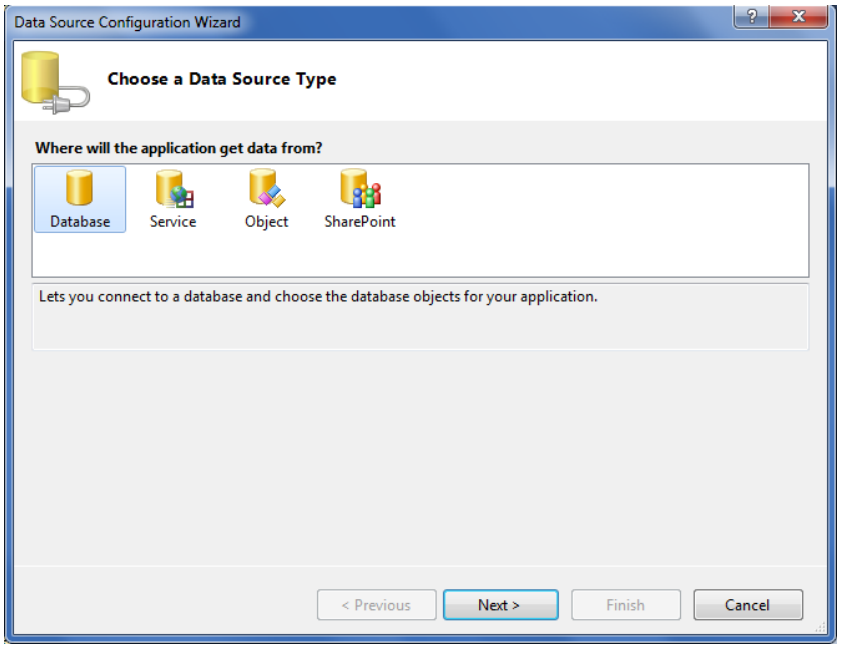
Базе података представљају виши ниво рада са подацима у односу на класичне програмске језике. Реч је о технологији која је настала са намером да се уклоне слабости традиционалне "аутоматске обраде података" из 60-тих и 70-тих година 20. века. Та технологија осигурала је већину продуктивност, квалитет и поузданост у развоју апликација које се своде на чување и претраживање података у рачунару.

База података је скуп међусобно повезаних података, сачуваних у спољној меморији рачунара. Подаци су истовремено доступни разним корисницима и апликационим програмима. Убацивање, измена, читање и брисање података у бази су методе које вршимо над податцима, подрестом неке апликације направљене за рад на тој бази података. Постоји много врста софтвера који се могу применити за израду база података, неки од најкоришћених софтвера база података су SQL и Microsoft Access.

6.1. ОДАБИР ИЗВОРА ПОДАТАКА

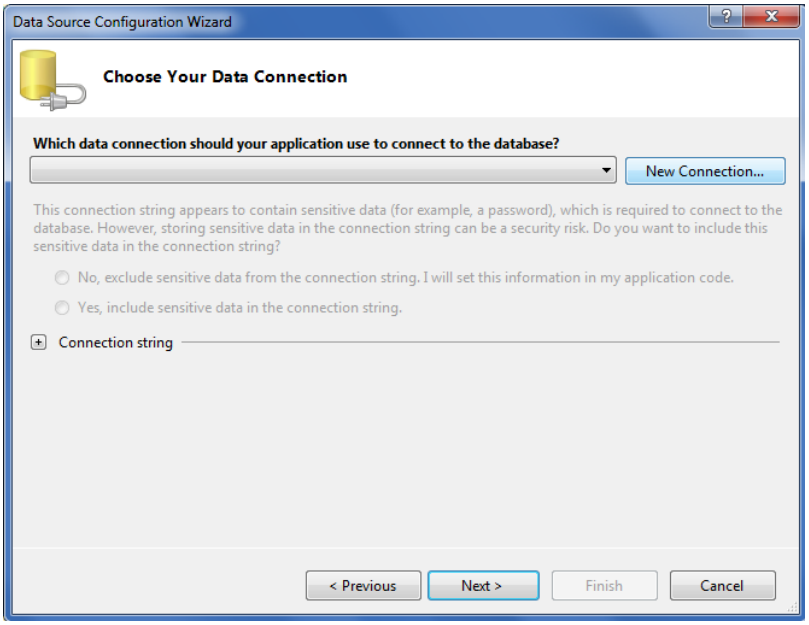
Да бисмо могли мењати податке у програму они морају бити чувани у формату који им то омогућује. Сам *.exe (executable- извршна датотека) фајл не омогућује мењање неког садржаја. Помоћу програмског језика, а без базе можете унети неке податке (нпр бројеве телефона), међутим ако желите било шта да проширите, морате поново приступати компајлеру. Да бисмо апликацији омогућили сталну промену података морамо јој дати извор са којег преузима податке и на који их шаље. Може бити *.txt, *.xls, *.mdb, *.mdf, *.ldf и још много формата за складиштење података.

За потребе овог рада користи се база у SQL серверу 2008 сачињена од 10 табела, 6 погледа (не приказују се сви) и 1 процедура. У фајл-менију *Visual Studia* имамо картицу *data* која служи за управљање подацима. Кликом на исту она се излистава (*roll down*) и идемо на *add new data source*. Појавиће се прозор као на слици 25.



слика25. *Избор базе података*

Идемо на нехт појавиће се прозор веома сличан овом, бирамо *Dataset* па опет next. Одабиром New Connection можемо бирати тип базе података (Excel, Access, SQL Server, mySql и слично) као што је приказано на слици 26.



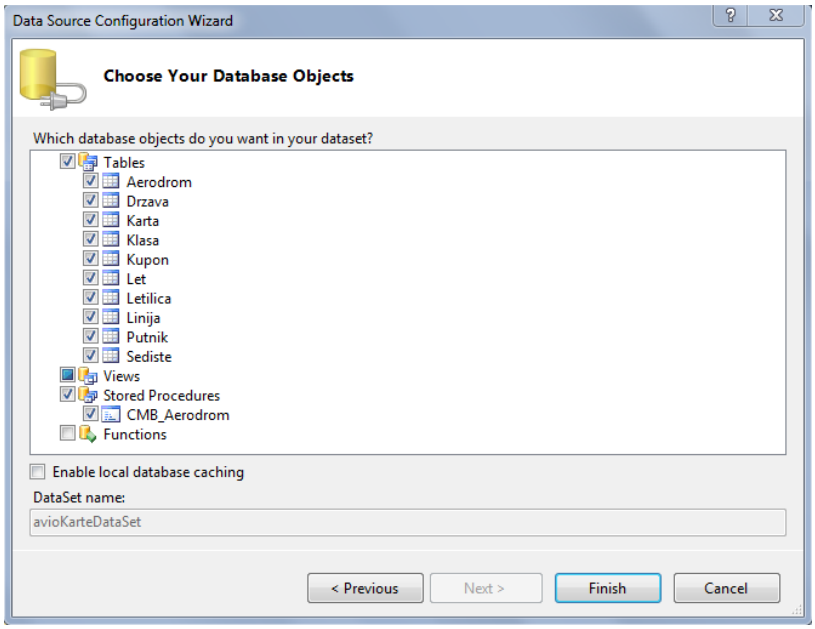
слика26. *Избор нове конекције*

Када завршите све кораке имате избор тестирања конекције и конекциони стринг који треба да копирате у нотепад за накнадну употребу. У нашем случају Dataset name = avioKarteDataSet, ConnectionString:

```
Data Source=BNKATANIC\MSSQLSERVER2008;Initial Catalog=avioKarte;Integrated Security=True
```

Постоји могуцност евентуалног постављања корисничког имена и шифре за конекцију на базу ради још веће сигурности. По завршетку нам се отвара могућност провере конекције (test connection) да би се уверили да веза програма са базом тече неометано.

Након форимирања конекционог стринга, следеће што је потребно урадити је одабир објеката базе а то су табеле, погледи, процедуре и сл. Објекте које желимо да одаберемо чекирамо у квадратићу поред назива табеле као на слици 27. и притиснемо Finish.



слика27. *Избор објеката у бази*

У нашем случају нису одабрани сви погледи који се користе у самом серверу, а који овде нису неопходни, док су табеле и процедуре одабране све. Сви одабрани објекти ће се појавити у *Data Sources* пољу (слика 10, поље означено бројем 6). То је отприлике детаљан поступак остваривања везе апликације са базом података. Библиотека неопходна за рад са базом:

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

Реченица која остварује конекцију у нашем случају (има два адресна раздвајача "\"):

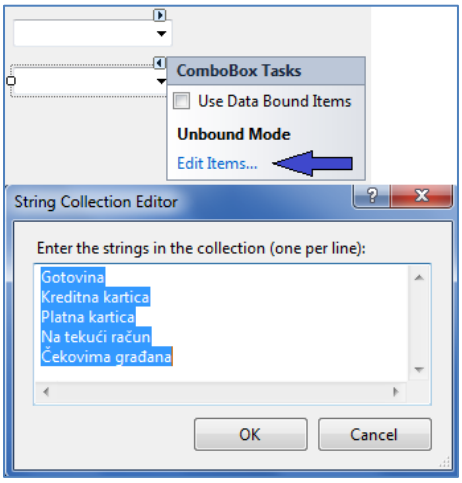
```
SqlConnection konekcija = new SqlConnection ("Data
Source=BNKATANIC\\MSSQLSERVER2008;Initial Catalog=avioKarte;Integrated Security=True");
```

Цео текст се пише у истом реду и ова команда се позива на свакој форми која има потребу за везом са sql сервером или неком другом базом података.

6.2. ПАДАЈУЋЕ ЛИСТЕ (COMBOBOX)

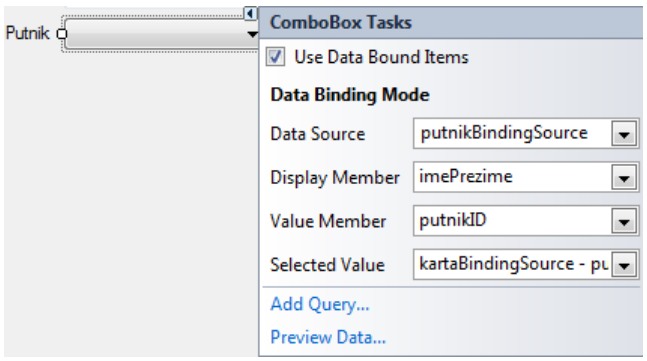
За потребе олакшаног приступа великој количини података која се преноси кључем из једне табеле у другу користи се падајућа листа која ће приказати (излистати) одређене колоне из претходне табеле. Нпр ако имате табелу ученик са именом и презименом, датумом рођења, полом и слично и другу табелу оцене из музичког вама треба падајућа листа која ће излистати имена и презимена свих ученика у табели предмети како бисте ви одабрали једног. Из toolboxa превуците (*drag and drop*) combobox.

Самочитљива падајућа листа (*Unbound Mode Combobox*) се креира тако што се након превлачења цомбобоха кликне стрелица, која затим мења смер. Затим кликнете на *Edit Items* и додате редове текста који су вам потребни за ваш combobox (слика 28).



слика28. *Unbound Mode combobox*

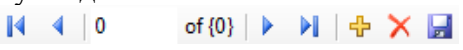
Падајућа листа са подацима (*Data Bound mode Combobox*) се користи у случајевима где се одређени податак преноси из једне табеле у другу. Он самим тим не функционише исто као *unbound mode* који има само један излаз података. Падајућа листа са подацима има два излаза први је испис а други вредност. Враћамо се на почетак овог поглавља где смо узели пример две табеле ученик и предмет (нпр. музичко). Табела ученик има целобројни примарни кључ (1,2,3...) и колону име (Марко, Ивана, Сузана). Када преносимо ученика Ивана у табелу музичко ми преносимо вредност "2", а реч "Ивана" је испис јер се налази у истом реду са бројем 2 у табели ученик. На слици 29 је приказан прост пример коришћења combobox са подацима.

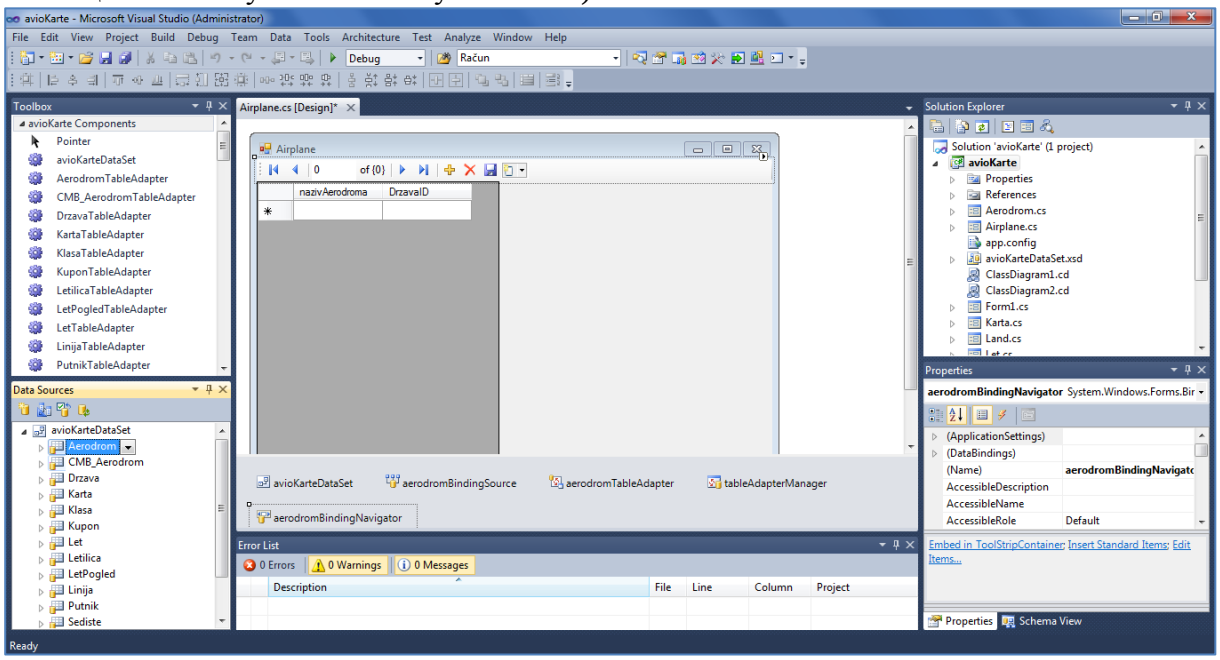


слика29. *Data Bound Mode combobox*

Да се опет вратимо на наш пример у табелама ученик-предмет, ако ученика ивану пребацујемо у табелу предмет, *Data Source* је табела ученик, *Displey Member* је име, у овом случају Ивана, Value Member = "2"(ucenikID у табели ученик), а Selected Value = "2" (ucenikID у табели предмет). Сатим тим визуелно ће нам бити исписани Марко, Ивана, Сузана... а меморисни 1,2,3...

6.3. МРЕЖА ПОДАТАКА (DATAGRIDVIEW)

Да бисмо приказали више редова у табели одједном и омогућили њихово мењање, користи се приступ најсличнији datasheet из MS Access-а. Самим превлачењем једне табеле из картице *Data Sources* у форму (*Form Design*) креираће једну *Мрежу података* и једну *Навигацију* за ту мрежу података. Сличан облик навигације мреже података се среће у формама MS Accessa.  Као што видимо он се састоји од неколико генеричких дугмића који служе као навигација за редове: ▌◄ први, ◄претходни, ► наредни, ► ▌ последњи, ✚ додај ред, ✘ уклони ред и сачувај изменене (без активације ове иконице неће се чувати измене у табелама).

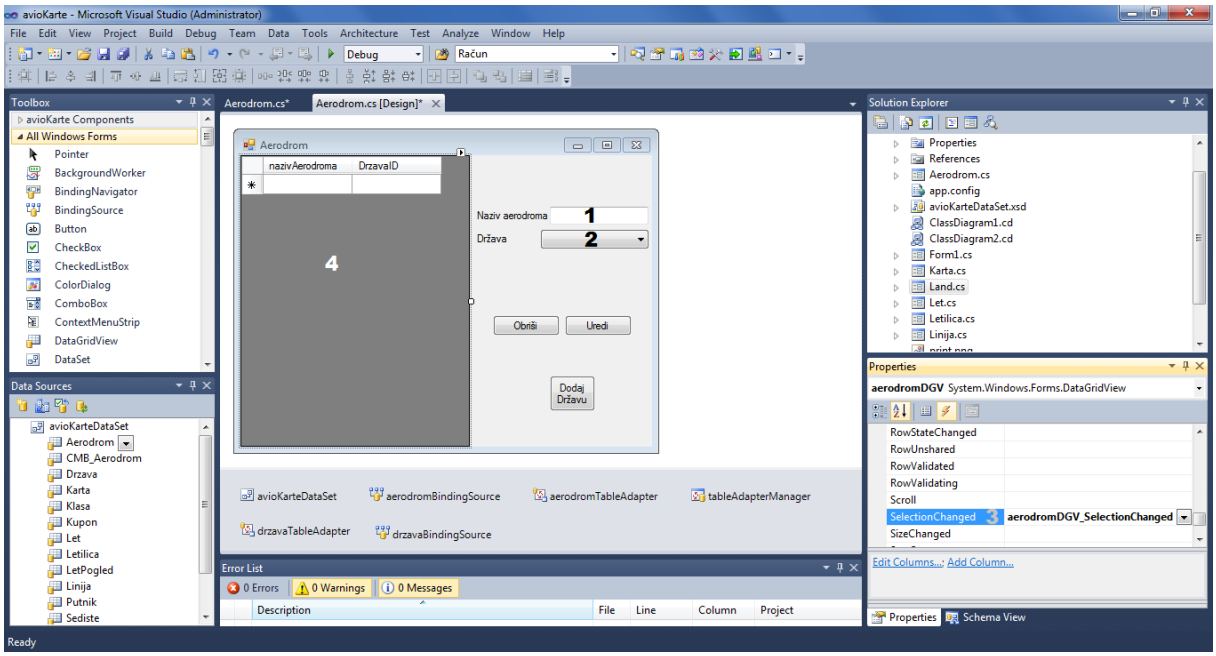


Слика30. *Мрежа података и навигација*

У мрежи података измене можете уносити ручно налик на MS Excel и кликом на иконицу Save (floppy disc) оне ће бити сачуване. Коришћење мреже података не захтева конекциони стринг (страна 34).

6.4. ПРОСЛЕЂИВАЊЕ ПОДАТАКА ИЗ МРЕЖЕ ПОДАТАКА

Прослеђивање података из мреже података у компоненте тоолбоха се изводи ради прегледнијег и лакшег уређивања истих. За демонстрацију користићемо најпростију табелу у бази, то је свакако Аеродром са само два атрибута (назив и држава).



Слика31. *Промена одабира у мрежи података*

Посматрајте слику 31, бројем 1 је представљен textbox који прима податак из прве колоне Аеродром мреже података(број 4), а бројем 2 је представљен цомбобох који прима податак из друге колоне Аеродром мреже података. У другој колони Аеродром мреже података се налази трословни регистарски знак државе: СРБ, ХРВ, БИХ. Combobox Држава ће примити тај податак, али ће исписати Србија, Хрватска, Босна и Херцеговина (објашњено у поглављу 7.2). Са десне стране броја 3 имате падајућу листу која ће код вас вероватно бити празна. Двокликните на SelectionChanged (маркирано плаво на слици 31) и отвори ће се прозор за уређивање кода.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace avioKarte{
   public partial class Aerodrom : Form{
      public Aerodrom() {
         InitializeComponent();}
```

Горњи део кода представља иницијализацију, не треба му давати велики значај. Уочава се да је у библиотекама (using) прескочен SQL клијент јер се све изводи преко мреже података па нема потребе за тим.

```
private void aerodromBindingNavigatorSaveItem_Click(object sender, EventArgs e){
   this.Validate();
   this.aerodromBindingSource.EndEdit();
   this.tableAdapterManager.UpdateAll(this.avioKarteDataSet);}
```

Овај део кода је аутоматски генерисан након убацивања Мреже података(*DataGridView*), он омогућује чување измена у табелама.

```
private void Aerodrom_Load(object sender, EventArgs e) {
   avioKarteDataSet.EnforceConstraints = false;
   this.drzavaTableAdapter.Fill(this.avioKarteDataSet.Drzava);
   this.aerodromTableAdapter.Fill(this.avioKarteDataSet.Aerodrom); }
```

Aerodrom__Load->све унутар овог кода се дешава чим се учита форма аеродром.
avioKarteDataSet.EnableConstraints=false->ова команда служи да избегне неке провере у бази и да омогући лакше ажурирање базе.
drzavaTableAdapter.Fill ->Табела држава се учитава због comboboxa Држава.
aerodromTableAdapter.Fill ->Табела аеродром се учитава због Аеродром мреже података.

```
private void aerodromDGV_SelectionChanged(object sender, EventArgs e){
 if (aerodromDGV.SelectedRows.Count > 0){
     foreach (DataGridViewRow row in aerodromDGV.SelectedRows){
        nazivAerodromaTB.Text = Convert.ToString(aerodromDGV.SelectedRows[0].Cells[0].Value);
        drzavaCMB.Text = Convert.ToString(aerodromDGV.SelectedRows[0].Cells[1].Value); }}}
```

aerodromDGV_SelectionChanged->Команде које се извршавају кликом на врсту у аеродром мрежи података (аеродромДГВ). Одабир врсте се изводи кликом на стрелицу/звездицу у мрежи података (слика 31, број 4, dataGridView). Прва три реда кода се преписују и усклађују потребама и називу саме мреже података (у овом случају аеродромДГВ). Овај део кода шаље податке из мреже података у toolbox контроле (textbox, combobox, date/time picker...)

nazivAerodromaTB.Text = Convert.ToString(aerodromDGV.SelectedRows[0].Cells[0].Value);-> Овде textbox nazivAerodromaTB прима податак из прве колоне аеродром мреже података.
drzavaCMB.Text = Convert.ToString(aerodromDGV.SelectedRows[0].Cells[1].Value); -> Овде combobox drzavaCMB прима податак из друге колоне аеродром мреже података.

```
private void ukloniGMB_Click(object sender, EventArgs e){
   foreach (DataGridViewRow item in this.aerodromDGV.SelectedRows){
      aerodromDGV.Rows.RemoveAt(item.Index);
      this.tableAdapterManager.UpdateAll(this.avioKarteDataSet);}}
```

Ови делови кода омогућују брисање врсте изабране на исти начин као у претходном одељку.

```
private void urediGMB_Click(object sender, EventArgs e){
   try{
      aerodromDGV.BeginEdit(true);
      aerodromDGV.SelectedRows[0].Cells[0].Value = nazivAerodromaTB.Text;
      aerodromDGV.SelectedRows[0].Cells[1].Value = drzavaCMB.SelectedValue;
      aerodromDGV.EndEdit();
      this.Validate();
      this.aerodromTableAdapter.Update(this.avioKarteDataSet.Aerodrom); }
   catch { } }
```

Овде имамо обрнут процес од слања, а то је примање измењених података. try/catch метод се користи да се избегне крах програма у случају да неке измене не буду прихваћене. Кликом на дугме уреди биће прихваћене измене у comboboxuy и textboxu за назначени ред у ДГВ-у.

```
private void button1_Click(object sender, EventArgs e) {
   Land otvori = new Land();
   otvori.Show();}}}
```

Овај део кода отвара форму држава за додавање нових.

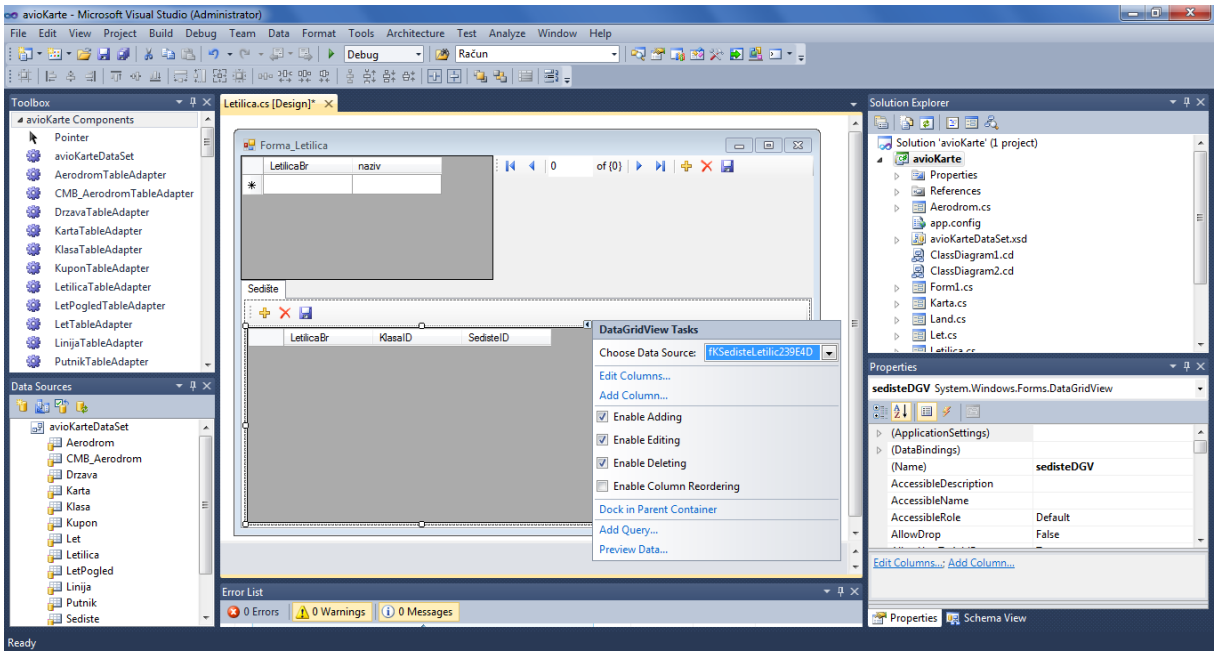
6.5. ЗАВИСНИ ЕНТИТЕТ МРЕЖЕ ПОДАТАКА

У неким случајевима имате зависни ентитет. Зависни ентитет је описан самим собом и својим газдом од којег зависи. Најпростији пример је рачун – ставке рачуна. Нпр. имамо два рачуна. Један је био купљен у уторак, други у петак. Имају исту садржину (5 јаја, 2 хлеба, 1 кечап).

СтавкаРачуна	Артикал	Комада	СтавкаРачуна	Артикал	Комада
1	Јаја	5	1	Јаја	5
2	Хлеб	2	2	Хлеб	2
3	Кечап	1	3	Кечап	1

Лева и десна табела су исте, СтавкаРачуна која овде представља примарни кључ није довољна да идентификује табелу и могућа је редуданса. Постоје два начина да се проблем реши. Први је мање елегантан, да ставке рачуна у десној табели преузму слободне бројеве (4,5,6). То решење не осликава прави живот јер на правом рачуну (када купујете нешто у продавници) прва ставка увек узима број 1. Друго решење је увођење табеле вишег хијерархијског нивоа (енг.*Master Table*). Самим тим примарни кључ сваке ставке је примарни кључ табеле рачун (Рачун_ИД) + примарни кључ табеле ставка (СтавкаРачуна). Ако кажемо да је левој табели РацунИД 205, а десној 209 њихови примарни кључеви су: 2051,2052,2053,2091,2092,2093. Они су настали спајањем два примарна кључа (енг.*ComplexKey*).

Да би се остварио зависни ентитет у Visual Studiju (слично као *MS Access subform*), потребно је убацити обе мреже података-надређену и подређену у исту форму. Већ смо рекли да превлачењем табеле из картице *Data Sources* креира мрежу података за ту табелу (datagrid). Сам програм јој аутоматски додељује извор података (саму табелу). У случају рачун-ставка рачуна, мрежа података за рачун би као извор имала табелу рачун, а ставкарачуна би уместо своје табеле као извор користила кључ који се преноси између те две табеле.

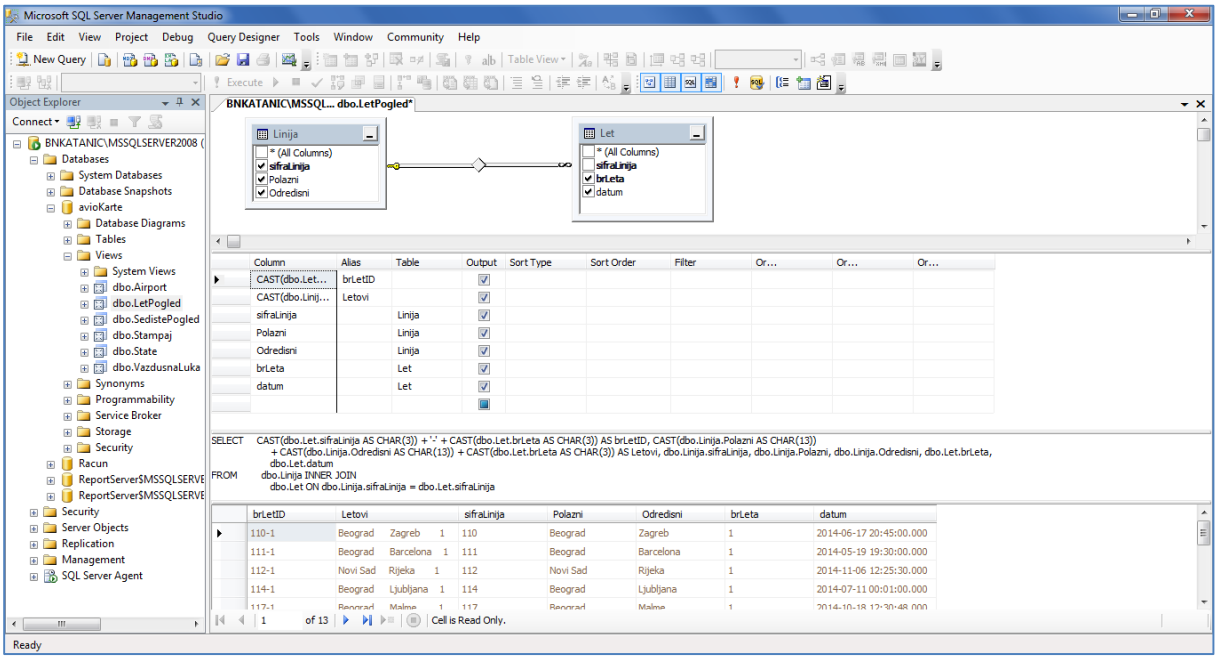


Слика32. *Зависни ентитет у мрежи података*

Форма летилица има мрежу података летилица и таб (картицу) седиште са мрежом података седиште. Извор података за DataGridView Летилица остаје исте (дифолт), док у случају седишта се користи пренесени кључ (fkSedisteLetilic239E4D).

6.6. MULTICOLUMN COMBOBOX

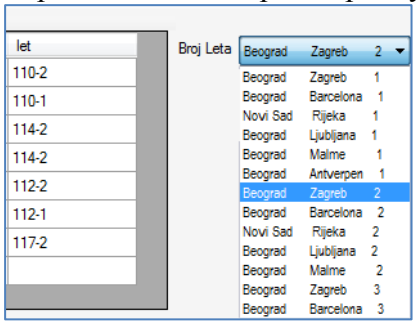
У неким ситуацијама када се преноси сложен кључ (енг.*complex key*), потребно је приказати више колона у comboboxu у истом тренутку. Visual студио нема ту опцију, самим тим све се изводи преко SQL погледа (SQL Views). Дакле, combobox може приказати само једну колону за идентификацију (Combobox.SelectedValue) и једну колону за испис (Combobox.Text). SQL View омогућује спајање више колона у једну, као и креирање коплексног кључа и спајање више стрингова у исти текст. На пример у овој бази смо у исти стринг стављали име и презиме путника. Уколико сте у вашој бази одвојили име и презиме путника, SQL View вам омогућује да их спојите у исти стринг.



Слика33. *Комплексан SQL поглед*

```
SELECT    CAST(dbo.Let.sifraLinija AS CHAR(3)) + '-' + CAST(dbo.Let.brLeta AS CHAR(3)) AS brLetID,
CAST(dbo.Linija.Polazni AS CHAR(13))  + CAST(dbo.Linija.Odredisni AS CHAR(13)) +
CAST(dbo.Let.brLeta AS CHAR(3)) AS Letovi, dbo.Linija.sifraLinija, dbo.Linija.Polazni,
dbo.Linija.Odredisni, dbo.Let.brLeta, dbo.Let.datum //sve u istom redu
FROM      dbo.Linija INNER JOIN
          dbo.Let ON dbo.Linija.sifraLinija = dbo.Let.sifraLinija
```

Овај део кода преставља спајање више стрингова у један. Назив полазног аеродрома, назив одредисног аеродрома и број лета су спојени у исти стринг и они ће служити за испис (кол.2). Шифра линије (која одговара рути полазног и одредишног аеродрома) заједно са бројем лета ће чинити примарни кључ. Остале колоне су попуњне да би у неком другом погледу програм успео да врати ове информације. Самим тим у Visual studiu combobox.SelectedValue=110-1, a combobox.Text=Beograd Zagreb 1, самим тим коринсик неће имати проблем да одабере потребну руту.



Слика34. *Multicolumn combobox у Visual studiu*

7. КОД АПЛИКАЦИЈЕ

У овом поглављу је приказан и коментарима објашњен код у формама, без дизајнера који је самогенерисан. Форма за табелу аеродром је већ приказана у поглављу 7.4. и због тога се прескаче. Такође се изоставља форма летилица која поседује махом генерички код. У мрежи података се за класу уместо textboxa користи combobox. Можемо почети одмах са главном формом (form1.cs). Код свих форми се учитавају исте библиотеке на почетку самог програма и оне неће бити приказане у коду:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient; //za rad sa sql bazom podataka
using System.IO; //za otvaranje/čuvanje fajlova
using System.Net.Mail; //za konverziju stringa u e-mail
```

7.1. ГЛАВНА ФОРМА (Form1.CS)

```
namespace avioKarte{
  public partial class Form1 : Form{
    public Form1(){
      InitializeComponent();}

    private void Form1_Load(object sender, EventArgs e){}

    private void putnikDGM_Click(object sender, EventArgs e){
      Putnik otvori = new Putnik();
      otvori.Show();} //funkcija otvara formu Putnik

    private void letilicaDGM_Click(object sender, EventArgs e){
      Letilica open = new Letilica();
      open.Show();} //funkcija otvara formu Letilica

    private void button5_Click(object sender, EventArgs e){
      Aerodrom openAirport = new Aerodrom();
      openAirport.Show();} //funkcija otvara formu aerodrom

    private void LinijaDGM_Click(object sender, EventArgs e){
      Linija openLin = new Linija();
      openLin.Show();} //funkcija otvara formu linija

    private void kartaDGM_Click(object sender, EventArgs e){
      Karta opencard = new Karta();
      opencard.Show();} //funkcija otvara formu karta

    private void printDGM_Click(object sender, EventArgs e){
      Let printcard = new Let();
      printcard.Show();} //funkcija otvara formu za čuvanje

    private void Timer_Tick(object sender, EventArgs e){
      label4.Text = DateTime.Now.ToLongTimeString();
      label2.Text = DateTime.Now.ToLongDateString();}}}//prikazuje datum/vreme, prevučen timer iz toolbox-a

    private void infoBTN_Click(object sender, EventArgs e){
      MessageBox.Show("Tribalia Airways\nul.Kraljice Marije 14 Belgrade\nRepublic of Serbia\nKontakt:
      (+381)11/22-34-465\nweb: www.tribalia.rs", "Informacije o kompaniji", MessageBoxButtons.OK,
      MessageBoxIcon.Asterisk); }// ispisuje informacije o firmi (kutija za poruke)
```

7.2. ФОРМА КАРТА И ПОДФОРМА КУПОН (Karta.CS)

```
namespace avioKarte{
  public partial class Karta : Form {
    public Karta(){
      InitializeComponent();}

    private void Kupon_Load(object sender, EventArgs e){
      this.sedistePogledTableAdapter.Fill(this.avioKarteDataSet.SedistePogled);
      this.kuponTableAdapter.Fill(this.avioKarteDataSet.Kupon);
      this.letPogledTableAdapter.Fill(this.avioKarteDataSet.LetPogled);
      avioKarteDataSet.EnforceConstraints = false;
      this.letTableAdapter.Fill(this.avioKarteDataSet.Let);
      this.putnikTableAdapter.Fill(this.avioKarteDataSet.Putnik);
      this.kartaTableAdapter.Fill(this.avioKarteDataSet.Karta);
      this.kartaTableAdapter.Fill(this.avioKarteDataSet.Karta);}

    private void kartaBindingNavigatorSaveItem_Click(object sender, EventArgs e) {
      this.Validate();
      this.kartaBindingSource.EndEdit();
      //this.sedistePogledBindingSource.EndEdit();
      this.tableAdapterManager.UpdateAll(this.avioKarteDataSet);}

    private void kartaDataGridView_SelectionChanged(object sender, EventArgs e) {
      if (karteDGV.SelectedRows.Count > 0) {
        foreach (DataGridViewRow row in karteDGV.SelectedRows){
          try{
            karteDGV.Refresh();
            kartaIDTB.Text = Convert.ToString(karteDGV.SelectedRows[0].Cells[0].Value);
            IzdatoDTP.Text = Convert.ToString(karteDGV.SelectedRows[0].Cells[1].Value);
            float val = Single.Parse(Convert.ToString(karteDGV.SelectedRows[0].Cells[2].Value));
            cenaTB.Text = val.ToString("N2"); //format za decimalni zapis (2 decimalna mesta)
            PutnikCMB.Text = Convert.ToString(karteDGV.SelectedRows[0].Cells[3].Value);
            PlatiCMB.Text = Convert.ToString(karteDGV.SelectedRows[0].Cells[4].Value);
            cardNoTB.Text = Convert.ToString(karteDGV.SelectedRows[0].Cells[5].Value);
            RutaCMB.Text = Convert.ToString(karteDGV.SelectedRows[0].Cells[6].Value);}
          catch { } }}}  //ova funkcija šalje podatke iz mreže podataka u textbox, combobox...

    private void IzmeniGMB_Click(object sender, EventArgs e) {
      try {
        karteDGV.BeginEdit(true);
        karteDGV.SelectedRows[0].Cells[1].Value = IzdatoDTP.Value.Date;
        karteDGV.SelectedRows[0].Cells[2].Value = Single.Parse(cenaTB.Text.ToString());
        karteDGV.SelectedRows[0].Cells[3].Value = Int32.Parse(PutnikCMB.SelectedValue.ToString());
        karteDGV.SelectedRows[0].Cells[4].Value = PlatiCMB.Text;
        karteDGV.SelectedRows[0].Cells[5].Value = cardNoTB.Text;
        karteDGV.SelectedRows[0].Cells[6].Value = RutaCMB.SelectedValue;
        karteDGV.EndEdit();
        this.Validate();
        this.kartaTableAdapter.Update(this.avioKarteDataSet.Karta);}
      catch { } } //uređuje mrežu podataka izmenama u textbox, comboboxu...

    private void rmvGMB_Click(object sender, EventArgs e){
      foreach (DataGridViewRow item in this.karteDGV.SelectedRows) {
        try {
          karteDGV.Rows.RemoveAt(item.Index);
          this.tableAdapterManager.UpdateAll(this.avioKarteDataSet);}
        catch { } }}} //za brisanje izabranog reda u mreži podataka
```

7.3. ФОРМА ЗА ДОДАВАЊЕ ДРЖАВА (Land.CS)

```
namespace avioKarte{
   public partial class Land : Form{
      SqlConnection konekcija = new SqlConnection("Data Source=BNKATANIC\\MSSQLSERVER2008;
      Initial Catalog=avioKarte;Integrated Security=True");//u istom redu sa konekcija-connectcionstring
      SqlCommand komanda = null;
      public Land(){
         InitializeComponent();}
      private void dodajGumb_Click(object sender, EventArgs e){
         komanda = new SqlCommand("insert into Drzava(DrzavaID,nazivDrzave) values
         (@DrzavaID,@nazivDrzave)", konekcija);//u istom redu sa komanda
         SqlParameter DrzavaID1 = new SqlParameter("@DrzavaID", SqlDbType.Char);
         SqlParameter nazivDrzave1 = new SqlParameter("@nazivDrzave", SqlDbType.Char);
         DrzavaID1.Value = regMTB.Text;
         nazivDrzave1.Value = stateNameTB.Text;
         komanda.Parameters.Add(DrzavaID1);
         komanda.Parameters.Add(nazivDrzave1);
         try{
            konekcija.Open();
            komanda.ExecuteNonQuery();
            konekcija.Close();}
         catch {}
         regMTB.Clear();
         stateNameTB.Clear();
         Land_Load(sender, e);}
      private void Land_Load(object sender, EventArgs e) { }}}
```

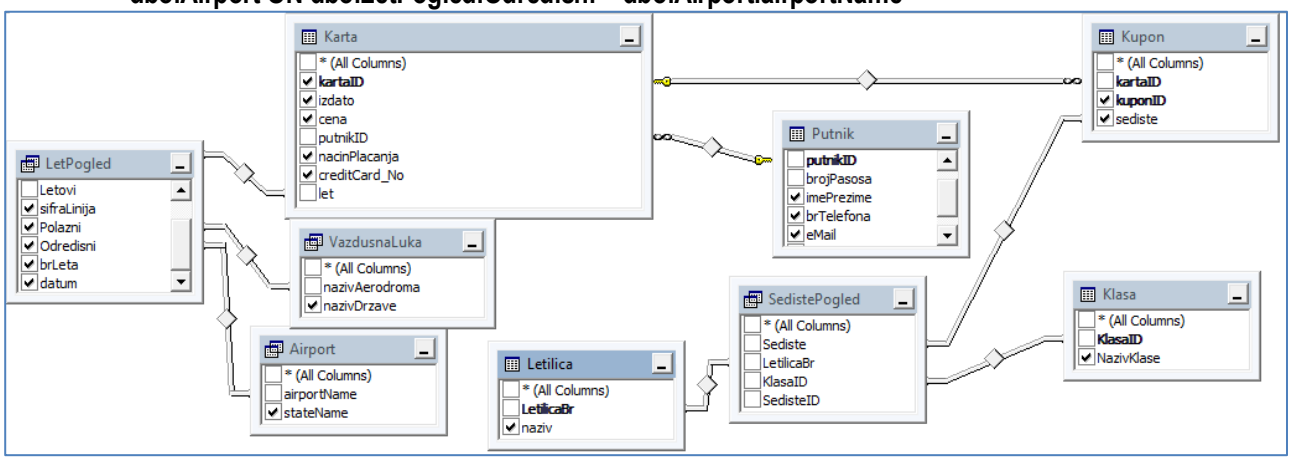
7.4. ФОРМА ПРИПРЕМЕ ЗА ШТАМПУ (Let.CS)

```
namespace avioKarte{
   public partial class Let : Form{
      SqlConnection konekcija = new SqlConnection("Data Source=BNKATANIC\\MSSQLSERVER2008;
      Initial Catalog=avioKarte;Integrated Security=True");// u istom redu sa konekcija-connectcionstring
      SqlCommand komanda = null;
      SqlDataReader cita = null;
      public Let(){
         InitializeComponent();}

      private void Let_Load(object sender, EventArgs e){
         avioKarteDataSet.EnforceConstraints = false;
         this.kartaTableAdapter.Fill(this.avioKarteDataSet.Karta);}

      private void PrintTB_Click(object sender, EventArgs e){
            StampajSFD.Filter = "Plain text|*.txt|All|*.*";  //zahtevanje čuvanja u *.txt fajl
            konekcija.Open();
            if (StampajSFD.ShowDialog() == DialogResult.OK){
               StreamWriter write = new StreamWriter(StampajSFD.OpenFile());
               komanda = new SqlCommand("select * from Stampaj where kartaID like '" + CMBprint.SelectedValue
+ "'", konekcija); //u istom redu sa komanda
               cita = komanda.ExecuteReader();
               while (cita.Read()){
                  try{          //ispisuje sadržaj pogleda štampa u txt fajl
                        write.WriteLine("\tTribalia Airways");
                        write.WriteLine("_____");
                        write.WriteLine("Broj karte:\t" + cita.GetValue(0));
                        DateTime datum = cita.GetDateTime(1);
                        string date = datum.ToShortDateString();
                        write.WriteLine("Datum izdavanja:\t" + date);
```

```
                write.WriteLine("Ime i prezime:\t" + cita.GetValue(2));
                write.WriteLine("Broj telefona:\t" + cita.GetValue(17));
                write.WriteLine("e-Mail Adresa:\t" + cita.GetValue(18));
                write.WriteLine("Cena(u evrima):\t" + cita.GetValue(3));
                write.WriteLine("Način plaćanja:\t" + cita.GetValue(4));
                write.WriteLine("Broj kartice:\t" + cita.GetValue(5));
                write.WriteLine("Šifra linije:\t" + cita.GetValue(6));
                write.WriteLine("Polazni aerodrom:\t" + cita.GetValue(7) + cita.GetValue(16));
                write.WriteLine("Odredišni:\t" + cita.GetValue(8)+cita.GetValue(15));
                write.WriteLine("Redni broj leta:\t" + cita.GetValue(9));
                DateTime datum2 = cita.GetDateTime(10); //deo koda razdvaja datum/vreme
                string date2 = datum2.ToShortDateString();
                string time = datum2.ToShortTimeString();
                write.WriteLine("Datum polaska:\t" + date2);
                write.WriteLine("Vreme polaska:\t" + time);
                write.WriteLine("Broj kupona:\t" + cita.GetValue(11));
                write.WriteLine("Tip Letilice:\t" + cita.GetValue(12));
                write.WriteLine("Klasa sedišta:\t" + cita.GetValue(13));
                write.WriteLine("Oznaka sedišta:\t" + cita.GetValue(14));
                write.WriteLine("_____");}
            catch { }}
           write.Dispose();
           write.Close();}
        cita.Close();
        konekcija.Close();
        Let_Load(sender, e);}}}
```

КОД SQL ПОГЛЕДА ШТАМПАЈ:

```
SELECT    dbo.Karta.kartaID, dbo.Karta.izdato, dbo.Putnik.imePrezime, dbo.Karta.cena,
          dbo.Karta.nacinPlacanja, dbo.Karta.creditCard_No, dbo.LetPogled.sifraLinija,
          dbo.LetPogled.Polazni, dbo.LetPogled.Odredisni, dbo.LetPogled.brLeta,
          dbo.LetPogled.datum, dbo.Kupon.kuponID, dbo.Letilica.naziv, dbo.Klasa.NazivKlase,
          dbo.Kupon.sediste, dbo.Airport.stateName, dbo.VazdusnaLuka.nazivDrzave,
          dbo.Putnik.brTelefona, dbo.Putnik.eMail
FROM      dbo.Karta INNER JOIN
          dbo.Putnik ON dbo.Karta.putnikID = dbo.Putnik.putnikID INNER JOIN
          dbo.LetPogled ON dbo.Karta.let = dbo.LetPogled.brLetID INNER JOIN
          dbo.Kupon ON dbo.Karta.kartaID = dbo.Kupon.kartaID INNER JOIN
          dbo.SedistePogled ON dbo.Kupon.sediste = dbo.SedistePogled.Sediste INNER JOIN
          dbo.Letilica ON dbo.SedistePogled.LetilicaBr = dbo.Letilica.LetilicaBr INNER JOIN
          dbo.Klasa ON dbo.SedistePogled.KlasaID = dbo.Klasa.KlasaID INNER JOIN
          dbo.VazdusnaLuka ON dbo.LetPogled.Polazni = dbo.VazdusnaLuka.nazivAerodroma INNER JOIN
          dbo.Airport ON dbo.LetPogled.Odredisni = dbo.Airport.airportName
```



Слика 35.*Шема SQL погледа штампај потребна за чување авио карте*

7.5. ФОРМА ЛИНИЈА СА ПОДФОРМОМ БРОЈ ЛЕТА (Linija.CS)

```
namespace avioKarte{
  public partial class Linija : Form{
    public Linija(){
      InitializeComponent();}

    private void Linija_Load(object sender, EventArgs e){
      avioKarteDataSet.EnforceConstraints = false;
      this.cMB_AerodromTableAdapter.Fill(this.avioKarteDataSet.CMB_Aerodrom);
      this.aerodromTableAdapter.Fill(this.avioKarteDataSet.Aerodrom);
      this.letTableAdapter.Fill(this.avioKarteDataSet.Let);
      this.linijaTableAdapter.Fill(this.avioKarteDataSet.Linija);}

    private void linijaBindingNavigatorSaveItem_Click(object sender, EventArgs e){
      this.Validate();
      this.linijaBindingSource.EndEdit();
      this.tableAdapterManager.UpdateAll(this.avioKarteDataSet);}

    private void LinijaDGV_SelectionChanged(object sender, EventArgs e){
      if (LinijaDGV.SelectedRows.Count > 0){
        foreach (DataGridViewRow row in LinijaDGV.SelectedRows){
          brLinTB.Text = Convert.ToString(LinijaDGV.SelectedRows[0].Cells[0].Value);
          PolazniCMB.Text = Convert.ToString(LinijaDGV.SelectedRows[0].Cells[1].Value);
          OdredisniCMB.Text = Convert.ToString(LinijaDGV.SelectedRows[0].Cells[2].Value);}}}

    private void urediBTN_Click(object sender, EventArgs e){
      LinijaDGV.BeginEdit(true);
      LinijaDGV.SelectedRows[0].Cells[1].Value = PolazniCMB.SelectedValue;
      LinijaDGV.SelectedRows[0].Cells[2].Value = OdredisniCMB.SelectedValue;
      LinijaDGV.EndEdit();
      this.Validate();
      linijaTableAdapter.Update(avioKarteDataSet.Linija);}

    private void ukloniBTN_Click(object sender, EventArgs e){
      foreach (DataGridViewRow item in this.LinijaDGV.SelectedRows) {
        LinijaDGV.Rows.RemoveAt(item.Index);
        linijaTableAdapter.Update(avioKarteDataSet.Linija);}}

    private void letDataGridView_SelectionChanged(object sender, EventArgs e){
      if (letDataGridView.SelectedRows.Count > 0){
        foreach (DataGridViewRow row in letDataGridView.SelectedRows){
          brLetaTB.Text = Convert.ToString(letDataGridView.SelectedRows[0].Cells[1].Value);
          datumDCB.Text = Convert.ToString(letDataGridView.SelectedRows[0].Cells[2].Value);}}}

    private void brLetRemove_Click(object sender, EventArgs e){
      foreach (DataGridViewRow item in this.letDataGridView.SelectedRows){
        letDataGridView.Rows.RemoveAt(item.Index);
        this.letTableAdapter.Update(this.avioKarteDataSet.Let);}}

    private void brLetEdit_Click(object sender, EventArgs e){
      try{
        letDataGridView.Refresh();
        letDataGridView.BeginEdit(true);
        letDataGridView.SelectedRows[0].Cells[2].Value = datumDCB.Value;
        this.Validate();
        letDataGridView.EndEdit(DataGridViewDataErrorContexts.Commit);
        this.tableAdapterManager.UpdateAll(this.avioKarteDataSet);
        this.letTableAdapter.Update(this.avioKarteDataSet.Let);}
      catch { } }}}
```

7.6. ФОРМА ПУТНИК (Putnik.CS)

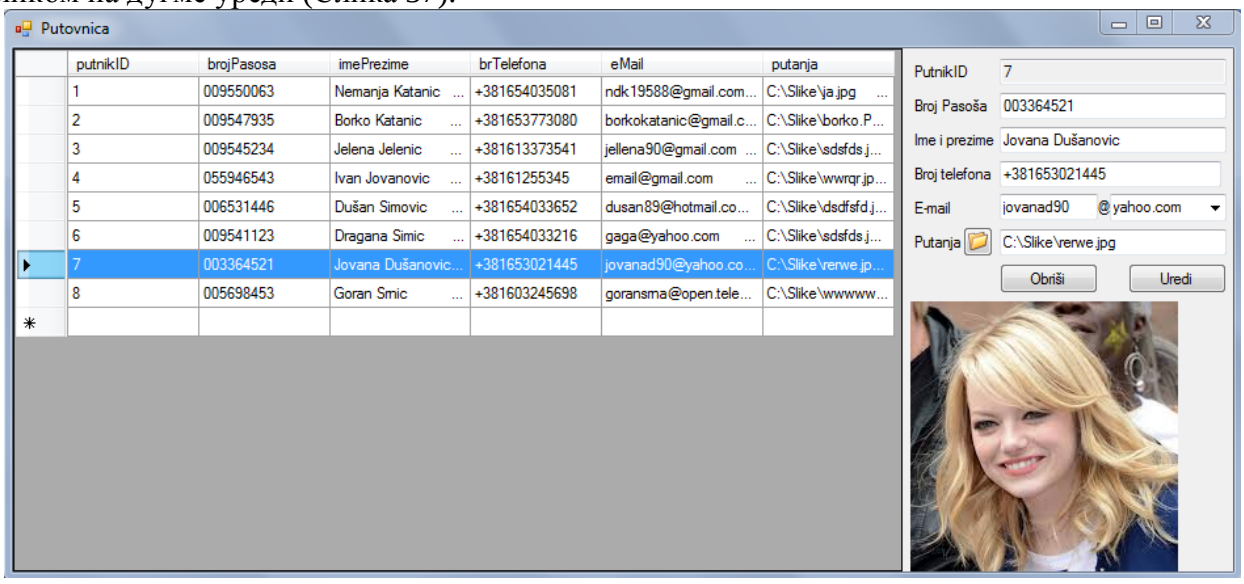
```
namespace avioKarte{
   public partial class Putnik : Form{
      public Putnik(){
         InitializeComponent();}
      private void putnikBindingNavigatorSaveItem_Click(object sender, EventArgs e){
         this.Validate();
         this.putnikBindingSource.EndEdit();
         this.putnikTableAdapter.Update(this.avioKarteDataSet.Putnik);}
      private void Putovnica_Load(object sender, EventArgs e){
         this.avioKarteDataSet.EnforceConstraints = false;
         this.putnikTableAdapter.Fill(this.avioKarteDataSet.Putnik);}
      private void ukloniDGM_Click(object sender, EventArgs e) {
         foreach (DataGridViewRow item in this.putnikDGV.SelectedRows){
            putnikDGV.Rows.RemoveAt(item.Index);
            this.tableAdapterManager.UpdateAll(this.avioKarteDataSet); } }
      private void SlikaGMB_Click(object sender, EventArgs e) {
         SlikaOFD.Filter = "JPG slika|*.jpg|Windows bitmap|*.bmp|Portable net graphics|*.png|All|*.*";
         if (SlikaOFD.ShowDialog() == System.Windows.Forms.DialogResult.OK) {
            PutanjaTB.Text = SlikaOFD.FileName;}} //funkcija otvara datoteke slika i čuva im putanju
      private void putnikDGV_SelectionChanged(object sender, EventArgs e) {
         if (putnikDGV.SelectedRows.Count > 0) {
            foreach (DataGridViewRow row in putnikDGV.SelectedRows) {
               try{
                  string mailstring = Convert.ToString(putnikDGV.SelectedRows[0].Cells[4].Value);
                  mailstring.Trim(new Char[] { ' ' });
                  MailAddress addr = new MailAddress(mailstring);
                  eMailTB.Text = addr.User;
                  eMailCMB.Text = addr.Host;}//funkcija razdvaja e-mail na slovu @
               catch { }
               putnikIDTB.Text = Convert.ToString(putnikDGV.SelectedRows[0].Cells[0].Value);
               brojPasosaTB.Text = Convert.ToString(putnikDGV.SelectedRows[0].Cells[1].Value);
               imePrezimeTB.Text = Convert.ToString(putnikDGV.SelectedRows[0].Cells[2].Value);
               brTelMTB.Text = Convert.ToString(putnikDGV.SelectedRows[0].Cells[3].Value);
               PutanjaTB.Text = Convert.ToString(putnikDGV.SelectedRows[0].Cells[5].Value);
               string slika = PutanjaTB.Text;
               slika.Trim(new Char[]{' '}); //brise razmake u stringu putanje slike
               try{
                  PutnikPB.Image = Image.FromFile(slika);} //pokusava učitati sliku korisnika
               catch {
                  PutnikPB.Image = null;}}}} //briše prethodnu sliku ako nije učitana nova
      private void urediDGM_Click(object sender, EventArgs e){
         try{
            string mail = eMailTB.Text + "@" + eMailCMB.Text;
            putnikDGV.BeginEdit(true);
            putnikDGV.SelectedRows[0].Cells[1].Value = brojPasosaTB.Text;
            putnikDGV.SelectedRows[0].Cells[2].Value = imePrezimeTB.Text;
            putnikDGV.SelectedRows[0].Cells[3].Value = brTelMTB.Text;
            putnikDGV.SelectedRows[0].Cells[4].Value = mail;
            putnikDGV.SelectedRows[0].Cells[5].Value = PutanjaTB.Text;
            putnikDGV.EndEdit();
            this.Validate();
            this.putnikTableAdapter.Update(this.avioKarteDataSet.Putnik); }
         catch { } }}}
```

8. ПРИКАЗ АПЛИКАЦИЈЕ У РАДУ И КРАТКО УПУТСВО

На радној површини се налази пречица ка апликацији авиоКарте, двокликом покрећемо апликацију. Дочекаће нас главни мени са 7 тастера од чега 5 служи за приступање формама, дугме "Штампај" се користи да би се креирала текстуална датотека прикладна за штампање карте, а дугме "О нама" приказује основне информације о компанији. Са десне стране приказани су датум и време системског сата као и лого саме компаније(слика 36).

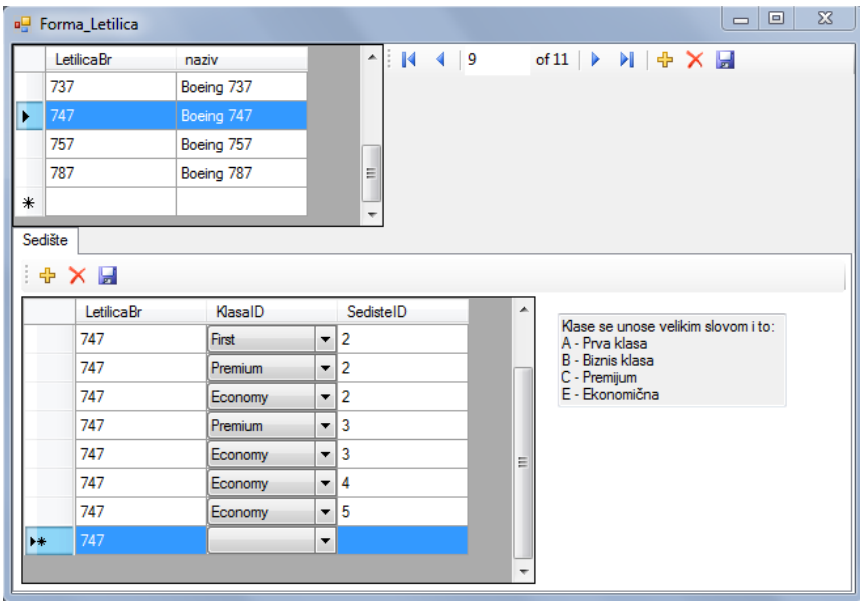
Слика 36. *Главни мени апликације*

Кликом на дугмад Путник, Летилица, Купи карту, Аеродром и Линија приступамо одређеним формама које нам омогућују приступ, унос и измену одређених табела (сличних имена) у SQL серверу. Кренимо са табелом путник кликом на њу отвара се нов прозор, где прво у очи упада велика табела са 6 колона и променљивим бројем врста. Са леве стране табеле уз леву ивицу прозора има мала индикаторска колона, без текста, која обавештава корисника која врста је активна (►) и где се додаје нова врсте (*). У попуњеним врстама кликом на индикаторску колону преносите податке из табеле у техтбохове и падајуће листе са десне стране, а за додавање новог путника, уносите путник_ИД у врсти хоризонталној са звездицом и поново је преносите кликом на њену индикацију. Кликом на иконицу фасцикле можете променити фотографију путника, а кликом на е-маил падајућу листу можете одабрати даваоца интернет услуга за вашу e-mail адресу. Унос и измене се завршавају кликом на дугме уреди (Слика 37).



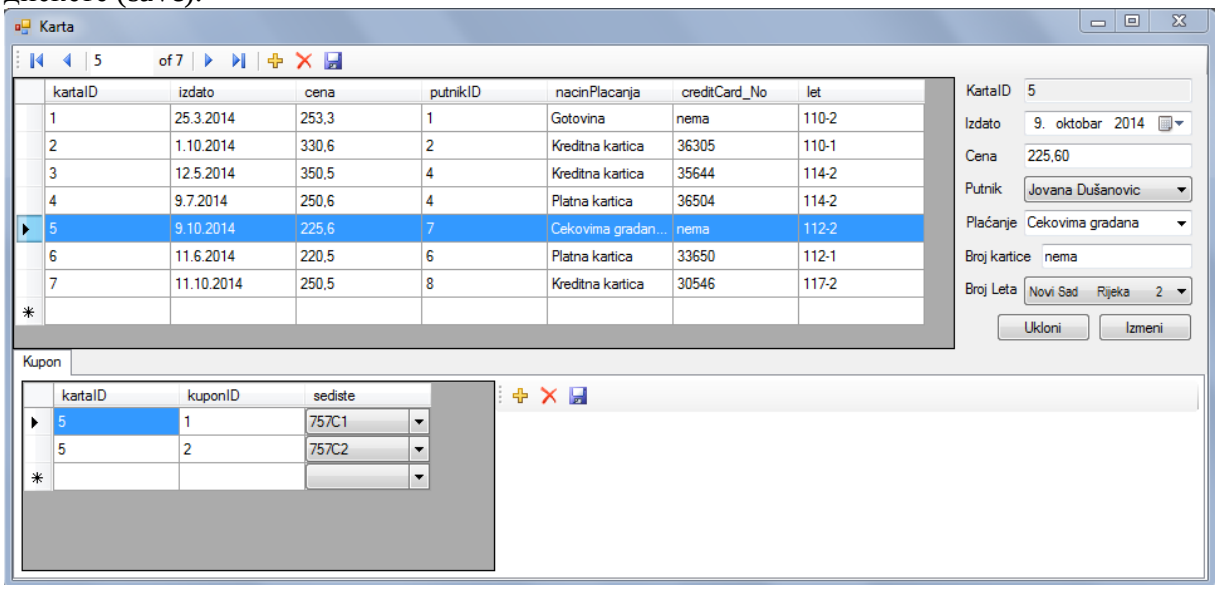
Слика 37.*Форма путник*

Следи форма летилица где можете додати/уклонити ваздухоплов којим се путује, као и убацити комплетну конфигурацију седишта у авиону. Ова форма се састоји од главне форме Летилица и подформе Седиште. Кликом на индикаторску колону одређеног авиона, вредност његовог кључа се преноси на седишта. Самим тим све измене које радите на конфигурацији седишта важе за актуелно изабрани авион.Кад завршите са изменама кликните Save дугме(сл.38).



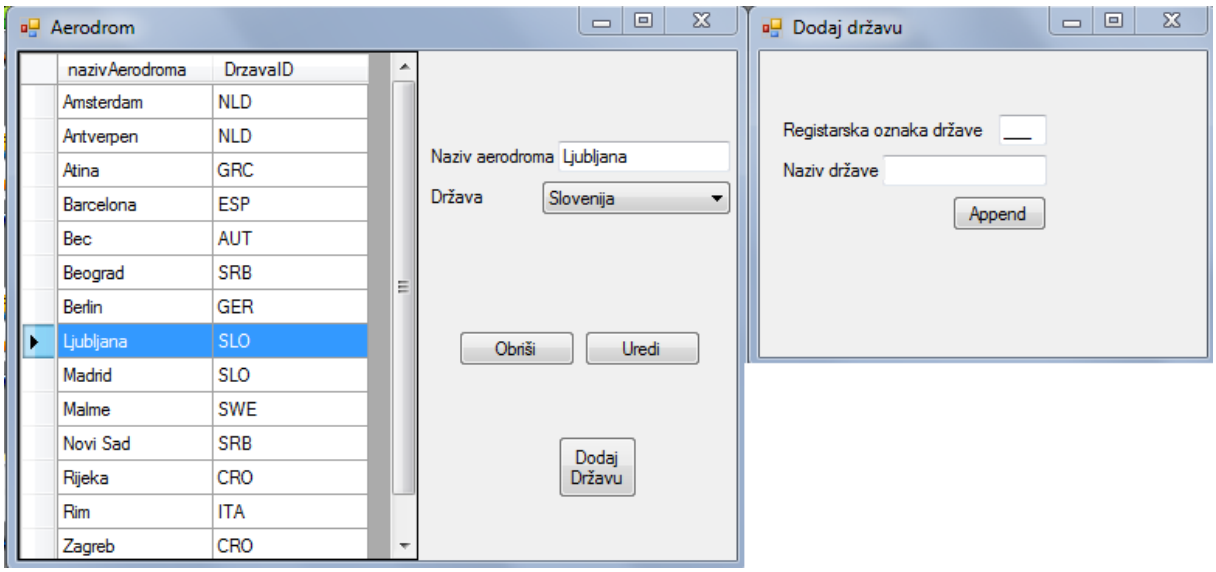
Слика 38.*Форма Летилица и подформа Седиште*

Следи форма за куповину карата, концептуално слична са предходне две форме. Код форме Карта као и код форме Путник textbox за идентификациони кључ је read-only, што значи да су промене дозвољен само директно на Data Grid View (мрежи података). Путник, начин плаћања и датум се бирају помоћу comboboxa. Када продате карте, користите подформу купон за одређивање седишта или укрцавање више путника на једној карти (породична карта). Изглед форме је приказан на слици 39. У случају табеле карта измене се чувају притиском на дугме измени, а у случају подформе то се изводи кликом на иконицу дискете (save).



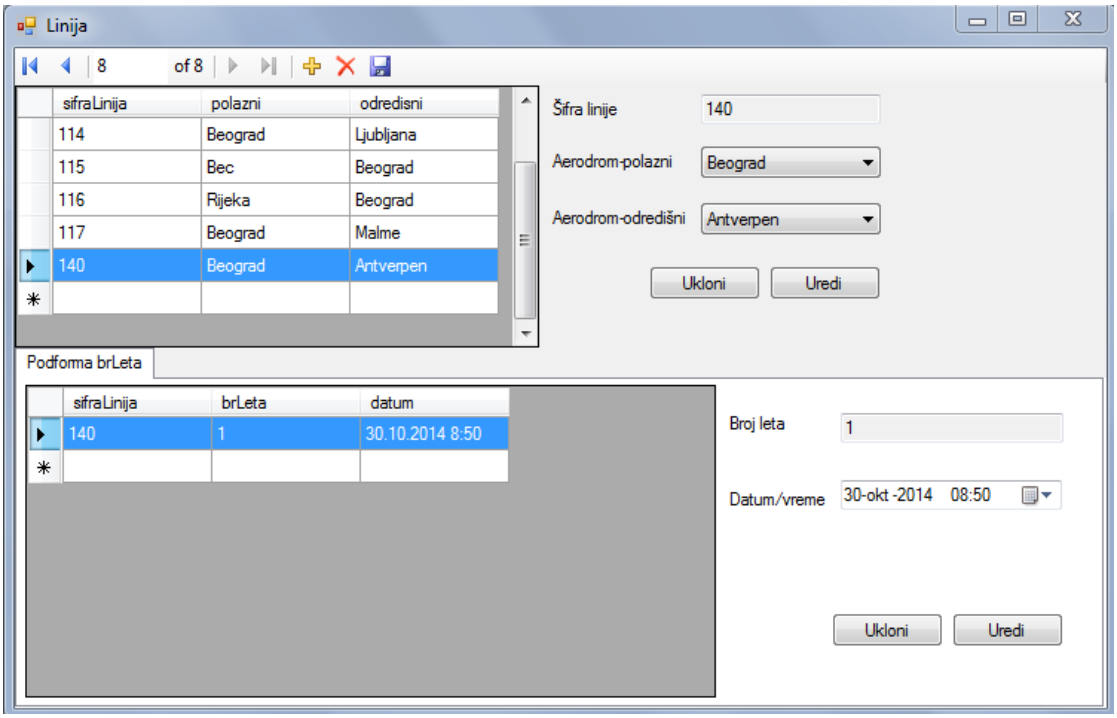
Слика 39.*Форма Карта и подформа Купон*

Форма аеродром служи за додавање/уклањање градова где се налазе аеродроми (полазни-одредишни). Функционише исто као остале форме и има додатак "Додај државу" који отвара нову форму за додавање државе (државе се не могу мењати/брисати). Идентификациони кључ ентитета држава је регистарска ознака од 3 слова и имате простор испод за унос пуног имана државе (слика 40).



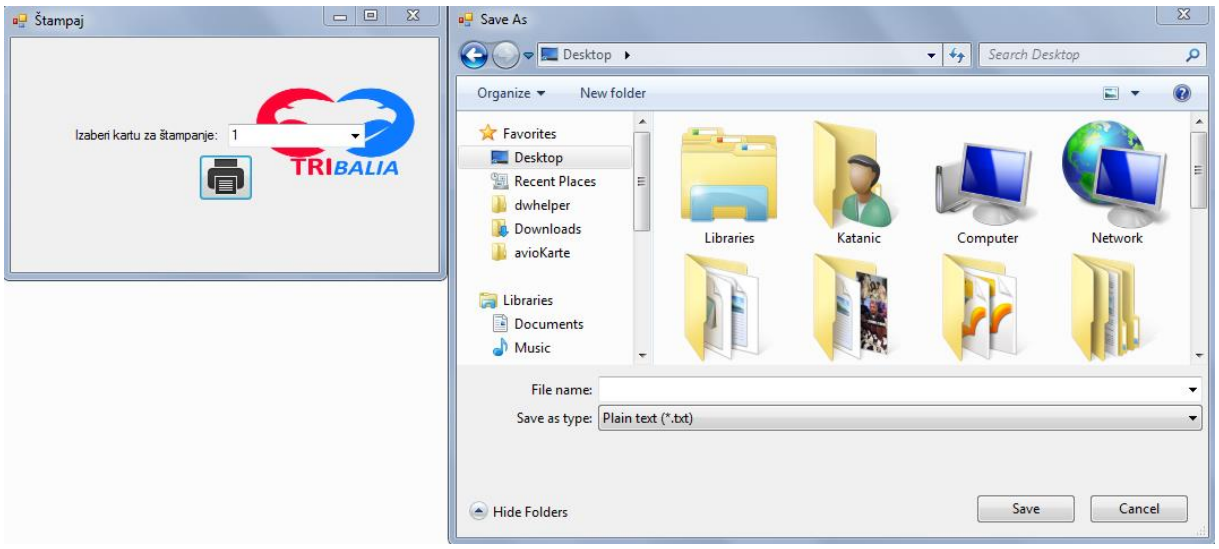
Слика 40.*Форме Аеродром и Додај државу*

Форма Линија има подформу број лета јер једна линија (нпр. Нови Сад-Франкфурт) може имати више летова (1,2,3...). У главној форми се полазни и одредишни аеродром уносе помоћу цомбобохева, а у подформи календарски цомбобох омогућује измену датума и времена, који се при штамању у *.txt фајл раздвајају (слика 41).



Слика 41.*Форма Линија и подформа број Лета*

Када сте купили карту (обавезно је да карта има купон) можете је чувати у форми погодној за штампање. То чините тако што из главног менија позивате форму штампај. Бирате број карте који чувате за штампу путем comboboxa и кликенте на иконицу стампача у центру форме (слика 42).



Слика 41.*Форма Штампај и дијалог прозор за чување карте*

Коначан резултат ће бити карта веома прегледна за штампање (слика 42). Препоручује се да у notepadu буде подешен фонт arial да би вертикално посматрано подаци били у линији. Сам програм има веома брз одзив, али тешко је давати коначни суд будући да му је сама база веома растерећена, и са тек неколико десетина попуњених редова. Програм је крајње једноставан за употребу и ради веома добро.

Tribalia Airways

Broj karte:	**9**	
Datum izdavanja:	**11.10.2014**	
Ime i prezime:	**Bojan Bojović**	
Broj telefona:	**+381603245698**	
e-Mail Adresa:	**bojanbox@yahoo.com**	
Cena(u evrima):	**250,5**	
Način plaćanja:	**Kreditna kartica**	
Broj kartice:	**30546**	
Šifra linije:	**117**	
Polazni aerodrom:	**Beograd**	**Srbija**
Odredišni:	**Sarajevo**	**Bosna i Hercegovina**
Redni broj leta:	**2**	
Datum polaska:	**10.11.2014**	
Vreme polaska:	**21:00**	
Broj kupona:	**1**	
Tip Letilice:	**Boeing 747**	
Klasa sedišta:	**First**	
Oznaka sedišta:	**747A1**	

Слика 42.*Штампана форма авио-карте из програма Notepad*

9. ПРИЛОЗИ

Примери комерцијалних авио-карата са обрасцима за штампање.

10. ЗАКЉУЧАК

У раду је приказан двослојни модел архитектуре, клијент-сервер за који смо се одлучили на основу тренутних потреба и будућег развоја. Док се на серверу налази СУБП (систем за управљање базама података), на клијент страни налази се апликација са одговарајућим корисничким интерфејсом која комуницира са сервером. За сервер страну је изабран Microsoft SQL Сервер 2008 а за клијент страну *Visual Basic*.NET 2010. Након пажљиве анализе корисничких захтева приступило се креирању табела, процедура, а затим и изради корисничке апликације. На примеру евиденције конструкционе документације приказане су могућности сервер стране која функцију проналази у:

- оптималном управљању заједничким ресурсима, што су најчешће подаци,
- управљању базом података којој приступа више корисника,
- контроли приступа и безбедности података,
- централизовано обезбеђење интегритета података за апликацију.

Клијентска апликација врши управљање корисничким интерфејсом и извршава део логичких операција. Она се спаја с оперативним системом ради кориштења мултитаскинга и графичког корисничког интерфејса које он осигурава. Такође се још спаја и с мрежном софтверском компонентом комуникацијског посредника ради приступа сервисима. Основна предност двослојне клијент-сервер система је та што базу података чини независном од апликације, јер интегритет базе података обезбеђује њену независност од апликације која над њом ради, што је једним делом реализовано у дипломском раду. Због тога су овакви системи данас све више присутни у пословним применама.

11. ЛИТЕРАТУРА

[1] Michael Lee (2008) Microsoft SQL Сервер 2008, преведена књига;

[2] Jesse Liberty (2008), Microsoft Visual Studio 2008, prevedena knjiga;

[3] Ненад Филиповић (2006), Објектно оријентисано програмирање, ФТН Чачак.

[4] SQL Сервер за 21 дан, Wutsher A., Компјутер библиотека, 2009, Београд;

[5] Пројектовање база података, Вељовић А., Гојгић Н. ВТШ 2006, Чачак;

[6] Ben Watson: Како до решења C# 4.0, Компијутерска библиотек 2010.

[7] http://stackoverflow.com

[8] http://www.codeproject.com

[9] http://msdn.microsoft.com

[10] http://www.youtube.com/user/ProgramiranjeDJB