Implementacija alata za modeliranje baza podataka

Lehpamer, Mihael

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Lehpamer

IMPLEMENTACIJA ALATA ZA MODELIRANJE BAZA PODATAKA

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mihael Lehpamer

Matični broj: 45137/16–R

Studij: Informacijski sustavi

Implementacija alata za modeliranje baza podataka

ZAVRŠNI RAD

Mentor/Mentorica:

Prof. dr. sc. Rabuzin Kornelije

Varaždin, rujan 2020.

Mihael Lehpamer

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Zadaća ovog završnog rada je odabrati sustav za upravljanje baza podataka te na temelju njega napraviti program koji će tu bazu kasnije modelirati konceptualno tj. napravit će se ERA(eng. *Entity-Relations-Atributes*) model. U ovome radu je na temelju konzultacije sa mentorom napravljena mala izmjena. Umjesto da se uzme već gotova baza podataka u programu je dodana mogućnost da se napravi jednostavna baza podataka sa osnovnim mogućnostima, sve je napravljeno pomoću grafičkog sučelja te se na temelju takve baze podataka napravi već spomenuti ERA model. Opisana aplikacija je napisana pomoću programskog jezika C# u okruženju Microsoft Visual Studio. Microsoft Visual Studio nudi puno mogućnosti za izradu aplikacija od konzolnih aplikacija pa sve do Windows Forma. Pa tako za izradu aplikacije su korištene Windows Forme kako bi se napravilo grafičko sučelje, a za pisanje koda kako bi aplikacija bila što više moguće objektno-orijentirana korištene su klase. Kako ćemo prolaziti kroz aplikaciju i sve njene funkcionalnosti biti će objašnjena svaka klasa i njena svaka metoda te svrhe tih metoda. Biti će objašnjeni postupci rješavanja pojedinih problema te moj način razmišljanja kako sam došao do željenog rezultata. Prilikom izrade nije bila pretjerana potreba za znanjem samih tehnologija, iako je i to poželjno u nekim dozama, nego je više bilo potrebno snalaženje u novim problemima, usudio bi se čak i reći matematičkih problema. Redom će biti opisani korišteni alati, rad same aplikacije, pojedine klase i Windows Forme, prikaz rezultata i na kraju moj sam dojam o izradi rada.

Ključne riječi: C#; Microsoft Visual Studio; Windows Forms; ERA-Dijagram; OOP;

Sadržaj

1. Uvod

Ponuda sustava za upravljanje bazama podataka je velika. Neki od najpoznatijih sustava su: Oracle, MySQL, SolarWinds, IBM DB2, Altibase, Microsoft SQL Server i drugi. Svaki sustav je drugačiji, ali na kraju svaki sustav ima istu svrhu. Prema definiciji bi sustav za upravljanje baza podataka bila aplikacija koja kao što možete pogoditi implementira bazu podataka te upravlja podacima i njihovom obradom. Zadaće takvih sustava malo detaljnije opisane jesu:

- Definicija podataka
- Održavanje podataka
- Sigurnost podataka
- Pretraživanje podataka i
- Kontrola pristupa podacima

U ovome radu nisu implementirane sve funkcionalnosti, nego samo one nužne koje su potrebne za konceptualno modeliranje. Pa tako od tih zadaća imamo implementirano samo definiciju podataka jer je ona nam jedino potrebna za navedeno modeliranje.

Sada možemo krenuti redom što se tiče implementiranih funkcionalnosti. Prilikom otvaranja programa imamo radnu površinu slobodnu za izradu modela, ali najprije se treba napraviti projekt koji će sadržavati sve potrebne podatke. Pod potrebnim podacima mislim na tablice sa njihovim atributima i tipovima podataka. Što se tiče atributa oni se definiraju tako da se odaberu neke od opcija koje će biti kasnije prikazane. Jednom kada se završi definiranje tablica koje predstavljaju skup podataka izabere se opcija koja nam nudi vizualni prikaz onoga što smo sami definirali, tako zvani ERA dijagram.

Prije nego što krenemo na detaljne funkcionalnosti programa navesti ću nešto o okruženju u kojemu sam radio i o tehnologiji koju sam odabrao za izradu aplikacije.

2. Metode i tehnike rada

Ovo poglavlje će se sastojati od tehnologija koje su bile korištene. Nije ih bilo puno korišteno bilo ih je svega jako malo. Biti će objašnjeno dosta generalne neke stvari u ovom poglavlju, a kasnije malo više toga. Ovo poglavlje je samo za uvid u izradu same aplikacije i što je zapravo sve bilo potrebno kako bi se izradila aplikacija.

2.1. Microsoft Visual Studio

Microsoft Visual Studio je okruženje koje nam nudi mogućnost izrade raznovrsnih aplikacija. Aplikacije mogu biti web aplikacije, mobilne aplikacije, desktop aplikacije te igre. Aplikaciju koju sam ja izradio je bila desktop aplikacija. Definicija desktop aplikacije je sljedeća: „Aplikacija koja se samostalno može izvoditi na računalu ili laptop uređaju“ [4]. Neki primjeri takvih aplikacija su: Notepad, WinRar Unzipper, Paint, Alarmi, Sat itd. Programski jezici koje podržava ovo okruženje je poprilično raznovrsno zato što pruža dodatne pakete koji omogućuju implementacije ostalih jezika, a jezici koji dolaze sa „Community“ verzijom su C#, C++, Python, F#, Java, JavaScript, Query Language, Type Script i Visual Basic. Platforme za koje se te aplikacije mogu izraditi su: Android, Azure, iOS, Linux, MacOS, tvOS, Windows i Xbox. Ostalo je još i nabrojiti tipove projekata koje je moguće izraditi, to su redom: Cloud, Console, Desktop, Games, IoT, Machine Learning, Mobile, Office, Service, Test, UWP te Web. Kao što sam naveo ranije mi ćemo se baviti Desktop aplikacijom.

2.2. Desktop aplikacija

U prethodnom poglavlju sam naveo definiciju desktop aplikacije te naveo par primjera, a u ovom poglavlju ću to malo pobliže opisati. Da bi se desktop aplikacija koristila prethodno je treba instalirati na radnu površinu ovisno o vrsti aplikaciji za koju radnu površinu je napisana. Da bi se aplikacija pokrenula ponekada su potrebne određene performanse računala. U usporedbi sa Web aplikacijama koje svoje resurse uzimaju dio sa lokalnog računala, a dio sa servera s kojim komunicira web aplikacije su time limitirane. Dok kod desktop aplikacija što je složenija i zahtjevnima za korisnika ima više smisla kao takva zbog količine koda i kompleknosti, a ne kao web aplikacija. Da bi se izradila jedna takva aplikacija koja će koristiti korisniku u određene svrhe, takva aplikacija treba biti napisana od strane programera koji će kao programska rješenja koristiti klase i funkcije unutar tih klasa koje je sam izradio ili u nekim slučajevima korištenjem već gotovih biblioteka. Takvim načinom pisanja koda na koji se izrada

same aplikacije pojednostavljuje skupa sa količinom napisanog programskog koda zove se Objektno-orijentirano-programiranje(OOP).

2.3. Objektno orijentirano programiranje

Objektno orijentirano programiranje ili OOP je način izrade programa kako bi se smanjila njegova kompleksnost, kod bi bio jednostavniji za čitanje, neki dijelovi koda bi se mogli upotrijebiti više puta te samo pronalaženje grešaka bi bilo lakše. OOP se sastoji od više dijelova koja ga čine kao cjelinu tj. neke njegove vrijednosti su sljedeće:

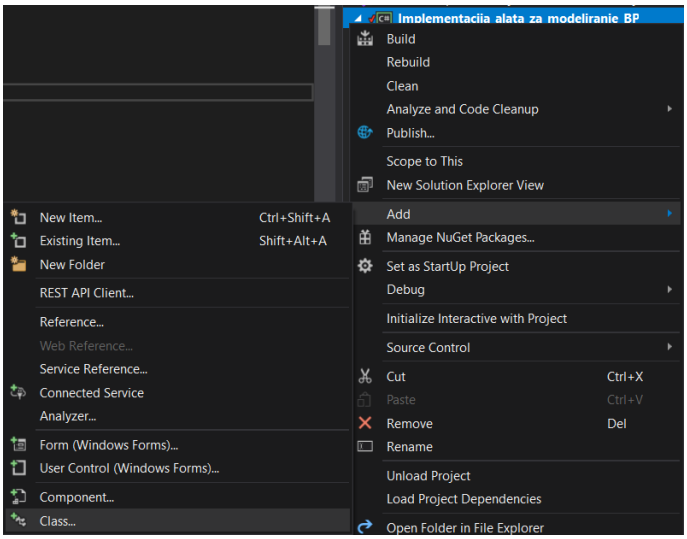
- Objekti - to je osnovna jedinica, a možemo gledati na objekt kao instancu klase. Umjesto stvaranja globalnih varijabli svaki objekt ima svoje varijable i također više nema samostojećih funkcija , nego svaka funkcija pripada određenoj klasi koje nazivamo metode.
- Učahurivanje objekata – kako nemamo globalne varijable iz drugih dijelova koda nije moguć pristup varijablama klase osim preko ugrađenih metoda za njihovo čitanje i pisanje. Tako se osigurava da objekt ne dođe u stanje za koje nije predviđen. [5]
- Apstrakcija – u slučaju da se neki objektni minimalno razlikuju redundantno je njihovo zasebno definiranje te se u tome slučaju dodane posebne klase kao što su apstraktne klase i sučelja [5].
- Nasljeđivanje – jednom kada je objekt definiran, ali nam kasnije zatreba sličan objekt koji je zapravo podskup početnog objektna moguće je naslijediti početni objekt. Time se štedi vrijeme za programiranje [5].
- Polimorfizam – drugi naziv za preopterećivanje metoda. Moguće je definirati par metoda istoga imena, a svaka od njih će primati kao parametre objekte različitih tipova.

Neki od objektno orijentiranih programskih jezika su: C++, Eiffel, Smalltalk, Java, VB .NET, C#, Python. Ali samo jezici nisu dovoljni za izradu takvih aplikacija, nego također moramo imati i određene stavke pa ćemo u nastavku reći nešto malo više o njima.

2.4. Klase i Windows Forme

Microsoft Visual Studio nam nudi mnogo različitih stavka za uporabu. Dvije stavke koje sam ja koristio su bile obične prazne klase koje su kasnije bile definirane prema potrebama aplikacije, a sadrže većinu potrebnog koda te druga također bitna stavka je Windows forma koja mi je dopustila izradu grafičkog sučelja aplikacije.

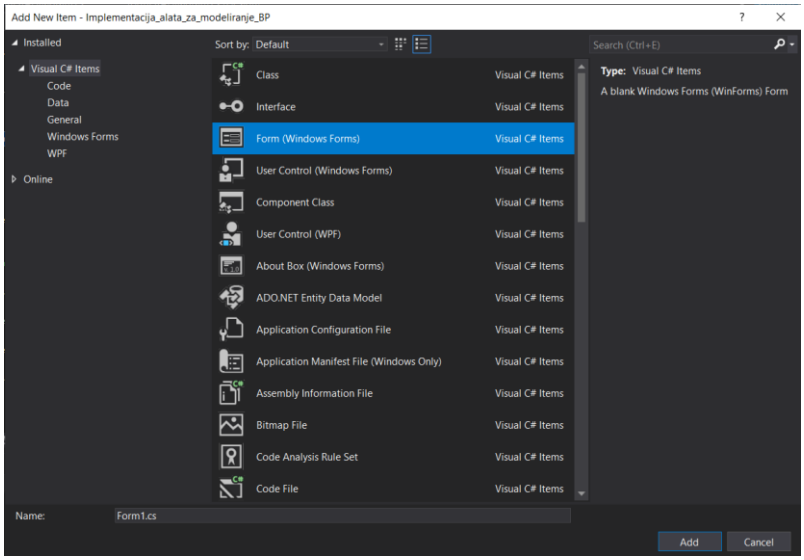
Klasa zapravo predstavlja tip podataka ili skup više podataka koji imaju iste vrijednosti. Najgeneralniji primjer za opisivanje klasa je na temelju „Vozila“, pa tako svako vozilo ima različite komponentne od kojih se sastoji sa svojim različitim osobinama. U programskom svijetu to znači da se može stvoriti klasa naziva „Vozilo“ sa svim potrebnim svojstvima. Jednom kada je sve to definirano možemo napraviti varijablu sa tipom podataka „Vozilo“ te tako napravimo instancu klase koja sadrži ono što nam je potrebno u ovom slučaju to može biti boja auta te pristupimo boji auta preko te instance. U Microsoft Visual Studio stvaranje nove klase je jednostavno, odabere se projekt za koji se želi izraditi klasa te se klikne desni klik pa redom add → class i nazove se prema želji. Imenovanje klasa treba započeti velikim slovom i ne smije sadržavati praznine.



Slika 1: Dodavanje nove klase (Izvor: vlastita izrada)

Windows forma je stavka koja se može definirati kao klasa sa grafičkim sučeljem. Svi vizualni elementi potječu iz posebne klase koja se zove kontrolna klasa što osigurava minimalnu funkcionalnost elemenata korisničkog sučelja poput lokacije, veličine, boje, fonta, teksta te uobičajenih događaja kao što su klikovi mišem ili povlačenje. Pruža poprilično veliku

alatnu kutiju sa mnogo iskoristivih stavki kao što su: gumb, oznaka za tekst, oznake i ostalo. Za dodavanje Windows forma je sličan postupak kao i za dodavanje klase samo se odabere add→new item→Windows form. Pravila imenovanja su ista.



Slika 2: Dodavanje Windows Forme (Izvor: vlastita izrada)

Navedene klase ne znače ništa bez programskog jezika te u nastavku se bavimo ključnom tehnikom, to je poznavanje programskog jezika C#.

2.5. Programski jezik C#

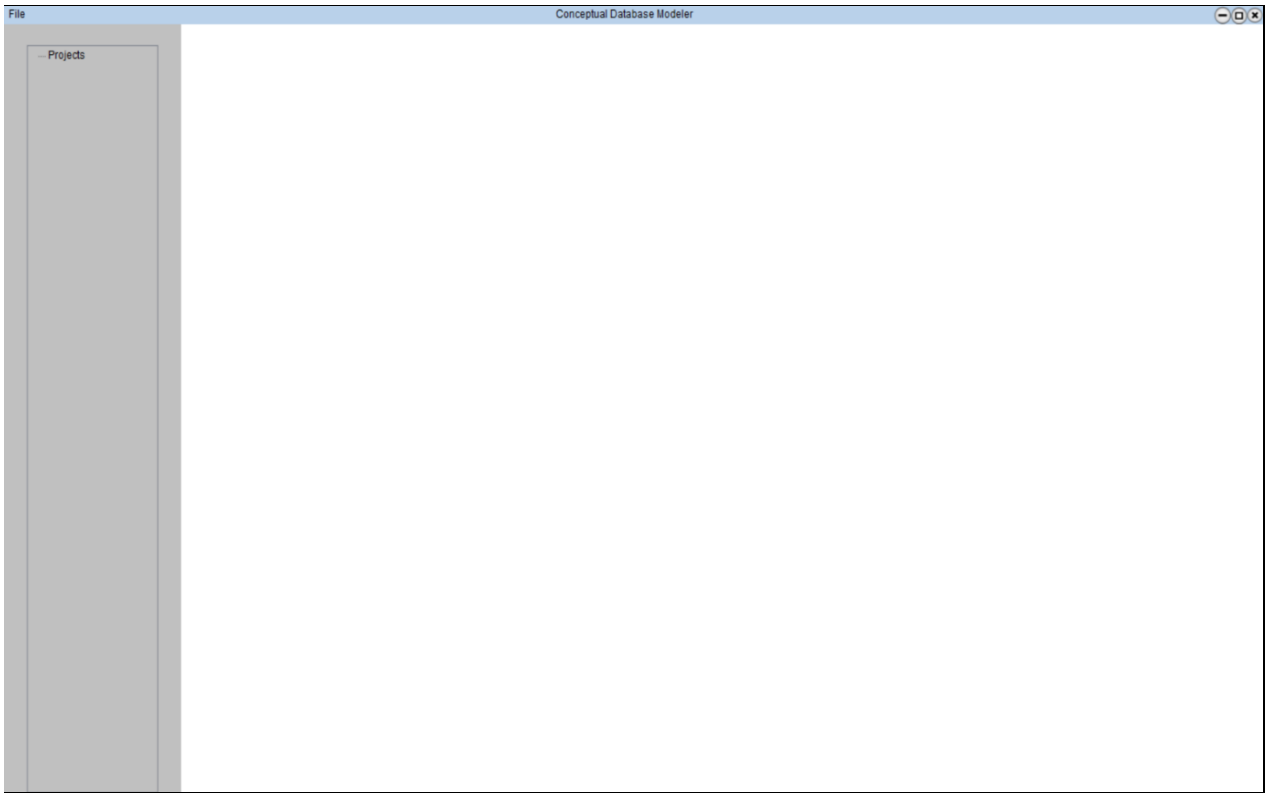
Kao što već znamo C# je jedan od najpopularnijih jezika za izradu aplikacija. Temelji se na objektno orijentiranom principu. Jezik se u početku javio kao kopija Jave, ali je kasnije krenuo u svojemu smjeru, zato se i kaže da su C# i Java dosta slični jezici. Osmišljen je od strane Microsofta, prva verzija se javila oko 2001. godine, a prošle godine 2019. se javila zadnja inačica. Jezik teži da se koristi u razvoju softverskih komponenti adekvatnih za razvoj u okruženju[7]. Jezik, kao i njegove implementacije, trebaju omogućiti podršku za softversko-inženjerske principe kao što su: automatska provjera, provjera granica nizova, provjera pokušaja korištenja nepozvanih varijabli, kao i automatsko upravljanje memorijom. Softverska robusnost, trajnost i programerska produktivnost su važni [7]. Besmisleno je govoriti o samoj sintaksi jezika jer je svaki jezik drugačiji. Meni osobno C# je jedan on najdražih jezika i lako ga je za razumjeti te sam ga stoga izabrao kao jezik u kojemu ću i napisati aplikaciju.

3. Razrada teme

Ova cjelina će se sastojati od svih funkcionalnosti koje su implementirane u aplikaciji. To znači da ćemo vidjeti što je sve korisniku ponuđeno unutar programa. Moram napomenuti prije nego što krenemo na aplikaciju da sve funkcionalnosti koji su dodane su osnovne tj. bile su dodane samo one koje su potrebne kako bi se uspješno izradio konceptualni model baze podataka. Tako da u ovom djelu poglavlja će biti objašnjeno samo grafičko sučelje i koje su mu uloge, a tek u sljedećem poglavlju ćemo vidjeti kako je to implementirano.

3.1. Početni izgled aplikacije

Izgled aplikacije kada se aplikacija jednom učita izgleda sljedeće:



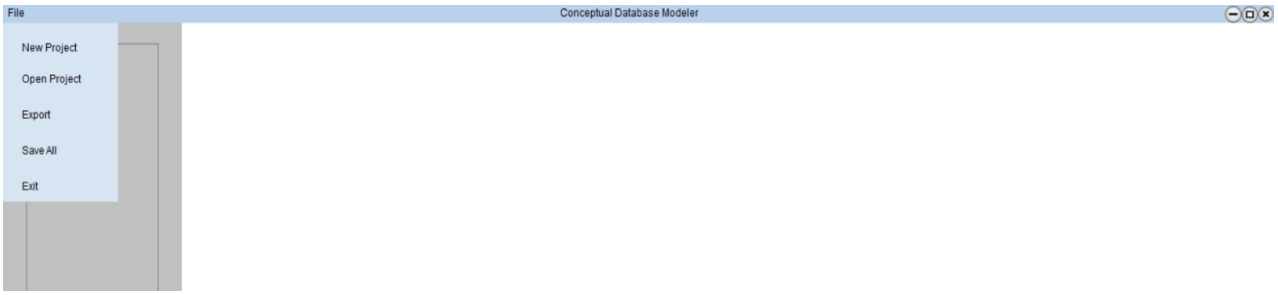
Slika 3: Početni izgled aplikacije (Izvor: vlastita izrada)

Na slici broj tri možemo vidjeti početni izgled aplikacije. Sastoji se od alatne trake na kojoj imamo samo „*File*“ oznaku, ime aplikacije te gumbove za: ugasiti aplikaciju, smanjiti aplikaciju ili spustiti aplikaciju. Dok je aplikacija minimizirana moguće je lijevim klikom kliknuti na alatnu traku i držati tipku te tako povlačiti formu po ekranu u bilo koju stranu. Sa lijeve strane

ispod „*File*“ panela vidimo prostor koji je rezerviran za projekte, a pored njega desno je prostor u kojemu se odvija konceptualno modelirane baze podataka.

3.1.1.File panel

Kao što možemo vidjeti na slici 3 alatna traka sadrži panel imena „*File*“. Na tom panelu se nalaze sljedeće opcije:



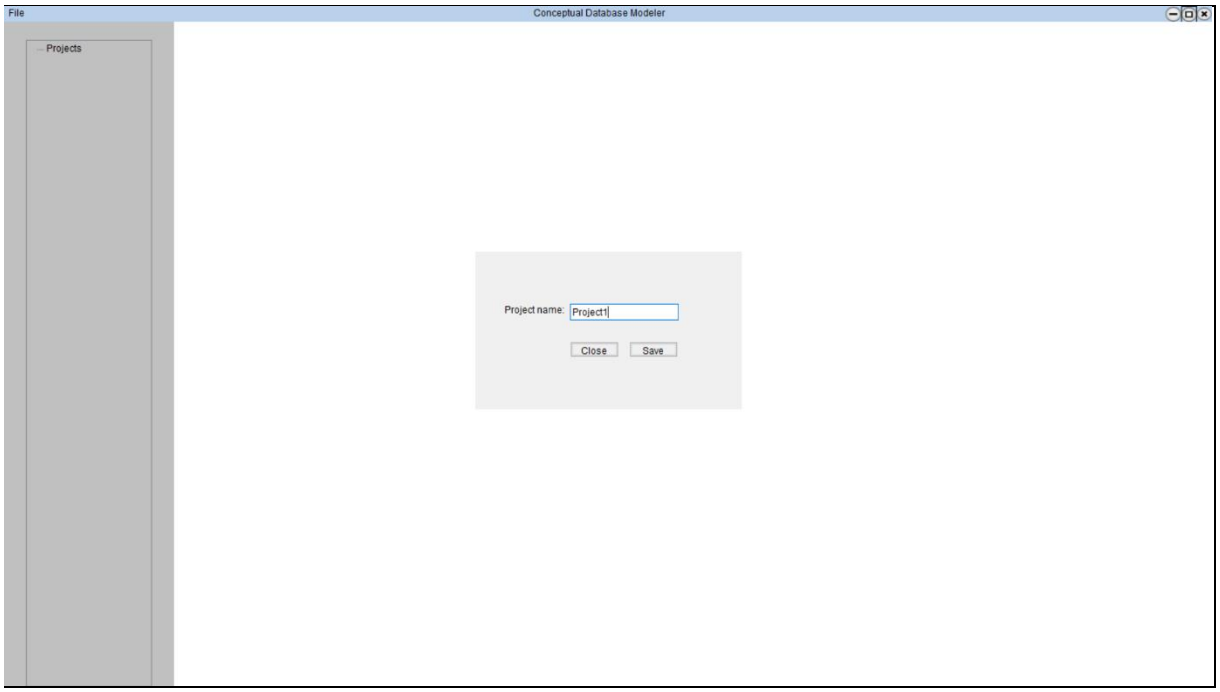
Slika 4: File panel (Izvor: vlastita izrada)

Lijevim ili desnim klikom na „*File*“ se otvara panel koji nam nudi opcije vezane za projekt tj. nudi nam opcije: „*New Project*“, „*Open Project*“, „*Export*“, „*Save All*“ i „*Exit*“.

Opcije koje su ponuđene su osnovne za uporabu aplikacije, sada ću objasniti jednu po jednu opciju.

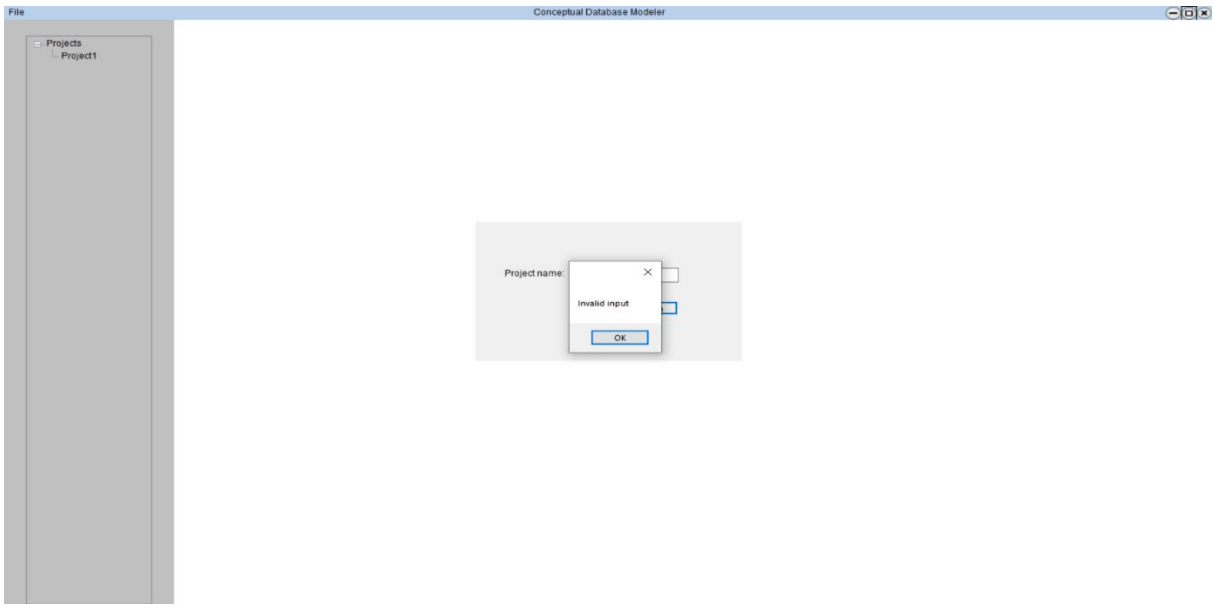
3.1.1.1. New Project

Klikom na opciju „*New Project*“ otvara nam se prozor prikazan na slici 5.



Slika 5: New Project (Izvor: vlastita izrada)

Otvara nam se novi prozor u koji možemo unijeti ime novog projekta kojega želimo izraditi. Kod unosa imena projekta imamo provjere imena kako se ne bi unijelo neispravno ime. Pritiskom na tipku „*Close*“ gasi nam se novootvoreni prozor te promijene nisu spremljene. Klikom na tipku „*Save*“ vrši se provjera imena te ako je ime u skladu sa uputama stvara se novi projekt u mapi lokalnog korisnika računala na putanji *C:User\Documents\Projects\NewProject.*

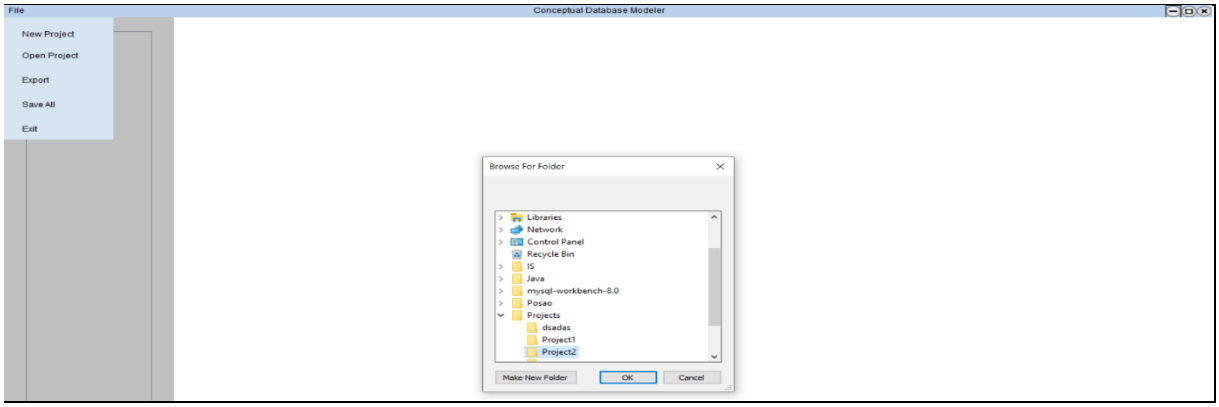


Slika 6: Primjer neispravnog unosa (Izvor: vlastita izrada)

Sljedeći unosi si neispravni: „Prazan „*String*“, početni znak je razmak, te ne smije sadržavati ništa više od velikih i malih slova, brojeva te znaka „_“ “.

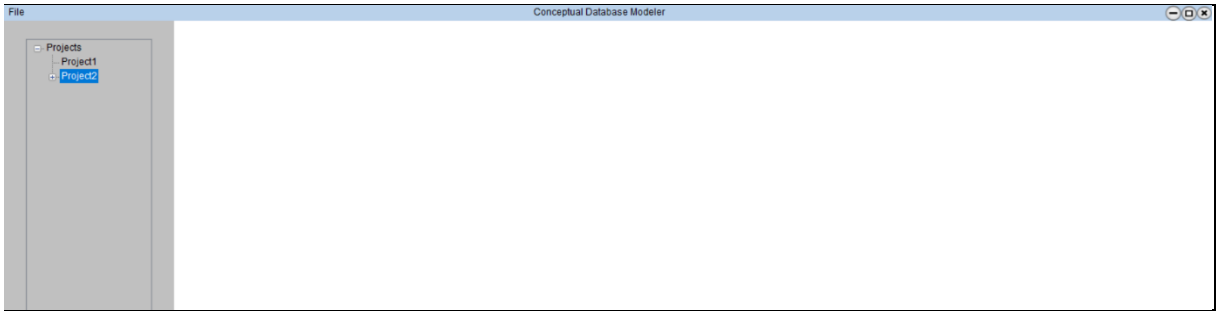
3.1.1.2. Open Project

Klikom na opciju „*Open Project*“ otvara nam se novi prozor od strane operacijskog sustava Windows gdje možemo odabrati projekt koji želimo otvoriti. Te se krećemo prema željenoj mapi.



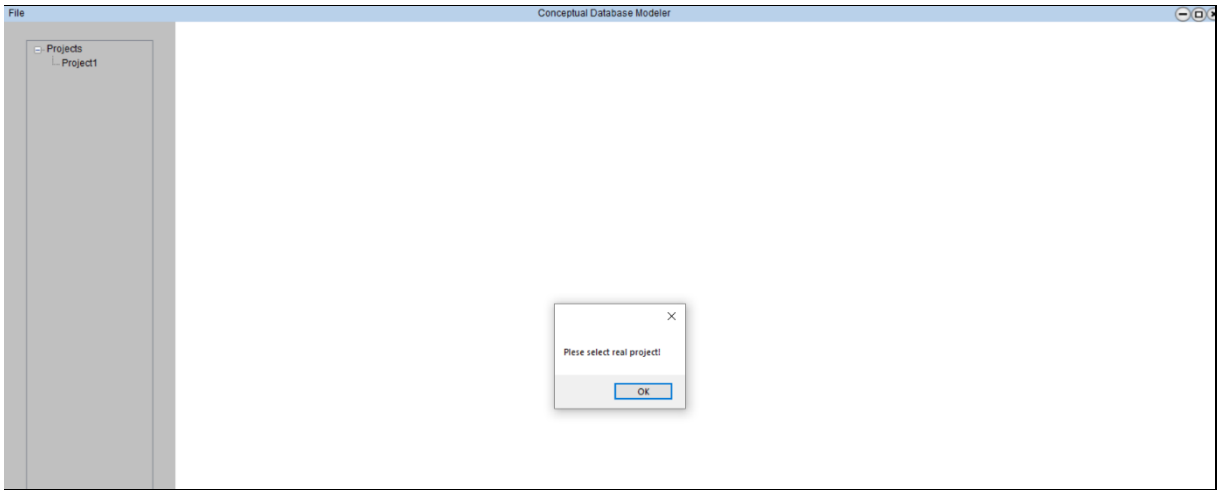
Slika 7: Open Project (Izvor: vlastita izrada)

Jednom kada odaberemo mapu projekta kojega želimo otvoriti potvrdimo unos te se provjerava jeli odabrana mapa zapravo ispravan projekt, ako je sve u redu projekt se dodaje u listu projekata i možemo ga vidjeti sa lijeve strane u prostoru koji je rezerviran za projekte.



Slika 8: Dodan projekt u rezervirani dio za projekte (Izvor: vlastita izrada)

Ako je odabrana mapa u kojoj se ne nalazi projekt javlja se greška te odabrana mapa se ne dodaje u listu projekata.



Slika 9: Odabir pogrešne mape. (Izvor: vlastita izrada)

3.1.1.3. Export

Pritiskom na tipku „*Export*“ stvara se slika konceptualnog modela baze podataka tj. sprema se slika baze podataka ako je prethodno ista generirana. Prvo što nas program pita je lokacija na koju želimo spremiti sliku. Slika se sprema u .*JPG* formatu te se slika spremi na željeno mjesto.

3.1.1.4. Save All

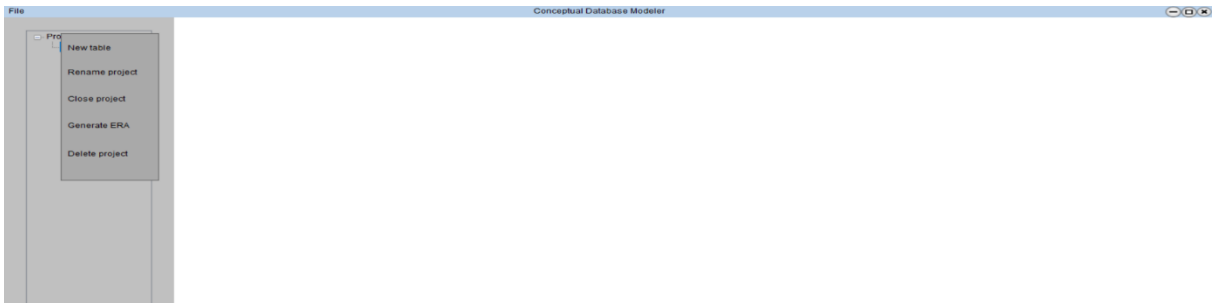
Pritiskom na tipku „*Save All*“ spremaju se svi dosadašnji unosi i izmjene.

3.1.1.5. Exit

Pritiskom na tipku „*Exit*“ gasi se aplikacija.

3.1.2. Opcije projekta

Kao što je već spomenuto sa lijeve strane aplikacije možemo vidjeti sve projekte koje koristimo unutar aplikacije. Svaki projekt ima još postavke koje se odnose samo na njega. Desnim klikom na odabrani projekt otvara nam se novi panel sa novim postavkama.

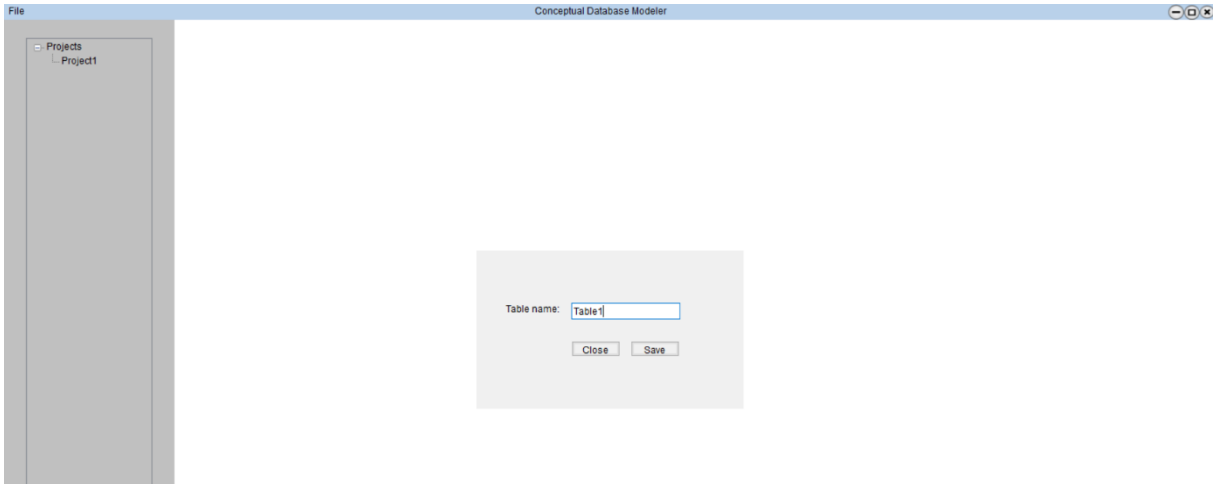


Slika 10: Opcije projekta (Izvor: vlastita izrada)

Na slici 10 možemo vidjeti da kada se otvore opcije projekta da nam se nude sljedeće mogućnosti: “*New table*“, „*Rename project*“, „*Close Project*“, *Generate ERA*“ i „*Delete Project*“.

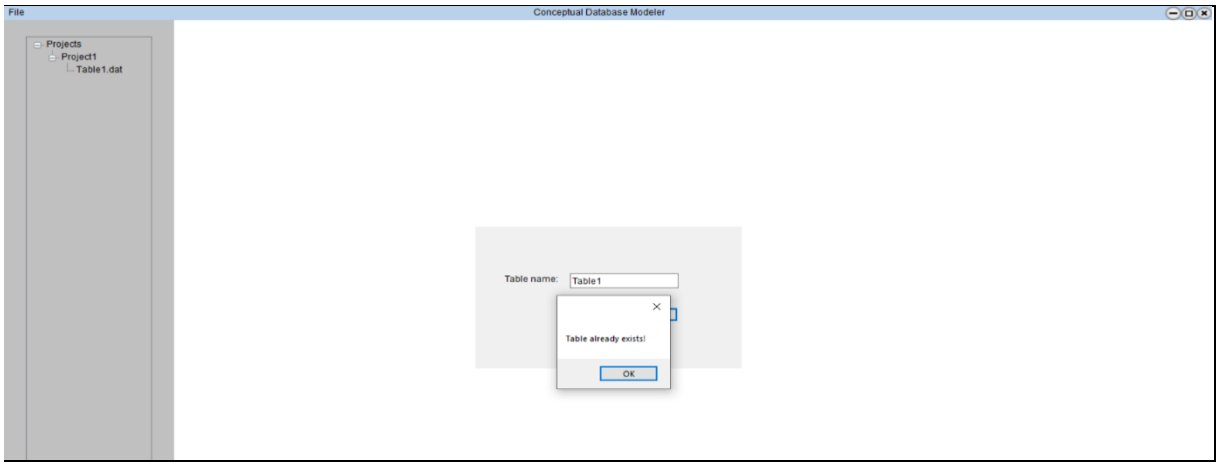
3.1.2.1. New table

Svaki projekt ima mogućnost kreiranja nove tablice pa tako odabirom te opcije prvo nam se otvara novi prozor u kojemu možemo unijeti ime nove tablice.



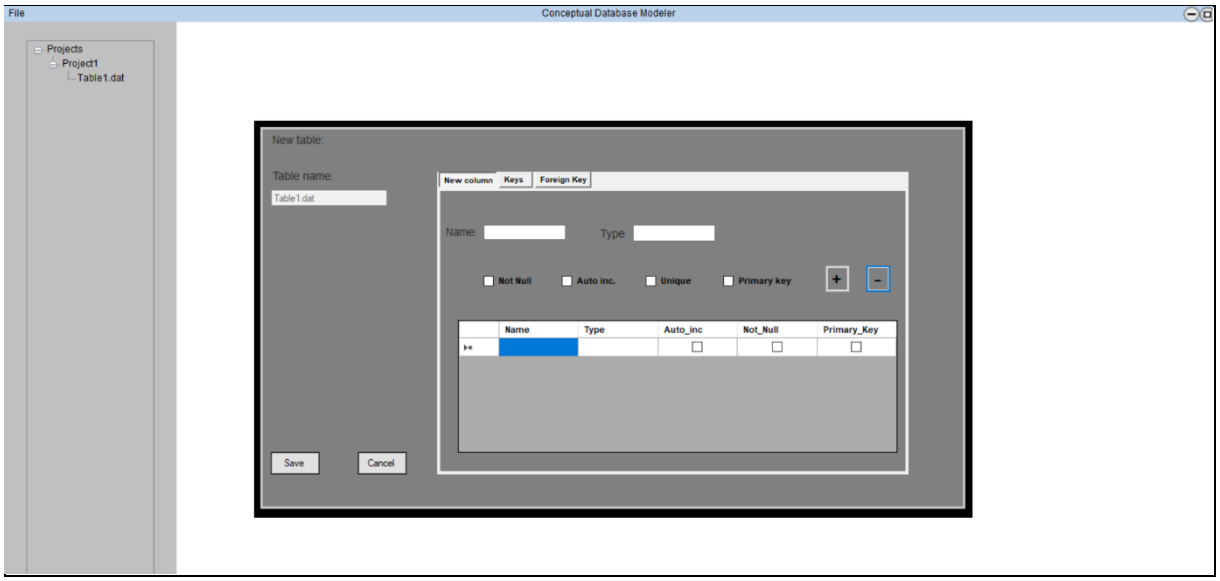
Slika 11: Unos nove tablice (Izvor: vlastita izrada)

Kod unosa nove tablice vrši se prvo provjera da li već postoji tablica sa unesenim imenom te ako ne postoji otvara se novi prozor za unos atributa tablice. Ako je uneseno ime tablice koja već postoji javlja se sljedeća greška:



Slika 12: Unos postojeće tablice (Izvor: vlastita izrada)

Kao što se vidi na slici 12 sa lijeve strane ispod naziva projekta već postoji tablica sa nazivom „*Table1*“ te ponovnim unosom tablice istoga imena javlja nam se greška da takva tablica već postoji te je nije moguće ponovno unijeti. Također se vrše i provjere unosa imena iste kao i kod unosa imena novog projekta pa nema potrebe ponovno pisati istu stvar(javlja se greška kod unosa koji nije podržan). Ako je svaka provjera zadovoljena otvara nam se novi prozor prikazan na slici 13.

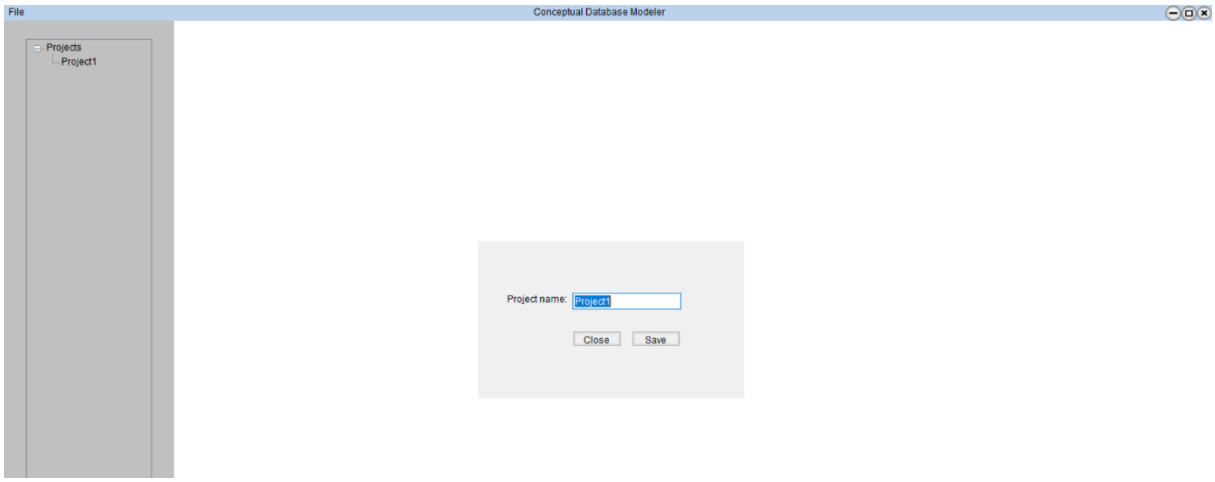


Slika 13: Prozor za unos atributa (Izvor: vlastita izrada)

U ovom poglavlju se nećemo baviti unosom novih atributa, to ćemo ostavit za poslije. Ovo je samo slika kako bismo vidjeli što se događa kada odaberemo opciju „*New Table*“.

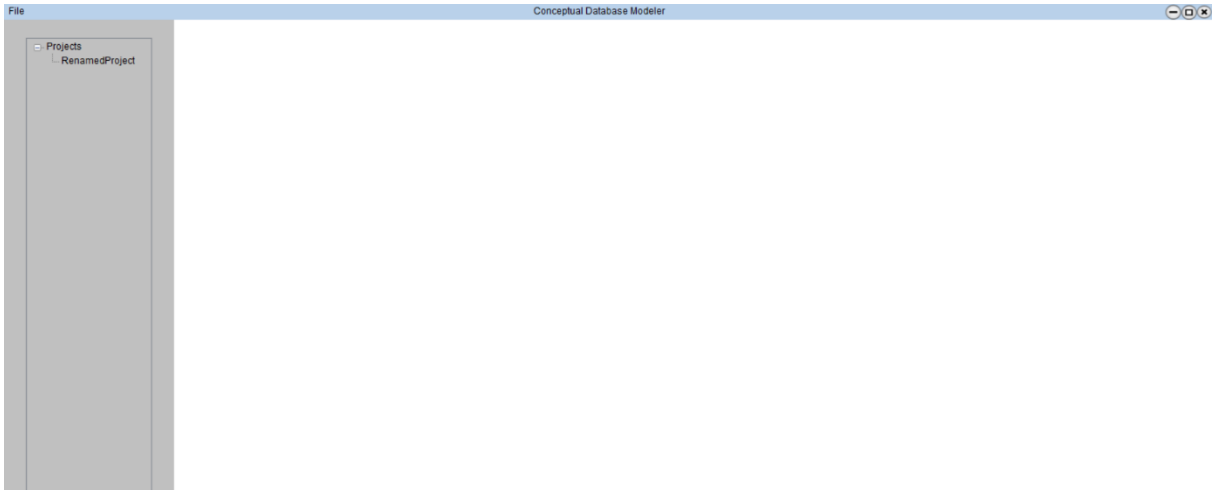
3.1.2.2. Rename Project

Jedna od opcija projekata je preimenovanje odabranog projekta. Vrši se tako da se odabere projekt koji se želi preimenovati te se klikne desni klik na njega i otvara se isti prozor koji je bio otvoren kada se unosio novi projekt samo ovoga puta taj prozor služi za preimenovanje.



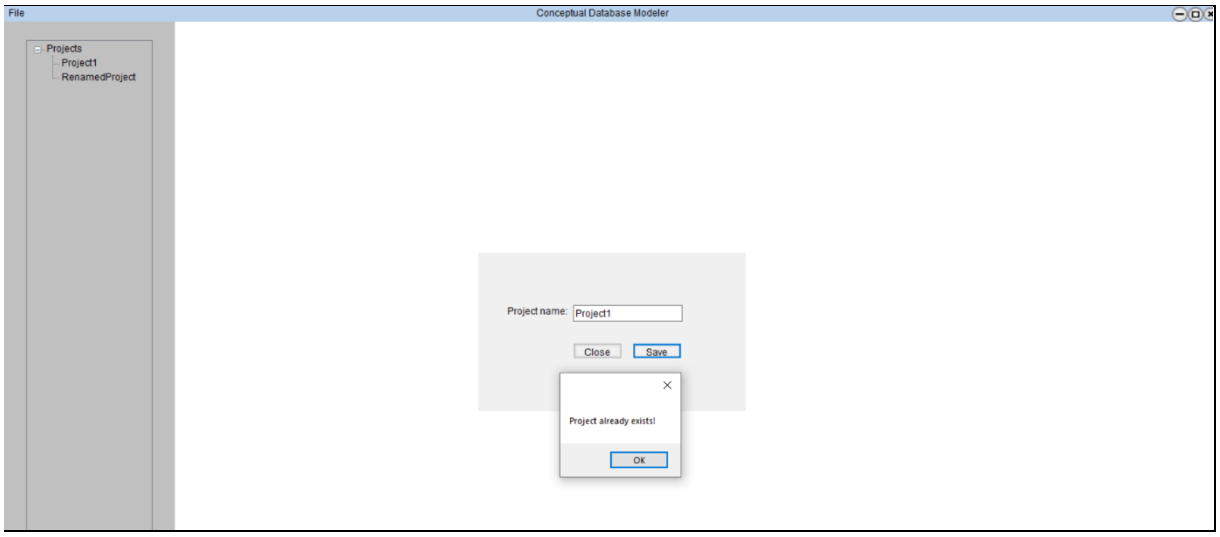
Slika 14: Preimenovanje projekta (Izvor: vlastita izrada)

U okviru gdje se treba upisati ime projekta je inicijalizirano trenutno ime projekta i sve što trebamo napraviti da bismo preimenovali projekt je taj da trebamo upisati novo ime projekta.



Slika 15: Preimenovano ime projekta (Izvor: vlastita izrada)

Ako se unese ime projekta koje već postoji u listi svih projekta javlja se sljedeća greška:



Slika 16: Unos preimenovanja projekta koji već postoji (Izvor: vlastita izrada)

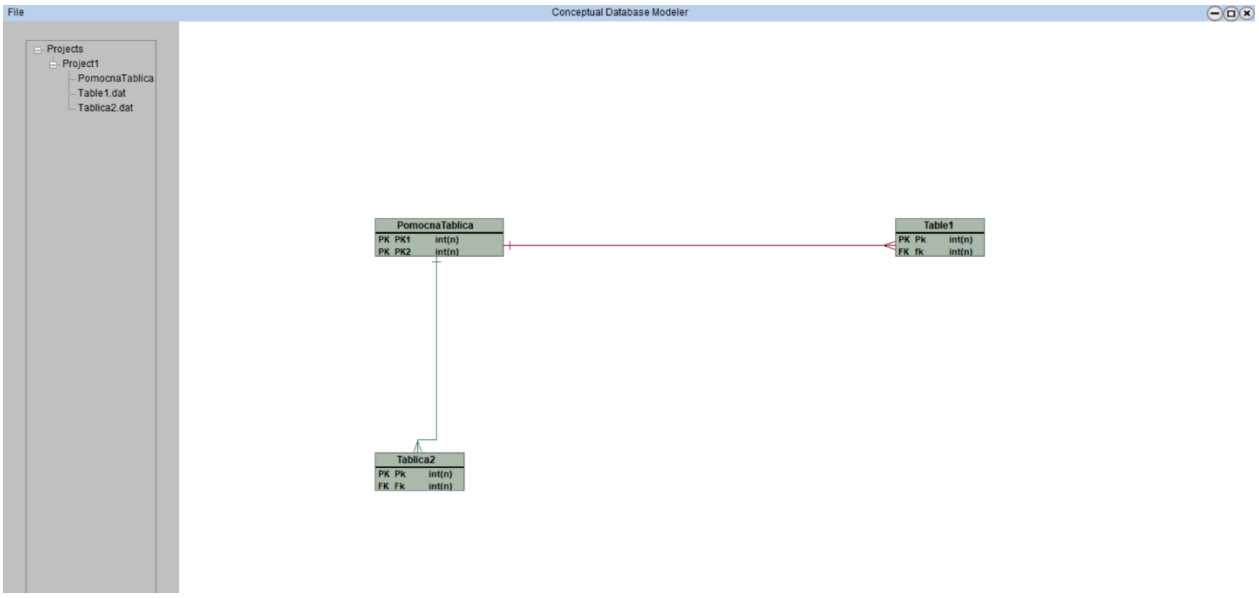
Na slici 16. smo pokušali preimenovati projekt „*RenamedProject*“ u ime „*Project1*“ koji očito već postoji te nam program to nije dopustio.

3.1.2.3. Close Project

Sljedeća opcija za objasniti nam je „*Close Project*“. Prvo moram napomenuti da se svi projekti nalaze u istoj mapi na lokalnom računalu korisnika u mapi *C:User\Documents\Projects* te se prilikom pokretanja programa stvara lista koja sadrži ime svake podmape tog direktorija. Pomoću te liste nam se popunjava prostor sa lijeve strane koji je predviđen za projekte pa tako vidimo svaki projekt u toj mapi jedom kada se aplikacija pokrene. U svrhu ove funkcije „*Close Project*“ kreirana je nova lista koja je zapravo kopija liste koja sadrži ime svakog projekta kako bi se projekti mogli gasiti dok je aplikacija upaljena tj. briše se ime projekta iz te liste pa se popunjava prostor projekata prema toj pomoćnoj listi. Jednom kada se aplikacija ponovno pokrene vidjet će se ponovno svi projekti koji nam se nalaze u gore navedenoj mapi. Ta funkcionalnost je dodana u svrhu lakšeg pregleda projekata, ako dođe do nagomilanosti. Stavio bi vizualni prikaz te funkcionalnosti, ali nema smisla jer samo je razlika ta da više ne vidimo projekt u predviđenom prostoru.

3.1.2.4. Generate ERA

Kada smo gotovi sa modeliranjem baze podataka možemo odabrati opcije „*Generate ERA*“ koja se odabire pritiskom na desni klik odabranog projekta te dobivamo sliku kako bi ta baza podataka izgledala u konceptualnom prikazu. Kada program završi sa generiranjem konceptualnog prikaza on se pojavljuje u predviđenom prostoru sa desne strane aplikacije kao na slici 17.

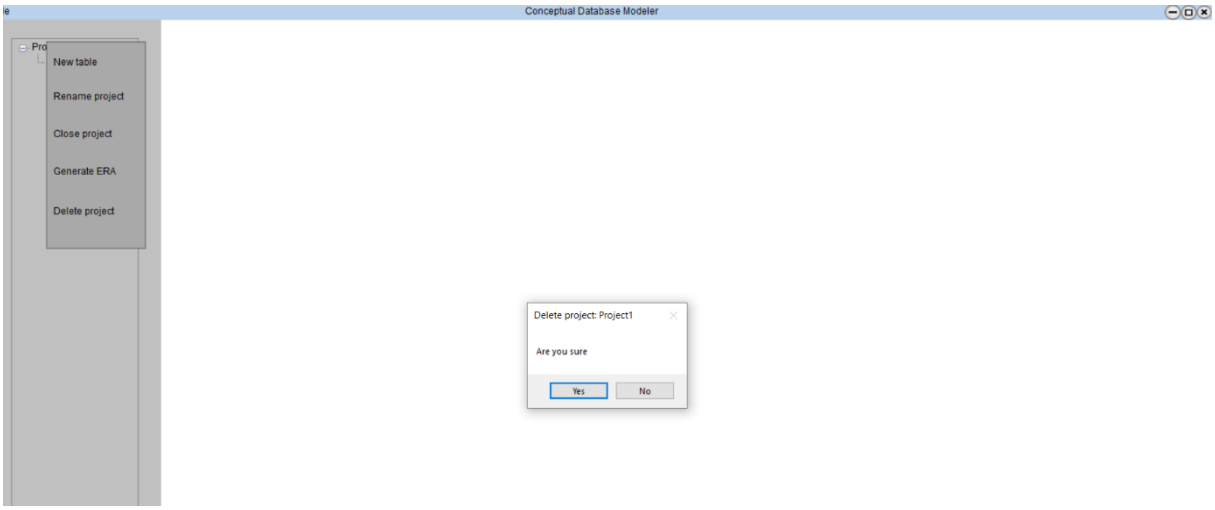


Slika 17: ERA dijagram (Izvor: vlastita izrada)

Razlog zašto su veze tablica u drugim bojama će biti objašnjeno u opisu programskog dijela rada.

3.1.2.5. Delete Project

Za razliku od funkcionalnosti „*Close Project*“ funkcionalnost „*Delete Project*“ ne briše samo iz liste ime projekta nego briše cijelu mapu koja je odabrana te se više taj projekt ne može vratiti, ali prije toga se traži potvrda korisnika da li je siguran da želi dovršiti tu akciju.



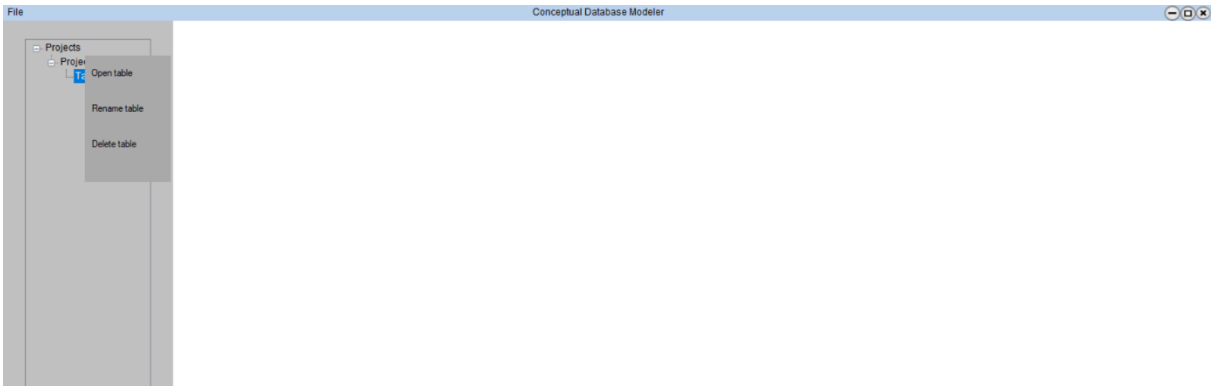
Slika 18: Brisanje projekta (Izvor: vlastita izrada)

Klikom na gumb „*Yes*“ trajno se briše projekt, a na gumb „*No*“ prekida se navedena akcija. Također možemo vidjeti ime projekta koji želimo obrisati.

Ovim poglavljem su završene sve opcije i mogućnosti koje se tiču projekta pa prelazimo na mogućnosti pojedinih tablica.

3.1.3. Opcije tablica

Opcije pojedinih tablica nam se otvaraju kada odaberemo željenu tablicu i desnim klikom miša izvršimo akciju koja nam otvara opcije tablice koje su sljedeće:

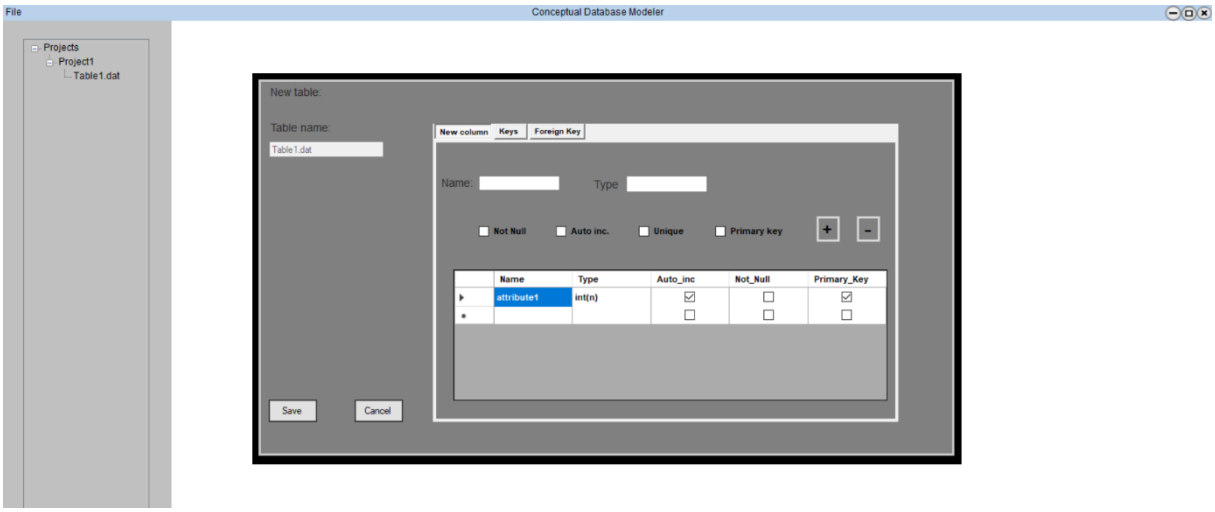


Slika 19: Opcije tablice(Izvor: vlastita izrada)

U svrhu ovoga primjera je dodana tablica koja sadrži samo jedan atribut koja je ujedno i primarni ključ tablice u suprotnom tablica ne bi mogla biti kreirana. Slika 19. nam pokazuje opcije svake tablice kojih ima samo troje : „*Open Table*“, „*Rename Table*“ i „*Delete Table*“.

3.1.3.1. Open Table

Navedeno opcija u naslovu služi kako bismo mogli modificirati željenu tablicu tj. možemo joj dodavati atribute ili ih brisati. Klikom na opciju zapravo nam se otvara prozor za unos nove tablice, ali ovaj put sa svim već unesenim atributima.

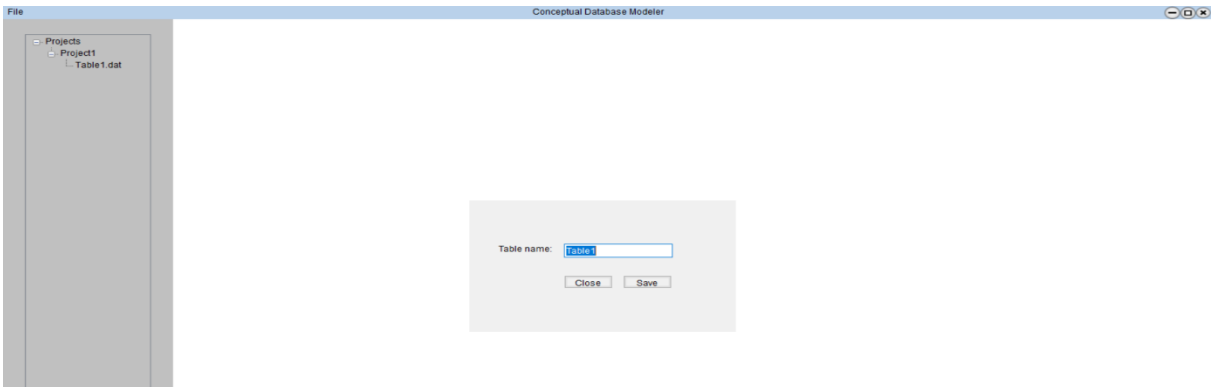


Slika 20: Otvorena postojeća tablica (Izvor: vlastita izrada)

Svi potrebni atributi koje tablica sadrži nalaze se u novootvorenom prozoru. Opet napominjem da ćemo funkcionalnosti toga prozora opisati kasnije u poglavlju.

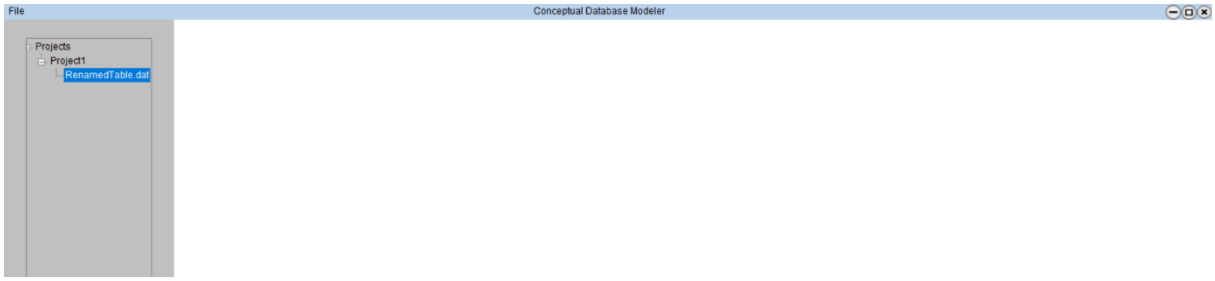
3.1.3.2. Rename Table

Funkcionalnost za preimenovanje tablice je dodana na taj način zato što mi je iskreno trebalo funkcionalnosti koje ću staviti u taj izbornik, ali iz komplikacije mijenjanja koda koji je već bio napisan. Na slici 20 se može vidjeti da je ime tablice zaključano te se ne može promijeniti iz toga prozora. Ova opcija funkcionira na isti način kao i opcija za preimenovanje projekta. Otvara nam se prozor koji je služio za unos nove tablice sa već ispunjenim poljem koji predstavlja trenutno ime tablice te se unosi novo željeno ime tablice koje ne smije biti isto kao ime tablice koje već postoji.



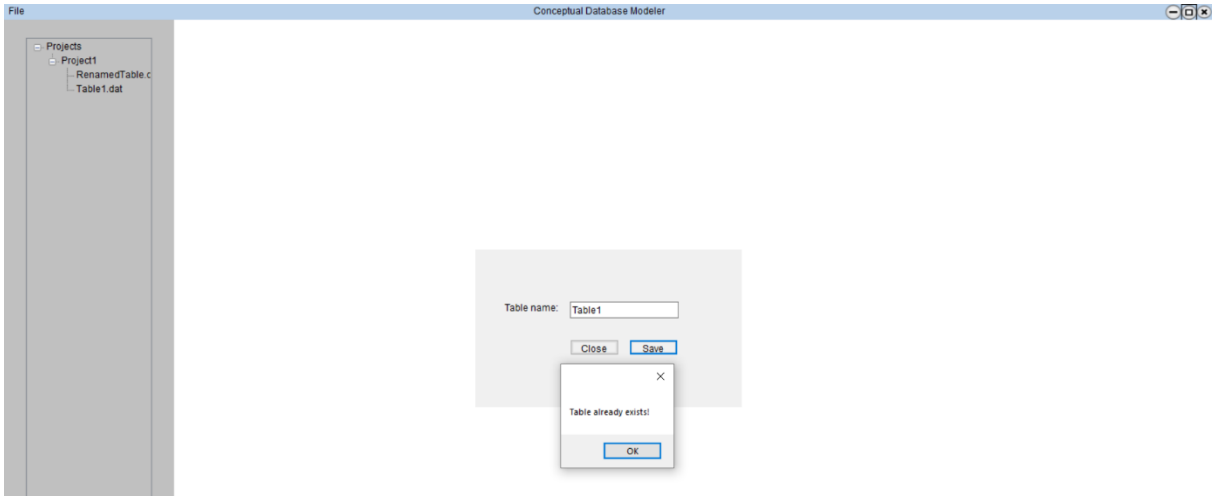
Slika 21: Preimenovanje tablice (Izvor: vlastita izrada)

Klikom na gumb „*Save*“ vrše se navedene provjere i ako je sve u redu tablica se preimenuje u željeno ime.



Slika 22: Preimenovana tablica (Izvor: vlastita izrada)

U slučaju da želimo preimenovati tablicu u ime tablice koja već postoji aplikacija javlja grešku



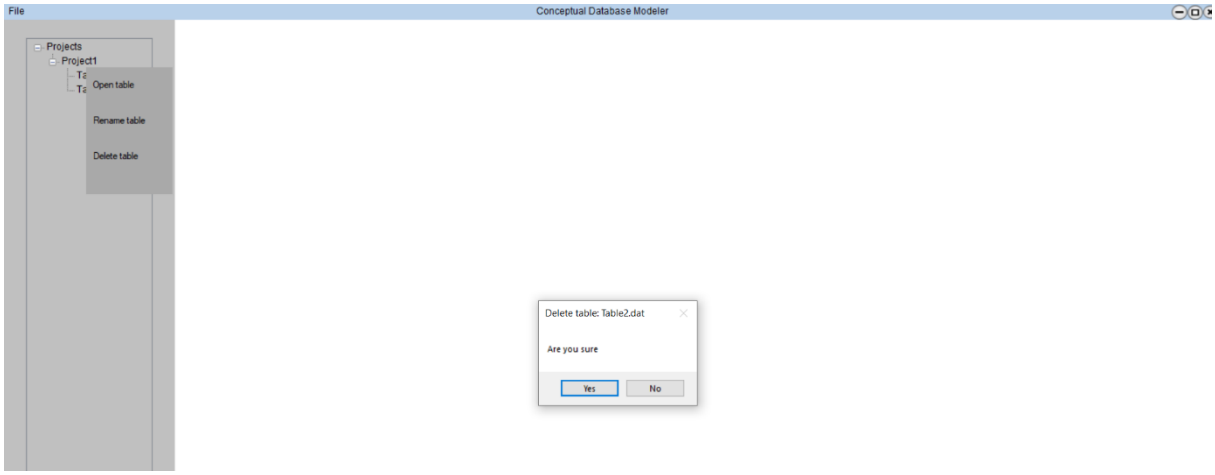
Slika 23: Greška preimenovanja tablice (Izvor: vlastita izrada)

Slika 23 nam pokazuje što se dogodi kada preimenujemo tablicu, u ovom slučaju „*RenamedTable*“, u tablicu sa imenom „*Table1*“. Aplikacija nam to ne dopušta te se ne događaju nikakve promijene.

3.1.3.3. Delete Table

Ovo je jedna od opcija koja je možda malo teža za implementirati jer kada se odabere jedna tablica koja bi se htjela obrisati sa njom bi se trebale i obrisati sve zavisnosti drugih tablica koje imaju veze prema toj tablici. U kodu nije napravljeno da se zabranjuje brisanje

takve tablice koja je ovisna nego je samo napravljeno da se brišu sve veze prema njoj. Brisanje se vrši tako da se odabere željena tablica te prilikom izbornika koji dobijemo desnim klikom obrišemo odaberemo opciju „*Delete Table*“ te isto kao i kod brisanja projekta prvo nam se javlja upozorenje koje traži dodatnu potvrdu za brisanje željene tablice te ako se potvrdi željena radnja tablica je izbrisana iz sustava skupa sa svima zavisnostima.



Slika 24: Brisanje tablice (Izvor: vlastita izrada)

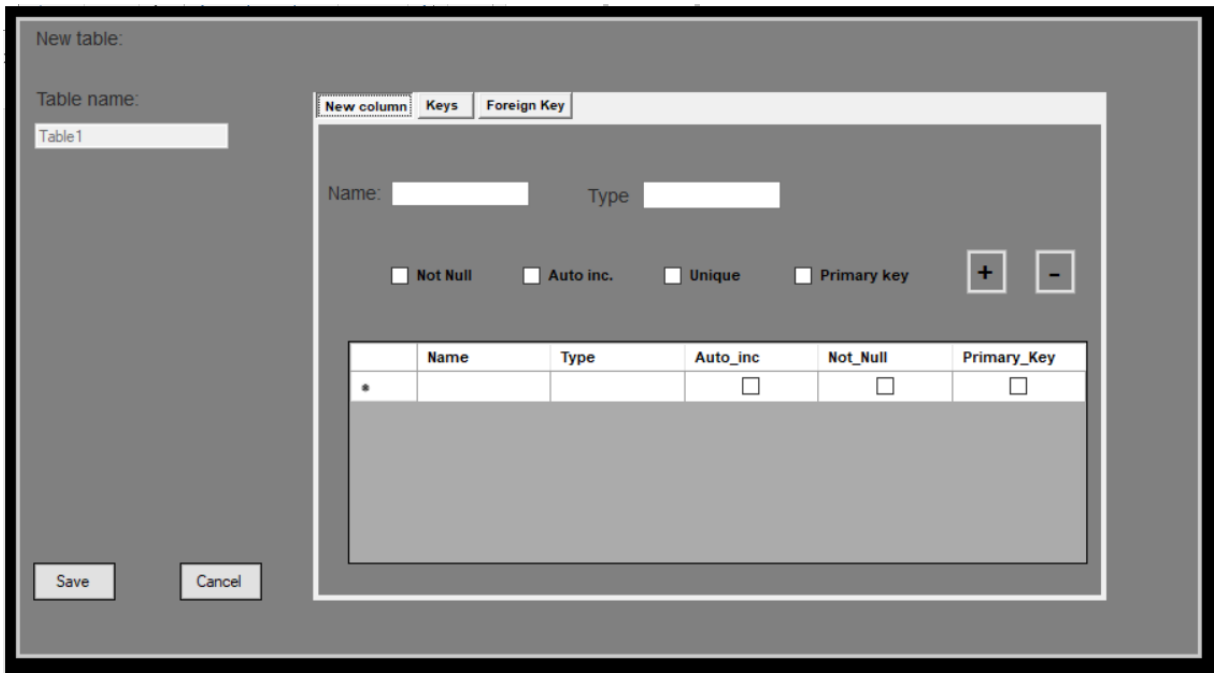
Potvrdom unosa tablica se miče iz popisa tablica u prostoru za projekte za taj određeni projekt i briše se iz mape u kojoj se nalazi.

Sa ovim poglavljem završavam sve funkcionalnosti koje se mogu raditi sa početne forme tj. sa početnog izgleda aplikacije i sad prelazimo na drugu formu, a to je unos nove tablice.

3.2. Unos nove tablice

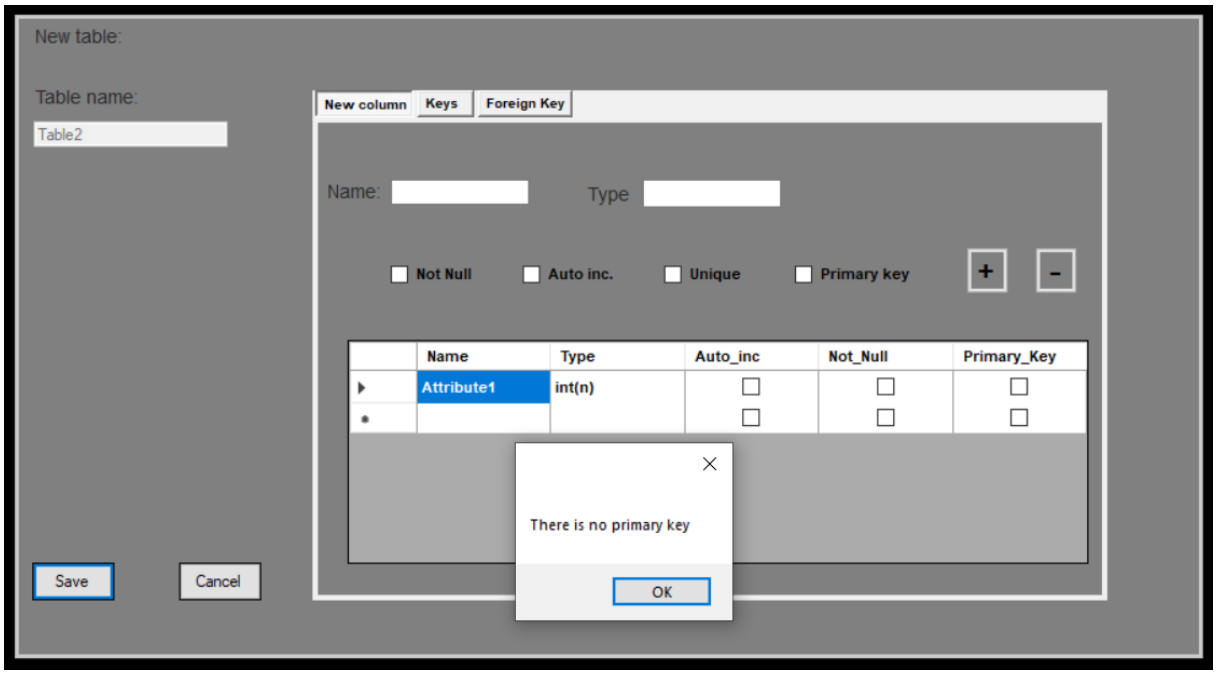
U jednom od prethodnog poglavlja imali samo uvid u jedan dio formu unosa nove tablice pa ćemo u ovom poglavlju pobliže opisati što se sve krije unutar te forme, zapravo dati ću uvid u sve opcije i načine implementacije unosa nove tablice. Pristupiti toj formi se može na dva načina. Prvi način je da kreiramo novu tablicu te joj dodijelimo ispravno ime i ako je sve u redu otvara se spomenuta forma. Drugi slučaj kako možemo pristupiti formi je na način da odaberemo već kreiranu tablicu i kliknemo na opciju „Open Table“ te nam se otvara forma sa već ispunjenim atributima tablice koju smo otvorili. Način koji ću odabrati za pobliže opisati

spomenuto je prvi način. Pa tako kada kreiramo tablicu sa nazivom koji želimo dobijemo sljedeći prozor:



Slika 25: Nova tablica (Izvor: vlastita izrada)

Prozor koji nam se javlja je fiksne veličine te se može povlačiti pritiskom lijevog klika miša kojega naravno držimo i povlačimo u željenom smjeru. Već je spomenuto da ime tablice nije moguće promijeniti sa ove forme. Početni izgled unosa nove tablice sadrži naziv forme, ime tablice, dva gumba („*Save*“ i „*Cancel*“) te prostor sa karticama „*New column*“, „*Keys*“ i „*Foreign Key*“. Te tri kartice će biti opisane malo niže u nastavku. Klikom na gumb „*Cancel*“ prvo se vrši provjera da li su uneseni atributi u tablici, u slučaju da nije unesen niti jedan atribut forma se gasi te se tablica briše pošto nema smisla držati praznu tablicu. Ako odlučimo kliknuti na gumb „*Save*“ također se vrše provjere. Prva provjera koja se vrši je ista kao i kod gumba „*Cancel*“ tako da se ne može spremiti prazna tablica, a druga provjera je da li postoji uneseni atribut koji je tipa Primarni ključ. U slučaju da je gumb „*Save*“ odabran, a u tablici nema primarnog ključa program ignorira zapovijed i baca grešku.

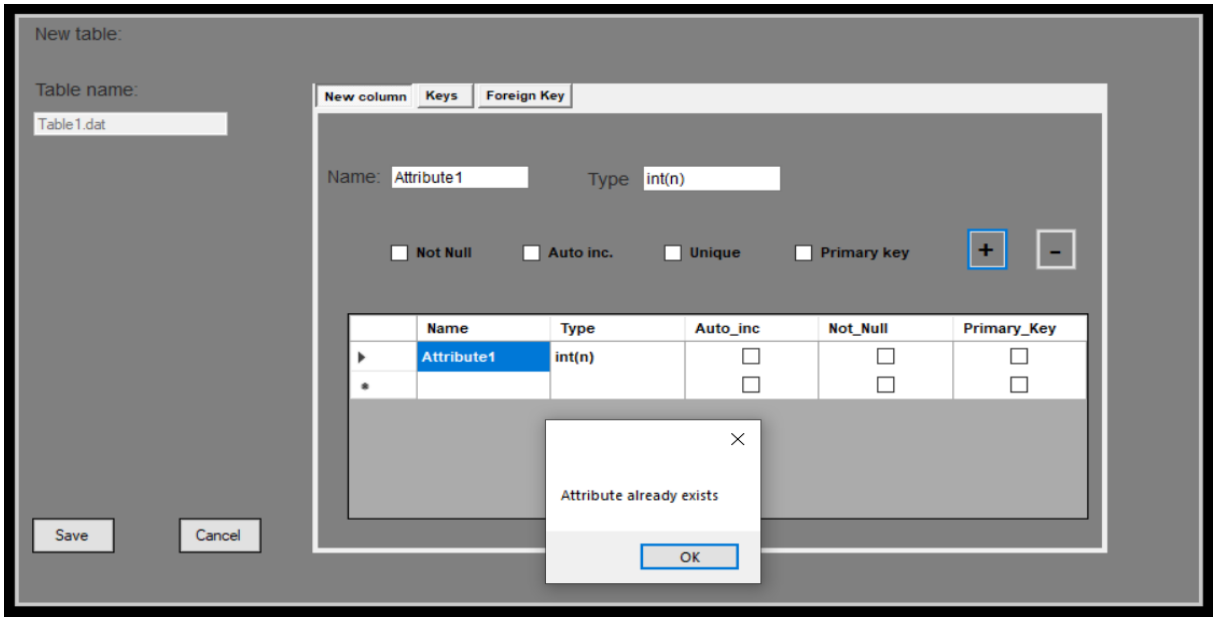


Slika 26: Nema primarnog ključa (Izvor: vlastita izrada)

Jedini način da se izađe iz te forme u tome slučaju je da se doda novi atribut koji će biti tipa Primarni ključ ili da se odabere opcija „*Cancel*“ koja nam briše tablicu u ovom slučaju.

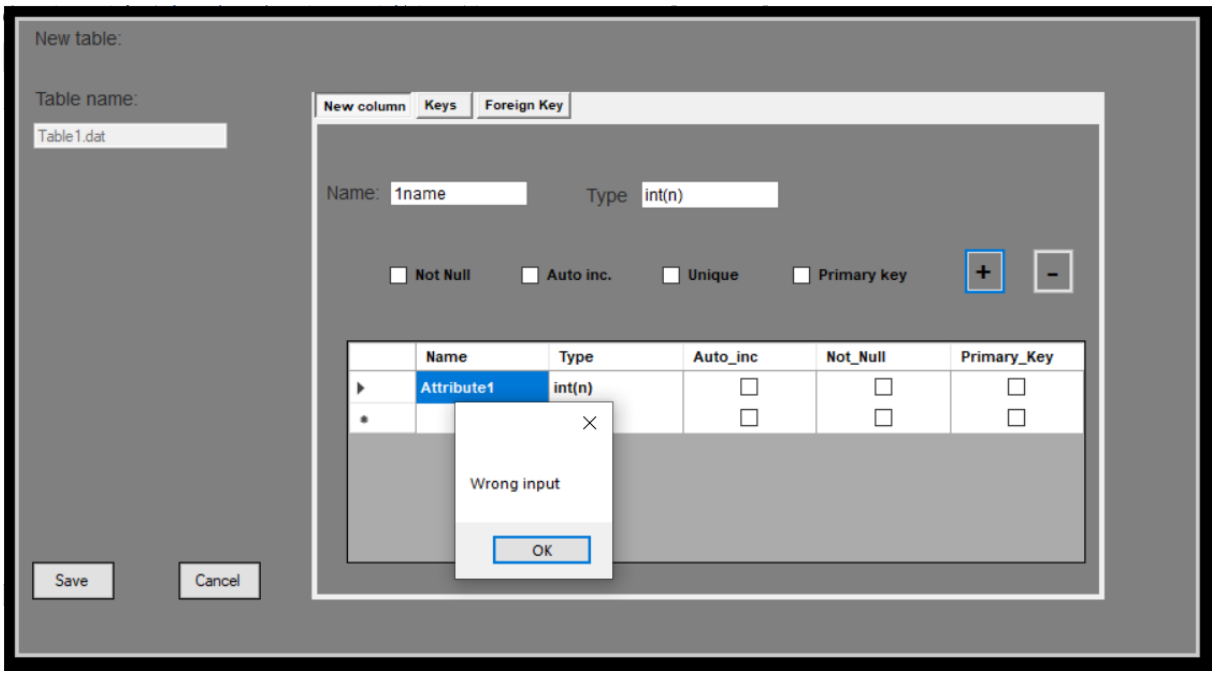
3.2.1.New column

Jedna od opcija kreiranja atributa nalazi se na kartici „*New column*“. Podaci koji se traže su ime atributa i tip atributa. Od dodatnih opcija imamo „*Not Null*“, „*Auto inc*.“, „Unique“ i „*Primary Key*“. Kod unosa imena atributa vrši se provjera jeli odabrano ime ispravno prema pravilima imenovanja. Prvo se provjera da li već postoji uneseni atribut.



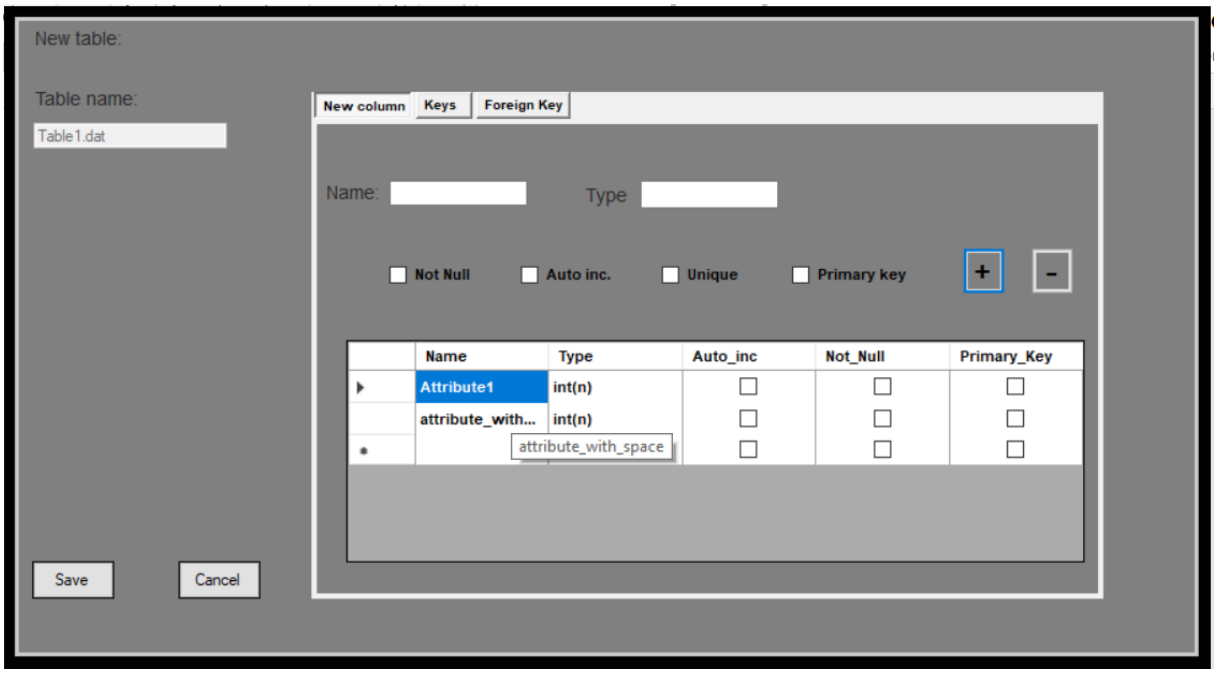
Slika 27: Provjera unesenog atributa (Izvor: vlastita izrada)

Ako se unese atribut sa imenom koji je već unesen javlja se greška te se akcija ne izvrši. Sljedeća provjera koja se vrši je provjera ispravnosti atributa. Dozvoljeni unosi imena atributa su sljedeći: Atribut smije samo sadržavati velika i mala slova, brojeve te znak „_“. Ime ne smije započinjati brojkom.



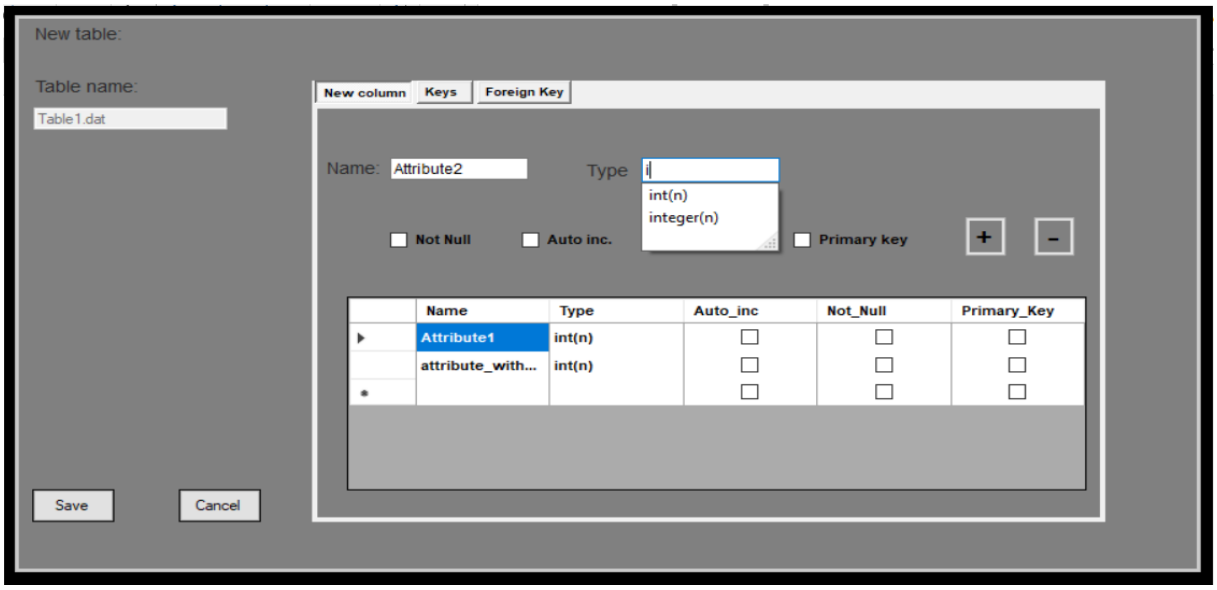
Slika 28: Primjer krivog unosa atributa (Izvor: vlastita izrada)

Ako se unese ime sa razmakom taj razmak se zamjenjuje znakom „_“.



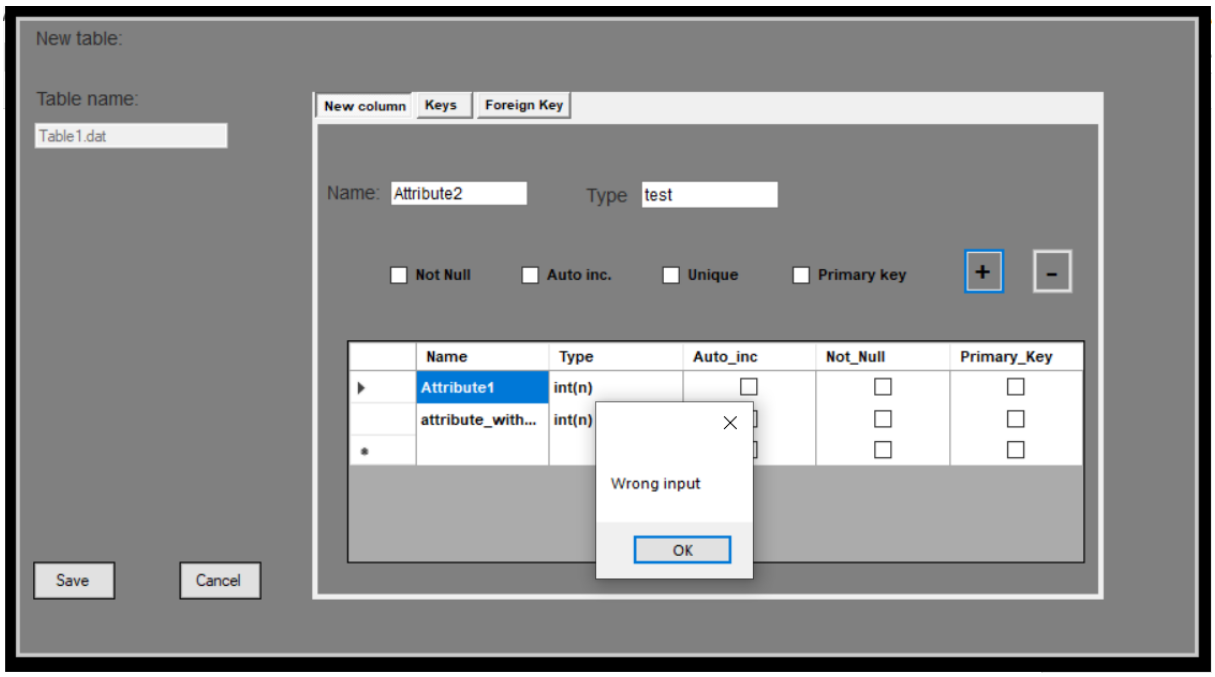
Slika 29: Unos atributa sa razmakom (Izvor: vlastita izrada)

Kao što se vrše provjere imena tako isto se i vrše provjere tipa podatka atributa. Prostor za unos tipa podataka je zapravo tipa „*AutoComplete*“ te prilikom upisa nudi dostupne atribute.



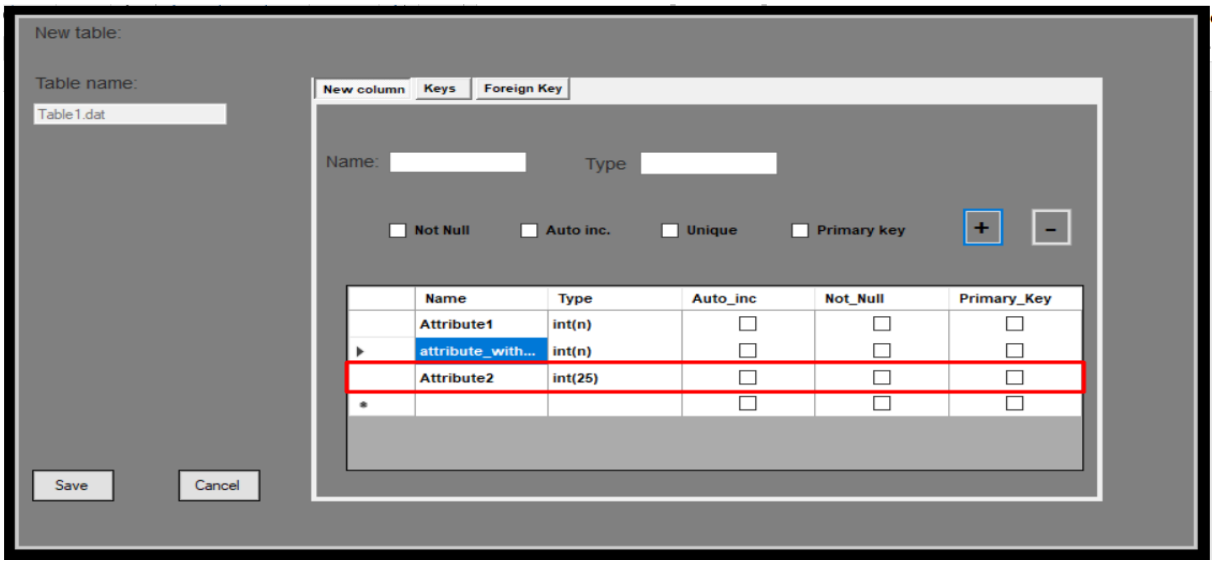
Slika 30: „*AutoComplete*“ opcija (Izvor: vlastita izrada)

Dostupni su samo oni tipovi podataka koji su ponuđeni „*AutoComplete*“ opcijom, svaki ostali unos rezultirat će greškom:



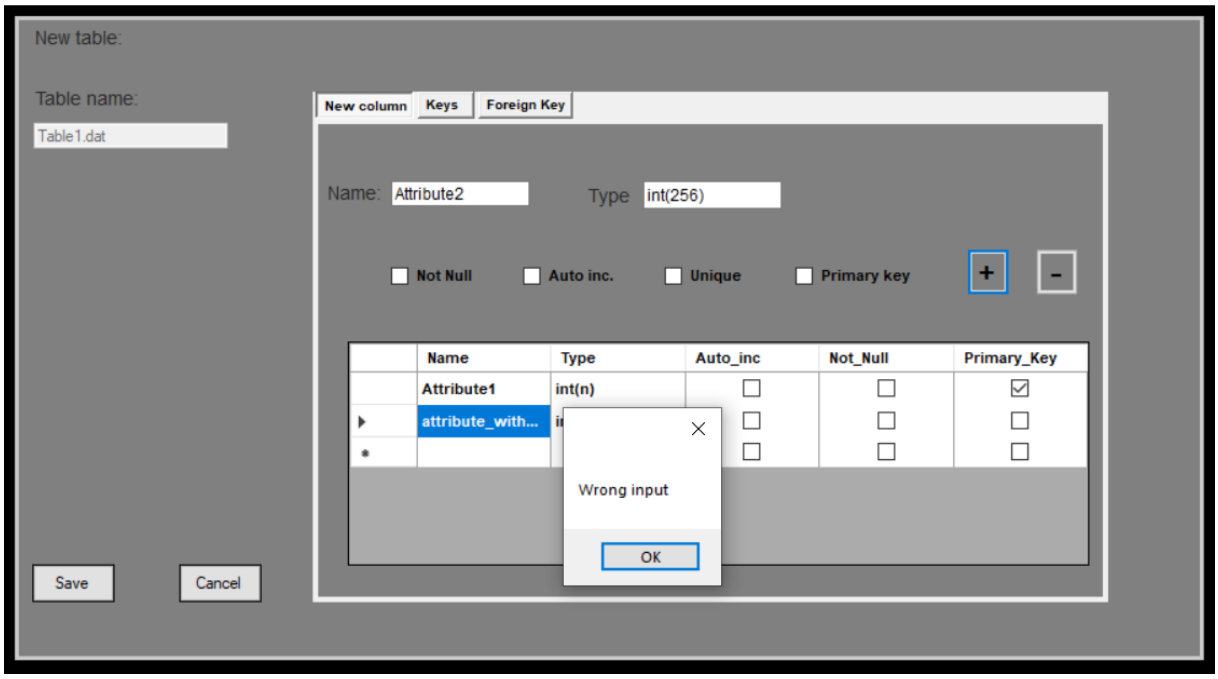
Slika 31: Krivi unos tipa podataka (Izvor: vlastita izrada)

Također za tip podatka umjesto slova „*n*“ unutar zagrada možemo i upisati željeni broj.



Slika 32: Unos tipa podataka sa određenom veličinom (Izvor: vlastita izrada)

Za svaki tip podataka je definirano kolika može biti najveća veličina tog podatka. Tako za tip podataka „*Integer*“ ne možemo unijeti vrijedno veću od 255 i manju od 0.

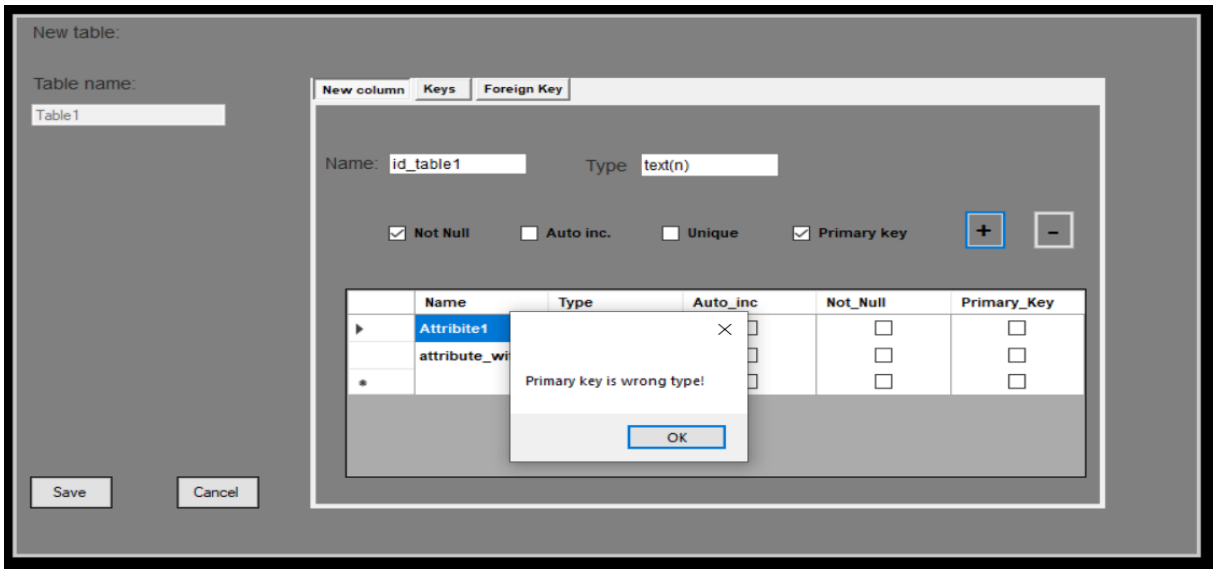


Slika 33: Krivi unos tipa podatka (Izvor: vlastita izrada)

Pošto u ovoj aplikaciji nema samog unosa podatka jedom kada su sve tablice definirane potreba za tom opcijom nije prijeko potrebna, ali je ipak implementirana. Sada smo riješili ispravne unose atributa što se tiče naziva i tipa podatka pa sada prelazimo na dodatne opcije.

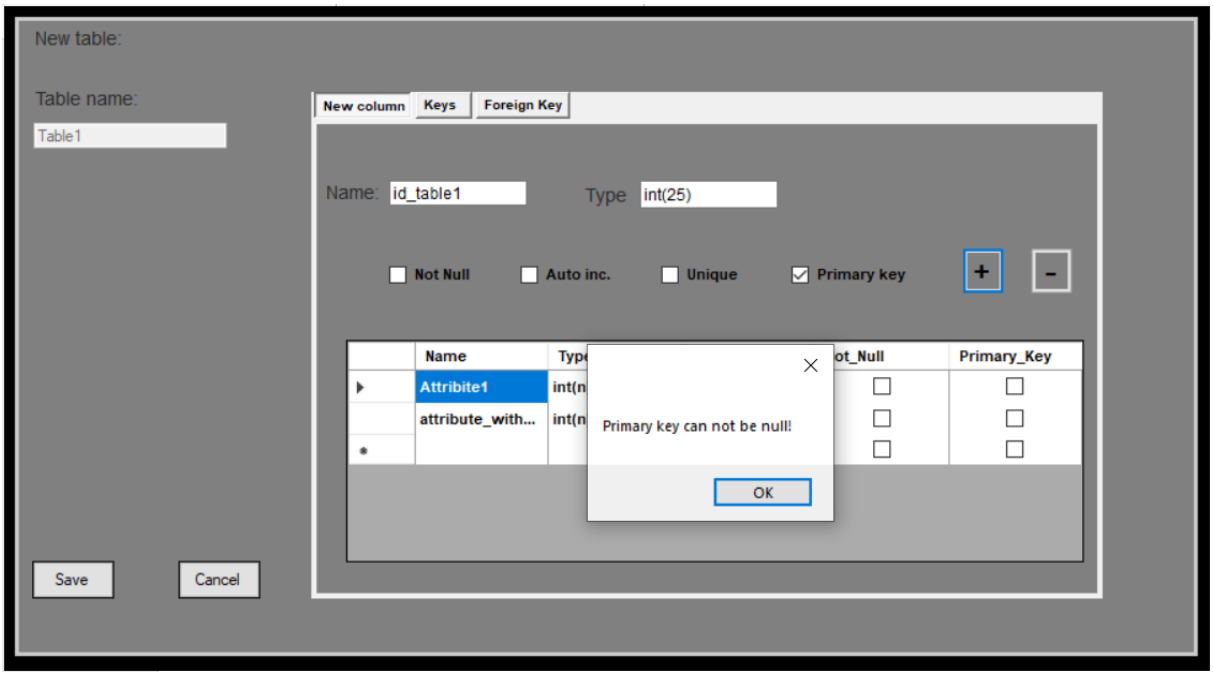
Na slici 33 možemo vidjeti da nam se nude dodatne četiri opcije koje želimo pridodati atributu.

Odaberemo li opciju „*Primary key*“ moramo biti sigurni da je tip podataka „*Integer*“.



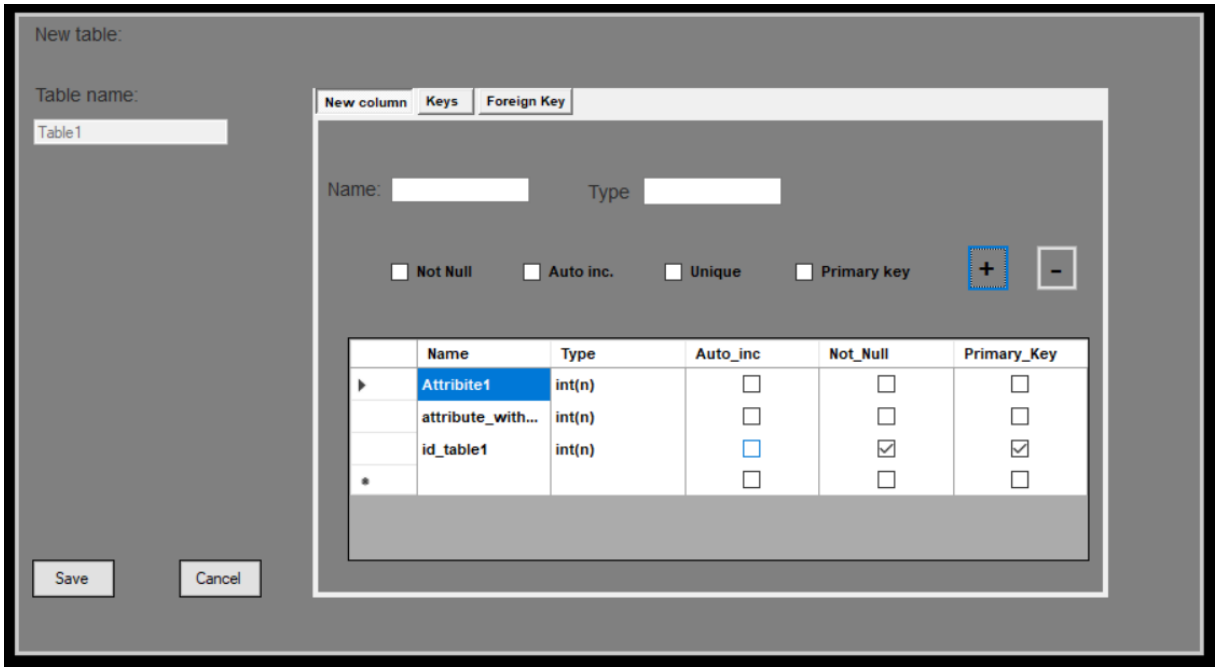
Slika 34: Primarni ključ je krivog tipa (Izvor: vlastita izrada)

Ova opcija je dodana samo iz jednostavnosti aplikacije te nema nikakve druge svrhe. U pravima sustavima za upravljanjima baze podataka primarni ključ može biti bilo kojega tipa (ili barem više nego u ovoj implementaciji). Još jedna od provjera unosa je da primarni ključ ne smije biti biti „*Null*“ pa se iz toga razloga mora odabrati opcija „*Not Null*“.



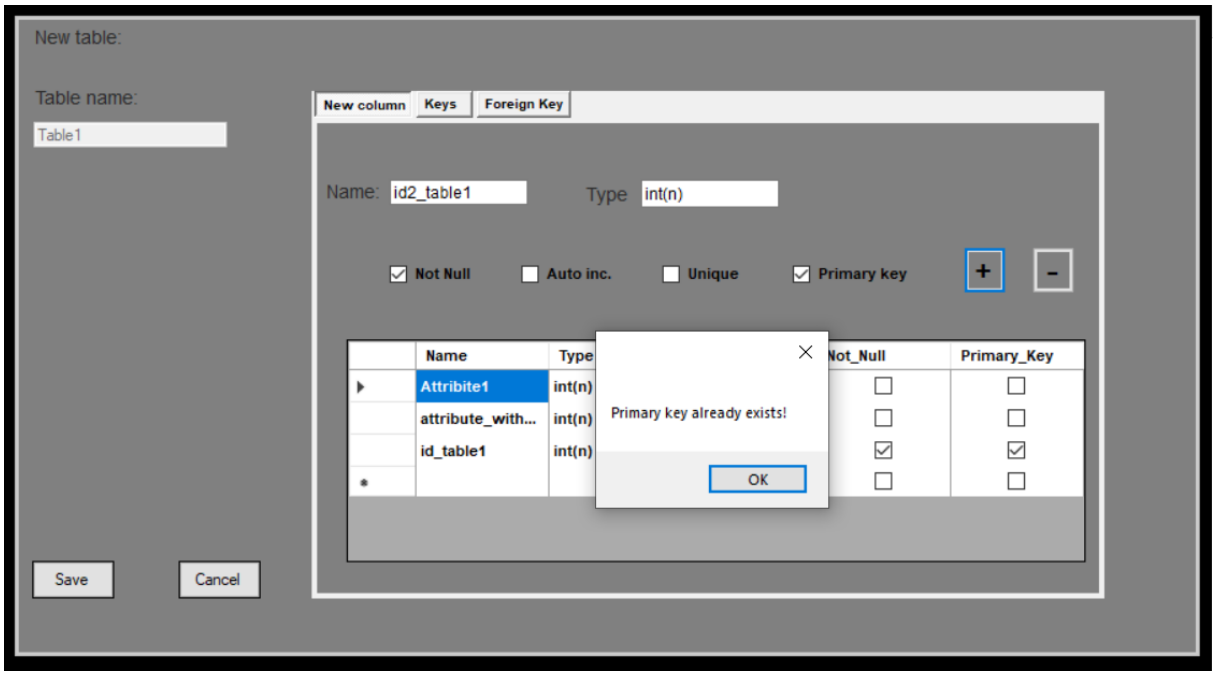
Slika 35: Primarni ključ je „*null*“ (Izvor: vlastita izrada)

Ako su svi uvjeti zadovoljeni primarni ključ će biti dodan u listu atributa.



Slika 36: Uspješno dodan atribut koji je primarni ključ (Izvor: vlastita izrada)

Sljedeće što još želim napomenuti je da se iz ovog prozora ne može dodati još jedan primarni ključ, ako se pokuša dodati novi atribut koji će biti primarni ključ aplikacija će zabraniti tu radnju te neće spremiti unos.

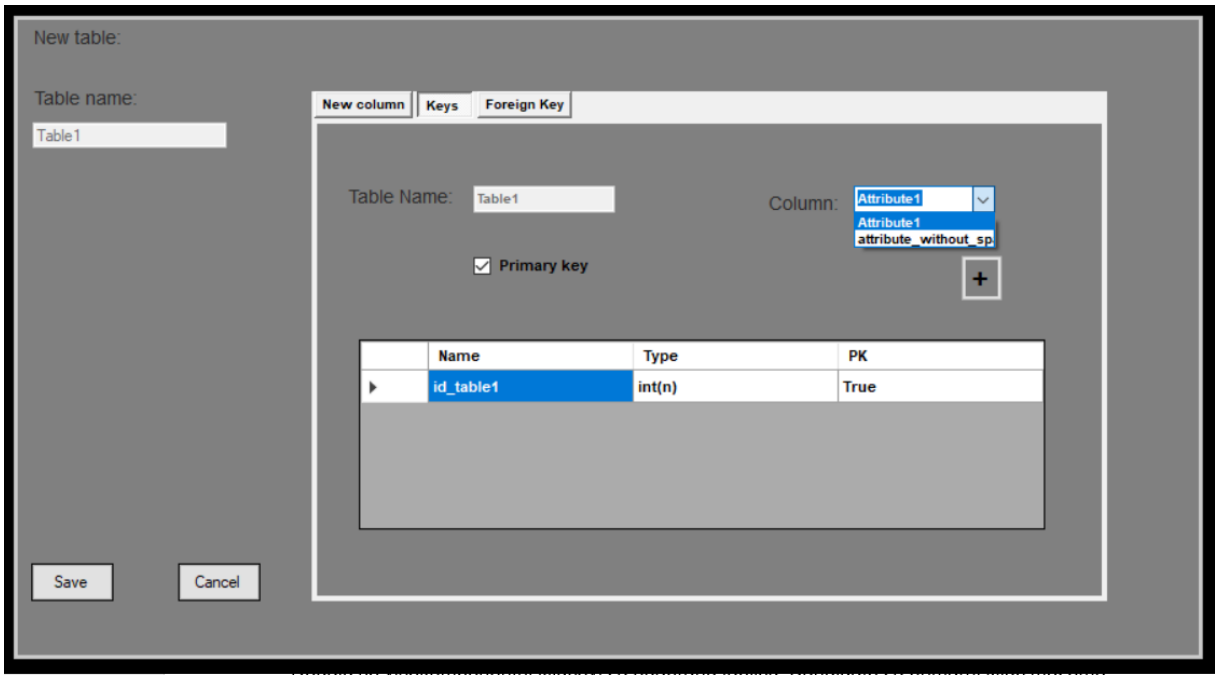


Slika 37: Unos novog primarnog ključa (Izvor: vlastita izrada)

Ostala nam je još samo jedna opcija za pojasniti, a to je brisanje atributa iz njegove liste, a to se događa na način da se odabere redak u kojem se nalazi atribut koji želimo obrisati te odaberemo gumb na kojem se nalazi znak „-„ . Vizualni prikaz nije moguć iz toga razloga jer je rezultat te akcije samo obrisani redak u listi atributa. Ako se obriše atribut koji je primarni ključ i više ne postoji ključ takvog tipa daljnjim odabirom na gumbove „*Save*“ i „*Cancel*“ izvršit će se akcije kao da je tablica prazna tj. kako je objašnjeno na početku poglavlju 3.2. Opcije „*Auto inc*“ i „*Unique*“ nemaju neku posebnu svrhu, ali se označuju u aplikaciji da su označene odnosno da nisu.

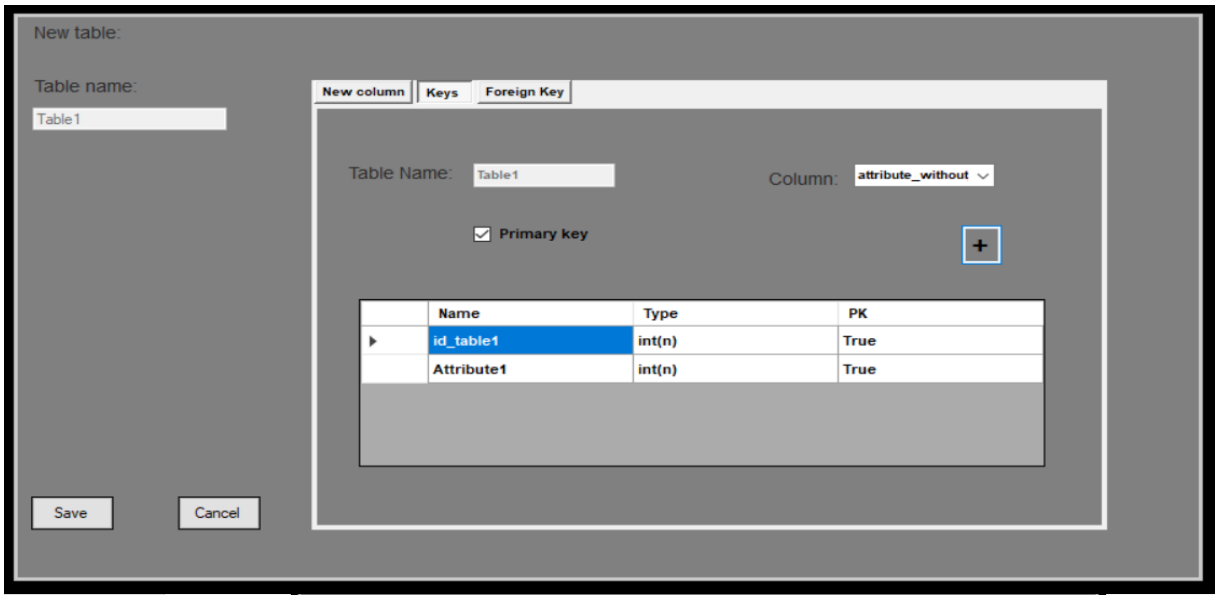
3.2.2. Keys

Na ovom djelu kartice imamo mogućnosti samo dodavati nove primarne ključeve za tablica kako bismo imali višekomponentni ključ.



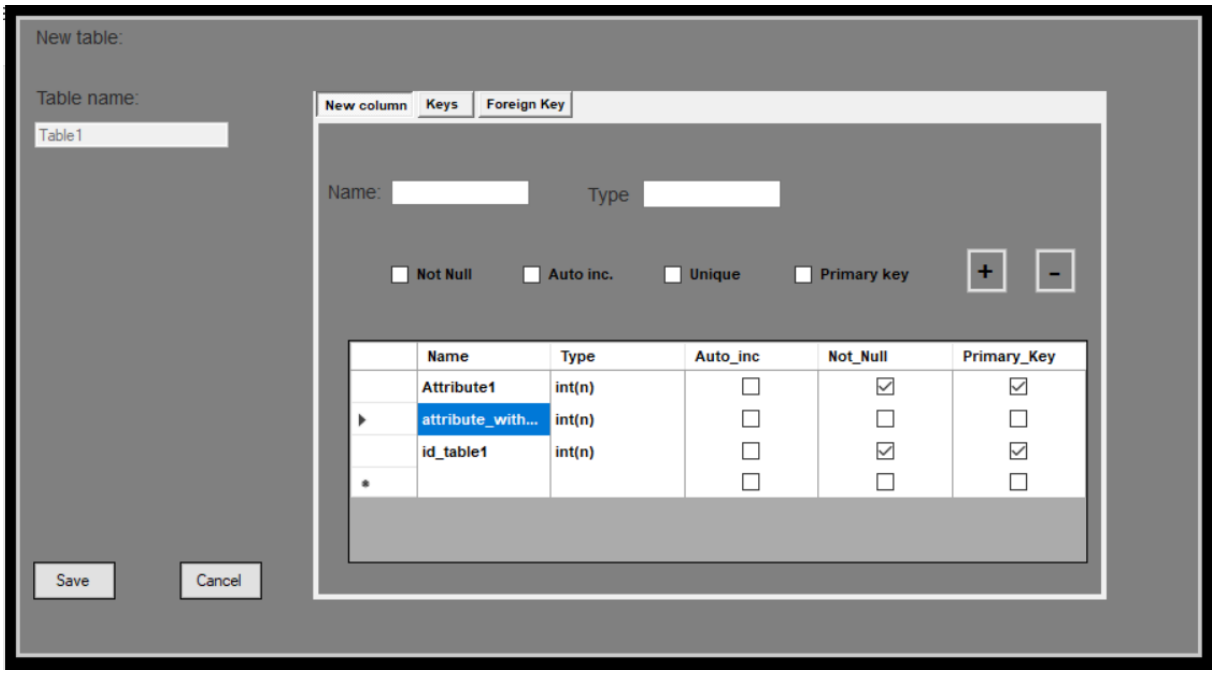
Slika 38: Unos višekomponentni primarnog ključa (Izvor: vlastita izrada)

Dodaju se višekomponentni ključevi za odabranu tablicu. Kandidate za primarni ključ možemo vidjeti u padajućem izborniku koji je nazvan „*Column*“. Kandidati su svi atributi te tablice koji su tipa „*Integer*“ i nisu već primarni ključ. Atribut se mijenja u primarni tako da se odabere željeni atribut prethodno objašnjenom akcijom i klikne se na gumb koji ima znak „+“ na sebi.



Slika 39: Novo dodani primarni ključ (Izvor: vlastita izrada)

Jednom kada je navedena akcija izvršena u listi atributa primarnih ključeva možemo vidjeti da je pridijeljen primarni ključ željenom atributa. Također se ta promjena vidi na kartici „*New column*“.



Slika 40: Rezultat promjene atributa u primarni ključ (Izvor: vlastita izrada)

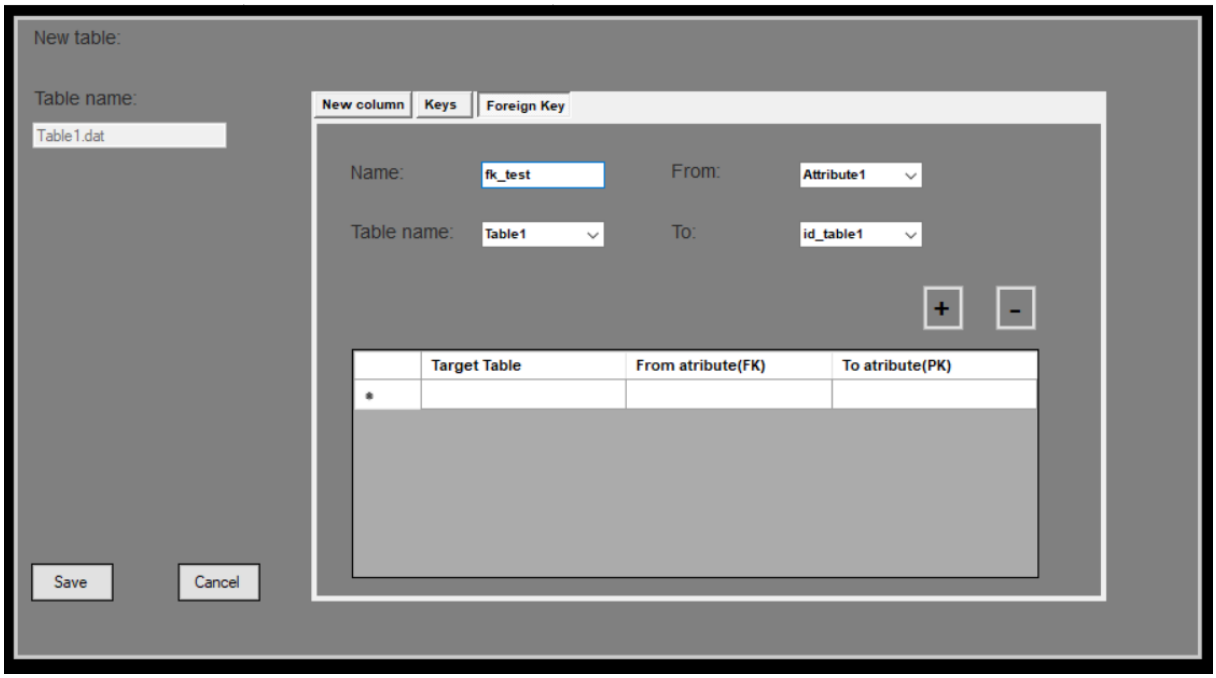
Na slici 40 možemo vidjeti promjenu koju je izazvana sa kartice „*Keys*“ te se ta promjena može samo poništiti na ovoj kartici tako da se obriše odabrani atribut na način koji je već spomenut te će se ta promjena dogoditi također i na kartici „*New column*“.

3.2.3. Foreign Keys

Jedna od važnijih funkcionalnosti aplikacije je zapravo ova. Ona nam zapravo omogućuje stvaranje veza između tablica, a to je zapravo cijela svrha konceptualnog modeliranja. Nakon implementacije ove zadanje funkcionalnosti imati ćemo sve potrebne podatke za izradu ERA dijagrama. Na sljedećoj slici 41 možemo vidjeti kako izgleda ta kartica u našem slučaju kroz koji prolazimo.

Slika 41:Kartica „Foreign keys“ (Izvor: vlastita izrada)

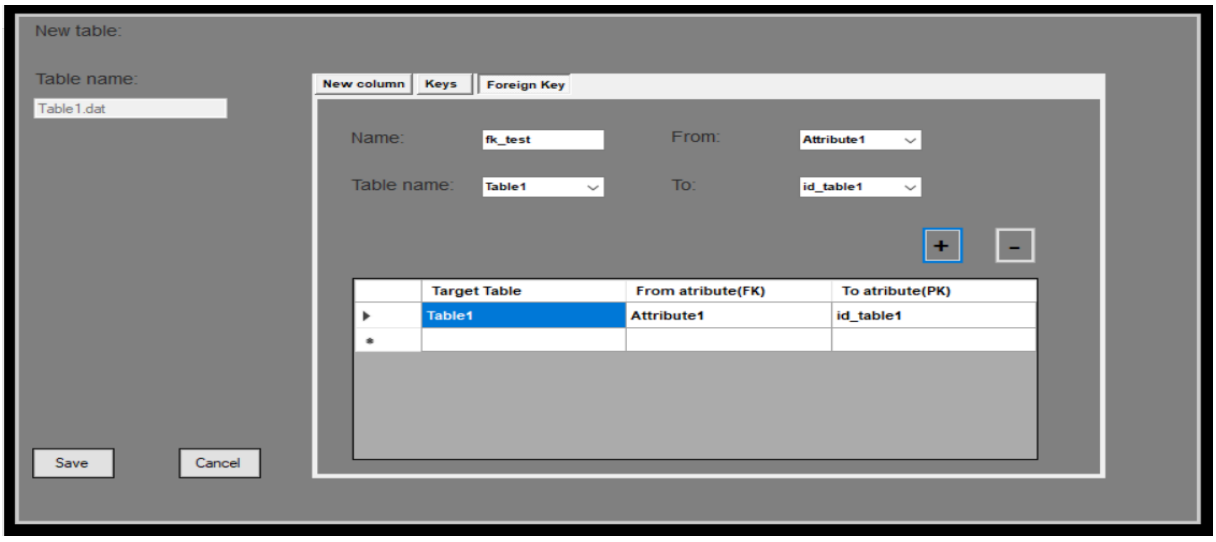
Izgled i uporaba izrade vanjskih ključeva je napravljena u stilu da bude jednostavna za korištenje. Prvo možemo dodati naziv vanjskog ključa koji nije toliko bitan u našem slučaju. Sljedeći izbori su nam „*Table name*“ koji je zapravo padajući izbornik u kojem se nalaze sve tablice u izabranom projektu. U padajućem izborniku se nalaze atributi koji su kandidati za izbor vanjskog ključa. A to su svi atributi trenutne tablice koji nisu primarni ključevi i tipa su „*Integer*“. Nakon odabira željene tablice prema kojoj ćemo raditi vezu popunjava se novi padajući izbornik označen sa „*To*“ u kojemu se zapravo nalaze svi primarni ključevi tablice na koju se cilja postaviti vezu.



Slika 42: Popunjeni izbornici u kartici „Foreign key“ (Izvor: vlastita izrada)

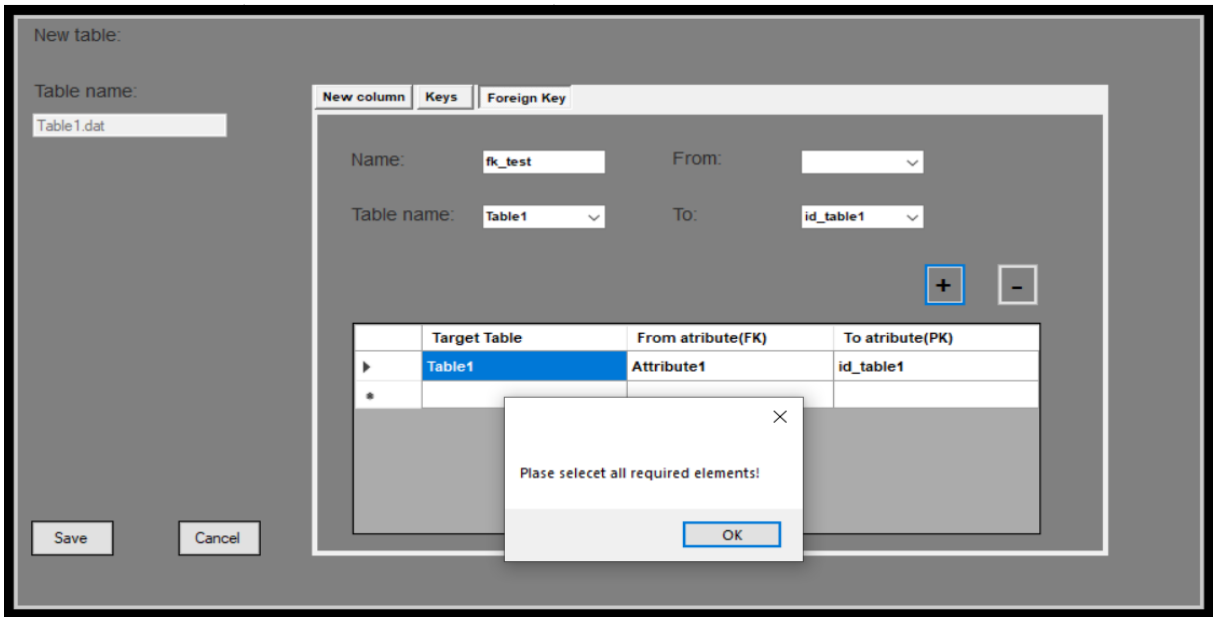
U našem slučaju imamo samo izrađenu jednu tablicu pa ćemo izraditi vezu na samu tablicu.

Odabrali smo znači tablicu koja je trenutno otvorena koja ima samo jedan primarni ključ pa će se veza raditi prema njemu i odabrali smo atribut koji je slobodan tipa „*Integer*“ sada nam samo preostaje potvrditi tu vezu, a to se događa na način da kliknemo na gumb sa oznakom „+“.



Slika 43: Dodan vanjski ključ (Izvor: vlastita izrada)

Bitna napomena je da se vrši provjera jesu li svi unosi ispravni te ako nisu aplikacija ne dopušta radnju.



Slika 44: Pogrešan unos vanjskog ključa (Izvor: vlastita izrada)

4. Izrada Era dijagram

Ovo poglavlje je rezervirano za opis programskog dijela izrade ERA dijagrama. Izabrao sam samo ovo poglavlje za objašnjavanje programskog djela jer je to bio najkompliciraniji dio i u njemu se možda čak nalazi najviše programskog koda. Ostali programski kodovi aplikacije nisu toliko zanimljivi i nisu zahtjevni kao navedeni dio. Podijeliti ću izradu ERA dijagrama na crtanje tablica i na crtanje veza. Prije nego što krenemo navesti ću klase u kojima se sve odvijalo.

- *formERADijagrama.cs*
- *GeneratingERA.cs*
- *Position.cs*
- *Canvas.cs*
- *LoadProjects.cs*

Možemo početi prvo sam kreiranjem tablica.

4.1. Izrada tablica

Krećemo od klase „*formERADijagrama*.cs“ u kojoj se nalazi „*clickEvent*“ koji se aktivira pritiskom na ranije spomenuto mjesto i od tuda će kretati kod skroz do kraja konceptualnog modeliranja.

```
private void lblGenerateERA_Click(object sender, EventArgs e)
        {
```

Prvo što se napravi pritiskom na navedeni „*clickEvenet*“ su instance klasa koje ćemo koristiti, ali prije svega toga brišemo sve što se nalazi na površini na kojoj ćemo crtati kako bismo bili sigurni da nema već kreiranih kontrola od prethodnog modeliranja, a to je napravljeno na sljedeći način:

```
Graphics gr = panelERADijagrama.CreateGraphics();
            gr.Clear(Color.White);
            panelERADijagrama.Controls.Clear();
```

„*panelERADijagrama*“ je naziv plohe na kojoj se odvija crtanje. Sada smo krenuli na instanciranje klasa koje ćemo koristiti i kreiranje liste kako bismo mogli u nju stavljati lokacije na kojima će se nalaziti tablice određene koordinatama i dimenzijama.

```
LoadProjects loadProjects = new LoadProjects();
GeneratingERA generatingERA = new GeneratingERA();
List<string>listOfTablePositionAndArea=newList<string>();
```

Korak dalje nam je inicijalizirati plohu za crtanje tj. odrediti gdje želimo da se naš model prikaže jer trenutno koristimo „*panelERADijagrama*“ koji ima svojstvo „*Fill*“ koje mu omogućava da prilikom promjene veličine prozora veličina navedene plohe ne ostane ista nego da se ona također prilagodi novom zaslonu zato je važno izračunati koordinate gdje ćemo početi crtati (gornji lijevi kut) te dužinu i visinu do koje smijemo crtati kako ne bismo izašli izvan okvira plohe za crtanje, a to je implementirano na sljedeći način:

```
int eraPanelWidth=panelERADijagrama.Width - (panel4.Width + panel1.Width);
int eraPanelHeight = panelERADijagrama.Height - panel3.Height;
int realEraPanelHeight = panelERADijagrama.Height;
int realEraPanelWidth = panelERADijagrama.Width;
int eraStartingPositionX = panel4.Width + panel1.Width;
int eraStartingPositionY = panel3.Height;
```

Izračunane su koordinate na način tako da smo oduzeli kreirane panele u kojima se nalazi imena projekata i ostalo. Sada proslijedimo te varijable preko konstruktora klasi „*Canvas.cs*“ za daljnju uporabu.

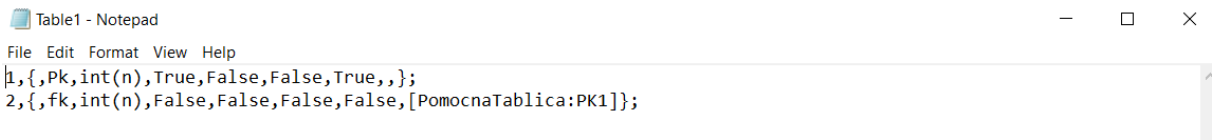
```
 Canvas canvas=newCanvas(eraPanelHeight,eraPanelWidth, eraStartingPositionX,
eraStartingPositionY,realEraPanelHeight,realEraPanelWidth);
```

Sada dolazimo do djela gdje uzimamo tablice iz projekta, a za dohvat tih tablica nam je potrebno samo ime projekta kojega trenutno modeliramo. Sada pozivamo instancu klase „*LoadProject.cs*“ i pomoću nje pozivamo metodu koja dohvaća imena svih tablica koje postoje u tom projektu te se na temelju toga radi putanja do same tablice kako bismo je mogli otvoriti u uzeti sve potrebne podatke koje je korisnik unio.

```
foreach (var table in loadProjects.LoadTables(projectName))
            {
                string tablePath = projectPath + "\\" + table;
```

Svaka tablica je spremljena u *.txt* datoteci u ranije definiranom formatu koji je sljedeći:

„RedniBrojAtributa,VitičastaZagrada,ImeAtributa,TipAtributa,NotNull,AutoInc,Unique,PK,FK, VitičastaZagrada;“



Slika 45: Zapis tablice (Izvor: vlastita izrada)

Sljedeći korak nam je kreiranje plohe koja će biti tablica, ali prije toga joj trebamo pridijeliti dimenzije te nove plohe koja će ovisiti u veličini znakovnog zapisa atributa i broju atributa. Te još jedna bitna karakteristika su koordinate tako nove kreirane plohe kako bismo znali gdje ćemo smjestiti novu tablicu. Dimenzije tablice dobivamo iz klase „*GeneratingERA.cs*“ tako da

proslijedimo putanju datoteke i dobijemo povratne informacije, koje ćemo još dodatno trebati modificirati zbog razmaka između atributa i veličini fonta koji ćemo koristiti pa programski kod izgleda ovako:

```
int height = (generatingERA.NumberOfLinesInFile(tablePath) * 15 + 17);
int width = (generatingERA.LengthOfNameAndType(tablePath, table) * 8) + 46;
```

Dio koji sada slijedi je dio koji će nam raspoređivati tablice po plohi kako bi razmak između svake tablice bio poprilično jednako podijeljen. Pokušaji raspoređivanja tablica koje sam isprobao prije nego sam došao do trenutnog izgleda su bili raspoređivanje tablica sa funkcijom koja traži slučajnu (eng. *Random*) točku te bi u binarnoj matrici `matrix[height,width]` provjeravalo je li navedena točka slobodna te u slučaju da je slobodna nacrtalo bi tablicu i označilo taj dio plohe zauzetim, ali taj način nije bio dobar zbog mogućnosti da se točka neće nikada naći tj. bilo bi puno ne iskorištenog prostora te vremenski gledano bilo bi dosta sporije od trenutno implementiranog načina.

Sada o načinu za koji sam se na kraju odlučio. Iskreno govoreći trebalo mi je duže vremena kako bi došao do željenog programskog rješenja jer problem nije bio u potpunosti jednostavan. Prvo pozivamo metodu koja se nalazi u klasi „*Canvas.cs*“ i spremamo rezultat u varijablu.

```
int[,] point = canvas.StartingTablePoints(biggestWidth,biggestHeight, width,
height, numberOfTables);
```

Parametre koje smo proslijedili su najveća dužina tablice koja postoji među drugim tablicama i najveća veličina, dimenzije trenutne tablice koja se crta, podsjetimo se da se sve to nalazi unutar „*foreach*“ petlje gdje uzimamo tablice po redu, te sveukupan broj tablica. Sada prelazimo u klasu „*Canvas.cs*“ kako bismo pobliže pogledali metodu sa kojom smo došli do koordinata. Za početak imamo globalne varijable kojima smo pridijelili vrijednosti pozivom konstruktora.

```
private int eraPanelWidth;
private int eraPanelHeight;
private int eraStartingPositionX;
private int eraStartingPositionY;
private static int[,] matrix;
private int newPointX;
private int realEraPanelHeight;
private int realEraPanelWidth;
private int j = 1;
private int biggestHeight;
private int biggestWidth;

public Canvas(int eraPanelHeight,int eraPanelWidth, int
eraStartingPositionX, int eraStartingPositionY, int realEraPanelHeight, int
realEraPanelWidth)
            {
                  this.eraPanelHeight = eraPanelHeight;
                  this.eraPanelWidth = eraPanelWidth;
```

```
            this.eraStartingPositionX = eraStartingPositionX;
            this.eraStartingPositionY = eraStartingPositionY;
            matrix = new int[realEraPanelHeight,realEraPanelWidth];
            this.newPointX = eraStartingPositionX;
            this.realEraPanelHeight = realEraPanelHeight;
            this.realEraPanelWidth = realEraPanelWidth;
        }
```

Kako ne bih nalijepio samo cijeli kod prvo ću objasniti kako sam riješio problem jednakog razmaka između tablica. Prvo što sam morao izračunati je bilo broj ćelija koje stanu na plohu. Ćelije su bile dimenzija „*biggestWidth*“ i „*biggestHeight*“ u slučaju da neka od tablica ne bude preko granica ćelije. Sljedeći korak je bio podijeliti koliko ćelija zapravo trebamo tj. koliko ćemo tablica imati što smo saznali ranije. Sada kada imamo te podatke izračunamo zauzetu površinu ćelija na način da pomnožimo površinu ćelije sa brojem tablica. Jednom kada imamo taj podataka trebamo izračunati koliko je površine preostalo i tu preostalu površinu podijeliti jednako na ćelije prema kojima smo se ranije ravnali. Sada ćemo zapravo napraviti ćelije novih dimenzija i to onoliko njih koliko imamo tablica.

Slika 46: Slika nove ćelije (Izvor: vlastita izrada)

Preostaje sada samo još pitanje kako samo dobio veličinu nove ćelije, a rješenje je zapravo vrlo elegantno. Pošto sam znao površinu stare ćelije i njemu dodijeljenu novu površinu zbrojio sam te dvije površine i dobio površinu nove ćelije te sam to iskoristio na način da inkrementiram

istovremeno dužinu stare ćelije i širinu za „1“ sve dok mi površina inkrementirajuće ćelije nije jednaka novoj ćeliji te sam onda tako dobio dimenzije nove ćelije.

```
private int  FindNewCellDimensions(int biggestWidth, int biggestHeight, int
rectangleSurface, int surfacePerTableAdded)
            {
                  int newRectangleSurface = biggestWidth * biggestHeight;
                  int i = 0;
                  while (newRectangleSurface <= (rectangleSurface +
surfacePerTableAdded)){
                        biggestWidth++;
                        biggestHeight++;
                        newRectangleSurface = biggestWidth * biggestHeight;
                        i++;
            }
                  this.tableSpace = i / 2;
                  return i;
            }
```

Sada je sve spremno za postavljanje koordinata tablice koje su bile centrirane unutar novokreirane ćelije pomoću sljedeće metode.

```
public int[,] StartingTablePoints(int biggestWidht,int biggestHeight,int
width,int height,int numberOfTables)
            {
            int numberOfRows = eraPanelHeight / biggestHeight;
            int numberOfCells = eraPanelWidth / biggestWidht;
            int numberOfSquares = numberOfRows * numberOfCells;
            int surfaceOfUnusedCells = (numberOfSquares - numberOfTables) *
(biggestHeight * biggestWidht);
                  int surfacePerTableAdded = surfaceOfUnusedCells /
numberOfTables;
                  int rectangleSurface = biggestWidht * biggestHeight;
                  this.biggestHeight = biggestHeight;
                  this.biggestWidth = biggestWidht;

                  int i=FindNewCellDimensions(biggestWidht, biggestHeight,
rectangleSurface,surfacePerTableAdded)/2;

                  //Dodajem prvi red
                  if (j == 1)
                  {
                        eraStartingPositionX += i;
                        eraStartingPositionY += i;
                        j++;
                        PaintMatrix(eraStartingPositionX,
eraStartingPositionY, height, width);
                        PaintMatrixMore(eraStartingPositionX,
eraStartingPositionY, height, width);

                        return new int[eraStartingPositionX,
eraStartingPositionY];
                  }

                  //Provjeravam da li treba ići u novi red
                  if (eraStartingPositionX +biggestWidht*2+i*2>=
realEraPanelWidth)
                  {
```

```
                        eraStartingPositionY += biggestHeight + i;
                        eraStartingPositionX = newPointX+i;
                        PaintMatrix(eraStartingPositionX,
eraStartingPositionY, height, width);
                        PaintMatrixMore(eraStartingPositionX,
eraStartingPositionY, height, width);
                        return new int[eraStartingPositionX,
eraStartingPositionY];
                    }
                    eraStartingPositionX += biggestWidht + i*2;
                    PaintMatrix(eraStartingPositionX, eraStartingPositionY,
height, width);
                    PaintMatrixMore(eraStartingPositionX,
eraStartingPositionY, height, width);

                    return new int[eraStartingPositionX,
eraStartingPositionY];}
```

Vraćamo se u „*formERADijagrama.cs*“ i nastavljamo ići dalje kroz „*clickEvent*“. Sada imamo sve potrebno za crtanje tablica te nam samo preostaje kreirati potrebne kontrole i definirati kako će nam tablica izgleda. Slijedno kreiramo prvo praznu plohu dimenzija tablice i na poziciji rezervirano za nju te također spremamo te koordinate u listu zbog daljnjeg crtanja veza kako bismo znali koja tablica se nalazi na kojoj poziciji.

```
Panel tablePanel = new Panel();
tablePanel.Size = new System.Drawing.Size(width, height);
tablePanel.Location = new Point(pointX, pointY);
tablePanel.BorderStyle = BorderStyle.FixedSingle;
tablePanel.BringToFront();
Color myCOlor = Color.FromArgb(169, 186, 171);
tablePanel.BackColor = myCOlor;
panelERADijagrama.Controls.Add(tablePanel);
string positionAndName = "[(" + pointX + "," + pointY + ");(" + width + "," + height +
");:" + table + "]";
listOfTablePositionAndArea.Add(positionAndName);
```

Trebamo smjestiti ime tablice u sredinu plohe tablice i odvojiti crtom ime tablice od atributa

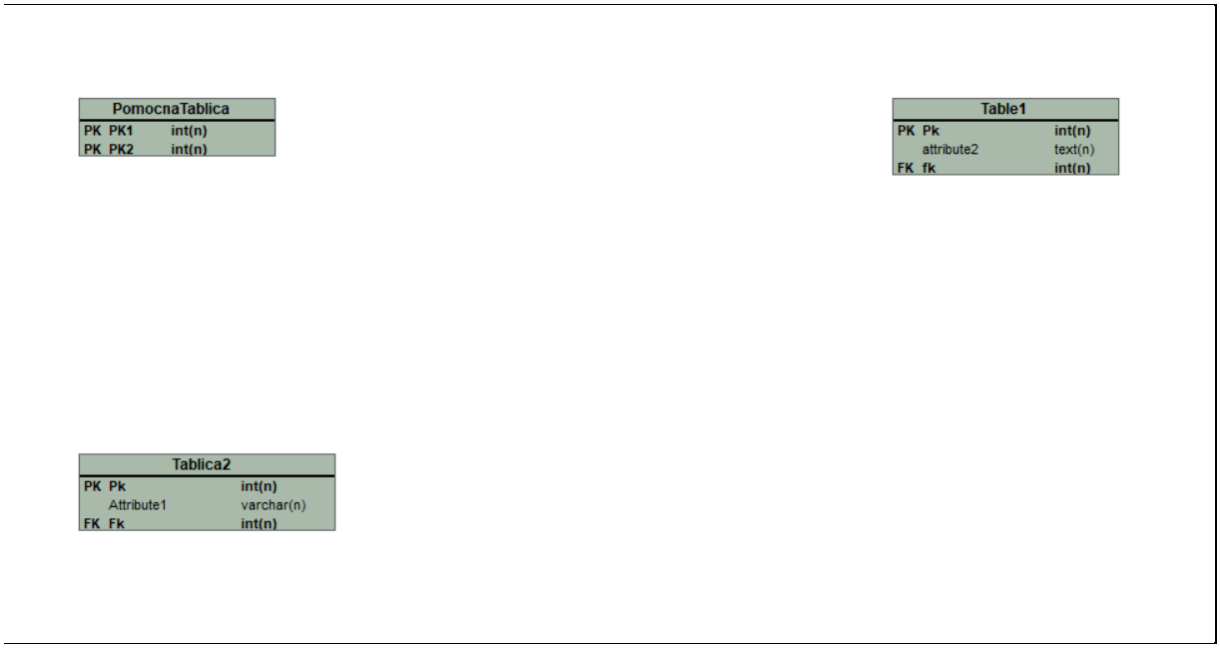
```
Label lblTableName = new Label();
lblTableName.Text = table.Substring(0, table.IndexOf('.'));
lblTableName.Font = new Font("Arial", 9, FontStyle.Bold);
int tableNameWidth = table.Substring(0, table.IndexOf('.')).Length * 8;
lblTableName.AutoSize = true;
lblTableName.Location = new Point((width - tableNameWidth) / 2, 0);
tablePanel.Controls.Add(lblTableName);

tablePanel.Paint += new PaintEventHandler(tablePanelPaint);
tablePanel.Refresh();
```

Sljedeće sam radio smještanje atributa u prostor tablice i to sam radio redom kako bi na vrhu tablice bili primarni ključevi, u sredini atributi, a na kraju vanjski ključevi. Tako da sada imamo opet par „*foreach*“ petlja koje opet prolaze kroz zapise tablica i traže redom navedene stvari. Kada petlja nađe što traži kreira se nova oznaka koja će sadržavati ime atributa i tip atributa.

```
foreach (var PK in generatingERA.ListOfPrimaryKeyInTable(tablePath))
                {
                    string[] parts = PK.Split(',');

                    //Creating label for primary keys
                    Label lbPK = new Label();
                    lbPK.Text = parts[0].ToString();
                    lbPK.Font = new Font("Arial", 8, FontStyle.Bold);
                    lbPK.Location = new System.Drawing.Point(20, newY);
                    lbPK.AutoSize = true;
                    tablePanel.Controls.Add(lbPK);

                    //Creating label for primary key type
                    Label lbTypePK = new Label();
                    lbTypePK.Text = parts[1].ToString();
                    lbTypePK.Font = new Font("Arial", 8, FontStyle.Bold);
                    lbTypePK.Location = new System.Drawing.Point(width -
(width - (generatingERA.BiggestAttribute(tablePath)) * 8) + 46, newY);
                    lbTypePK.AutoSize = true;
                    tablePanel.Controls.Add(lbTypePK);

                    //Creating Label for PK indicator
                    Label lblPk1 = new Label();
                    lblPk1.Text = "PK";
                    lblPk1.Font = new Font("Arial", 8, FontStyle.Bold);
                    lblPk1.Location = new System.Drawing.Point(0, newY);
                    lblPk1.AutoSize = true;
                    tablePanel.Controls.Add(lblPk1);
                    newY += 15;
                    lbTypePK.BringToFront();


                }


                //Loop for placing labels that are normal atributes
                foreach (var attribute in
generatingERA.AtributeNameAndTypeInFile(tablePath))
                {
                    string[] parts = attribute.Split(',');

                    //Creating labels for every atribute
                    Label lb = new Label();
                    lb.Text = parts[0].ToString();
                    lb.Font = new Font("Arial", 8, FontStyle.Regular);
                    lb.Location = new System.Drawing.Point(20, newY);
                    lb.AutoSize = true;
                    tablePanel.Controls.Add(lb);

                    //Creating label for type
                    Label lbType = new Label();
                    lbType.Text = parts[1].ToString();
                    lbType.Font = new Font("Arial", 8, FontStyle.Regular);
                    lbType.Location = new System.Drawing.Point(width -
(width - (generatingERA.BiggestAttribute(tablePath)) * 8) + 46, newY);
                    lbType.AutoSize = true;
                    tablePanel.Controls.Add(lbType);

                    newY += 15;
                    lbType.BringToFront();
```

```
                }

                //Adding foreing keys to table panel
                foreach (var PK in
generatingERA.ListOfFKAtributeAndType(tablePath))
                {
                    string[] parts = PK.Split(',');

                    //Creating label from FK keys
                    Label lbFK = new Label();
                    lbFK.Text = parts[0].ToString();
                    lbFK.Font = new Font("Arial", 8, FontStyle.Bold);
                    lbFK.Location = new System.Drawing.Point(20, newY);
                    lbFK.AutoSize = true;
                    tablePanel.Controls.Add(lbFK);

                    //Creating label for FK key type
                    Label lbTypeFK = new Label();
                    lbTypeFK.Text = parts[1].ToString();
                    lbTypeFK.Font = new Font("Arial", 8, FontStyle.Bold);
                    lbTypeFK.Location = new System.Drawing.Point(width -
(width - (generatingERA.BiggestAttribute(tablePath)) * 8) + 46, newY);
                    lbTypeFK.AutoSize = true;
                    tablePanel.Controls.Add(lbTypeFK);

                    //Creating Label for FK indicator
                    Label lblFk1 = new Label();
                    lblFk1.Text = "FK";
                    lblFk1.Font = new Font("Arial", 8, FontStyle.Bold);
                    lblFk1.Location = new System.Drawing.Point(0, newY);
                    lblFk1.AutoSize = true;
                    tablePanel.Controls.Add(lblFk1);
                    newY += 15;
                    lbTypeFK.BringToFront();

                }
```

Rezultat programskog djela koda je prikazan na slici 47.



Slika 47: Crtanje tablica (Izvor: vlastita izrada)

Ovom slikom ćemo završiti kreiranje tablica i krenuti na kreiranje veza.

4.2. Izrada veza

Izrada veza također nije bio jednostavan posao za implementirati iz razloga jer veze ovise o više varijabla. Prema početnoj zamisli veze na tablice su trebale biti crne boje i trebale su prolaziti slobodnim putem između tablica, ali sam ubrzo uvidio da to nije bilo najbolje rješenje u nekim slučajevima jer je dolazilo do beskonačne petlje. Idemo krenuti prolaziti kroz kod pa će nam biti jasnije kako je trebalo biti implementirano i kakvo je rješenje na kraju odabrano. Prije nego što počnemo trebam napomenuti da veze krećemo crtati tek kada su sve tablice jednom kreirane i znamo im početne koordinate, gornji lijevi kut, crtanja i njihove dimenzije.

Prvo što radimo kreiramo „*foreach*“ petlju koja će prolaziti kroz ranije zapisanu listu koja sadrži podatke o koordinatama i dimenzijama. Kreće se redom kako su tablice bile kreirane. Lista je bila tipa „*String*“ što možda nije bio najbolji način da sadržava sve te sve podatke u jednom „*Stringu*“ pa ih je prvotno trebalo razdvojiti na sljedeći način:

```
   foreach (var item in listOfTablePositionAndArea)
            {
                string[] parts = item.Split(';');

                string tableName = item.Substring(item.LastIndexOf(':') +
1, item.IndexOf(']') - item.IndexOf(':') - 1);
```

```
                int pointX = int.Parse(parts[0].Substring(item.IndexOf('(')
+ 1, item.IndexOf(',') - 2));
                int pointY = int.Parse(parts[0].Substring(item.IndexOf(',')
+ 1, item.IndexOf(')') - item.IndexOf(',') - 1));
                int width =
int.Parse(parts[1].Substring(parts[1].IndexOf('(') + 1,
parts[1].IndexOf(',') - 1));
                int height =
int.Parse(parts[1].Substring(parts[1].IndexOf(',') + 1,
parts[1].IndexOf(')') - parts[1].IndexOf(',') - 1));
```

Kao što rekoh nije najbolji prizor, ali rješenje funkcionira. U tom koraku dobili smo ime tablice te sada redom provjeravamo da li navedena tablica sadrži Vanjski ključ na neku drugu tablicu. U slučaju da navedena tablica sadrži vanjski ključ trebamo dohvatiti lokaciju tablice „*Roditelj*“ i ostale potrebne podatke, a to radimo na način da opet vrtimo navedenu listu koja sadrži vizualne zapise o svim tablicama. Kada nađemo tablicu „*Roditelj*“ spremamo koordinate u nove varijable i spremni smo za sljedeći korak.

```
  if (generatingERA.TableHasFK(projectPath + "\\" + tableName))
                {
                    foreach (var FK in
generatingERA.ListOfTableFK(projectPath + "\\" + tableName))
                    {

                        foreach (var tableFK in listOfTablePositionAndArea)
                        {
                            string[] newParts = tableFK.Split(';');
                            string parentTable =
newParts[2].Substring(newParts[2].IndexOf(':') + 1,
newParts[2].IndexOf('.') - newParts[2].IndexOf(':') - 1);


                            if (FK == parentTable)
                            {
 int newPointX = int.Parse(newParts[0].Substring(newParts[0].IndexOf('(') +
1, newParts[0].IndexOf(',') - 2));
                                int newPointY =
int.Parse(newParts[0].Substring(newParts[0].IndexOf(',') + 1,
newParts[0].IndexOf(')') - newParts[0].IndexOf(',') - 1));
                                int newWidth =
int.Parse(newParts[1].Substring(newParts[1].IndexOf('(') + 1,
newParts[1].IndexOf(',') - 1));
                                int newHeight =
int.Parse(newParts[1].Substring(newParts[1].IndexOf(',') + 1,
newParts[1].IndexOf(')') - newParts[1].IndexOf(',') - 1));
```

Napominjem da vjerojatno postoji lakši način za implementaciju ovog dijela rješenja, ali sam se ja odlučio za ovakav način implementacije.

Sada smo došli do lakšeg dijela implementacije veza, a to su tipovi veza. Tipovi veza koji su implementirani su

- 1:1
- 1:M

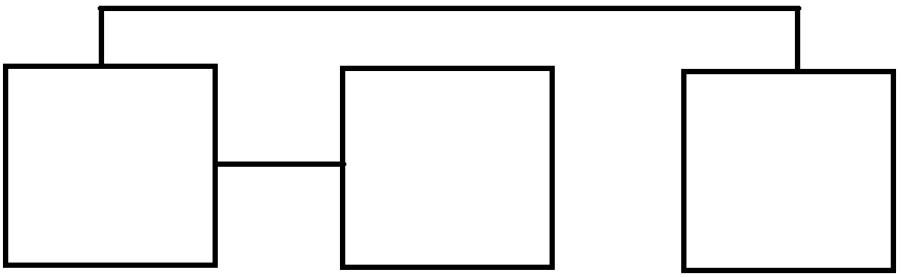
- M:N.

Program će implementirati samo takve tipove veza bez notacija (0:1, 0:M i ostalo). Da bismo saznali koje veze će biti na kojoj tablici trebamo pozvati metodu koja je kreirana u tu svrhu. Pa se sada selimo u klasu „*GeneratingERA.cs*“ koja ima metodu „*TypeOfConnection*“ koja prima ime tablice „*Dijete*“ i tablice „*Roditelj*“ i putanju projekta. Otvaramo zapise tablice „*dijete*“ i tražimo atribut koji je vanjski ključ. Ali prije toga kreiramo metodu tipa „*bool*“ koja će nam u slučaju ako tablica ima više vanjskih ključeva provjeravati je li taj atribut već provjeren kako se ne bi uzeo dva puta isti atribut u razmatranje. Dođemo do atributa koji je vanjski ključ i prije svega provjeravamo da li je taj atribut veza na istu tablicu. U slučaju da je taj uvjet istina provjeravamo je li dio atributa koji sadrži vrijednost atributa „*Unique*“ postavljen na „*True*“ veza će biti 1:1 na tu istu tablicu, ako to nije slučaj onda znači da veza mora biti 1:M. Ako atribut koji je vanjski ključ nije povezan sa svojom tablicom provjeravamo isto kao u prethodnom koraku samo nam je zapis koji će metoda vratiti je drugačiji. Metoda je tipa „*Dictionary<String,String>*“ i vraća tip veze u kojem indekse se nalazi: indeks 0 sadrži tip tablice „*Dijete*“, indeks 1 sadrži tip tablice „*Roditelj*“. Kod je poprilično dug pa ga iz toga razloga neću staviti ovdje iz toga razloga. Mislim da čitatelj može shvatiti na koji način otprilike je to implementirano. Ali kod koji se koristi u formi „*formERADijagrama.cs*“ će sažeti više manje što je se time postiže.

```
Dictionary<string,string> typeOfConnection =
generatingERA.TypeOfConnection(tableName, parentTable, projectPath);
                                string childTableConnectionType = "";
                                string parentTableConnectionType = "";
                                try
                                {
                                    foreach (var type in typeOfConnection)
                                    {
                                      childTableConnectionType = type.Key;
                                      parentTableConnectionType=type.Value;
                                    }
                                }
                                catch { }
```

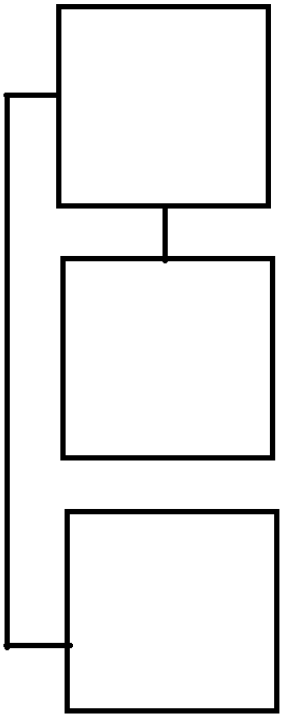
Lakši dio crtanja veza je završen i sada dolazimo do definiranja početnih točaka tablice pošto za sada znamo samo jednu točku tablice, a to je gornji lijevi kut. Odlučio sam se na ovaj korak iz razloga kako bi pregled veza bio što jasniji. Šest je uvjeta koje sam definirao, a oni su sljedeći

- Tablica „*Roditelj*“ se nalazi u istom redu kao i tablica „*Dijete*“, tablice su jedna do druge ili tablice nisu jedna do druge.



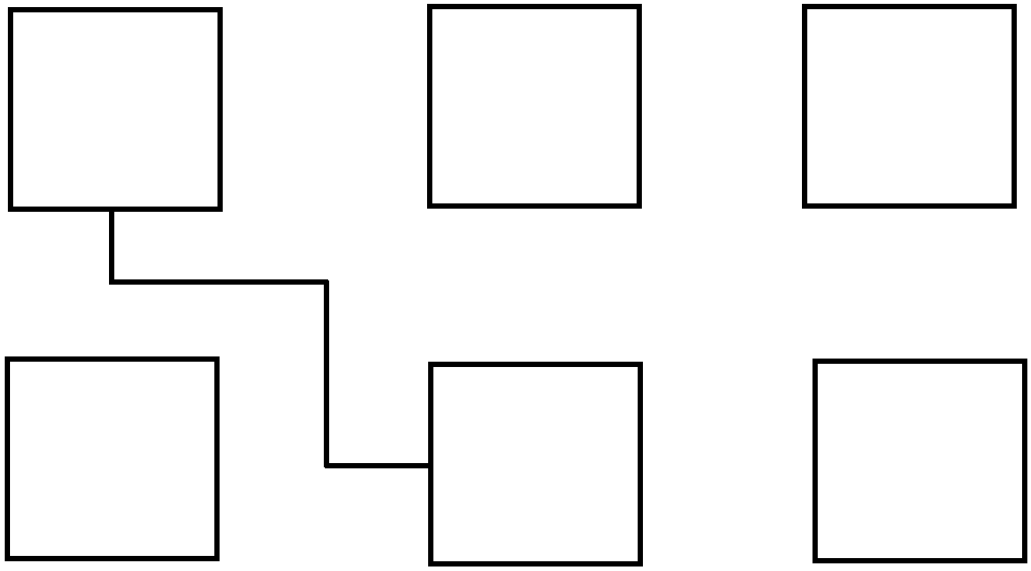
Slika 48: Tablice su i istom redu (Izvor: vlastita izrada)

- Tablice su u istom stupcu, tablice su jedna do druge ili tablice nisu jedna do druge.



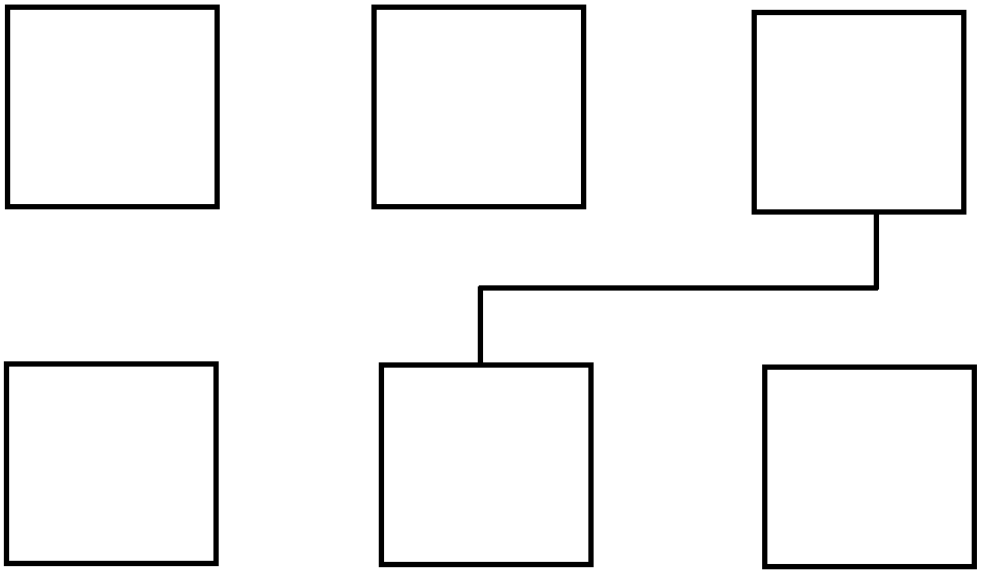
Slika 49: Tablice su u istom stupcu (Izvor: vlastita izrada)

- Tablica „*Dijete*“ je iznad lijevo



Slika 50: Tablica je iznad lijevo (Izvor: vlastita izrada)

- Tablica je iznad desno



Slika 51: Tablica je iznad desno (Izvor: vlastita izrada)

Da sada ne stavljamo još dva slučaja kada su tablice ispod lijevo i desno samo si zamislimo dva zadnja slučaja i preokrenemo im veze. Za biranje tih početnih točaka kreirana je metoda koja traži slobodno mjesto pomoću metode slučajnog odabira na ranije definiranoj strani tablice. Sve točke tablice je lako dobiti jer se znaju dimenzije tablica i početna točka prema kojoj su nacrtane. Da malo bolje pojasnimo metoda vraća početne točke tablice „*Roditelj*“ i tablice „*Dijete*“ te će se na tim mjestima iscrtati veza i taj dio tablice će biti označen kao zauzet od veze te se neće više moći spojiti neka druga veza na to mjesto te se time riješio problem preklapanja veza. Time su početne točke tablica nađene i spremljene u odgovarajuće varijable iz razloga jer su nam potrebne za idući korak.

U programskom alatu „*Microsoft Visual Studio*“ postoje predefinirane metode za crtanje grafike. Metoda koju sam je koristio je radila na sljedeći način. Trebalo je napraviti polje točaka koje će sadržavati „*x*“ i „*y*“ koordinate od ishodišta do odredišta i upotrijebiti te točke za izradu grafike. Sintaksa kreiranja takvog polja točaka je sljedeća:

```
Point[] pointsName= { new Point(x,y),
                      New Point(x1,y1),
};
Pen pen=new pen(Color,integer);
```

```
GraphicsPath pathName= new GraphicsPath();
pathName.AddLines(pointsName);
Graphics.DrawPath(pen,pathName);
```

Prema sintaksi ovaj dio nije prezahtjevan, ali postoji dio koji je zahtjevan, a to je samo dobivanje točaka. Podsjetimo se da svaki zapis tablice se nalazi u matrici tipa

`int matrix[panelHeight,PanelWidth];.`

U toj matrici zauzeta polja su označena brojem „1“, crte su označene brojem „2“ okvir matrice je označen brojem „3“. Algoritam koji sam izradio kako bi se našao put od jedne tablice do druge je više-manje jednostavan za razumjeti. Kao što sam rekao crtanje veza se događa slijedno tj. veze se crtaju tako kako dolaze tablice na provjeru. I podsjetimo se još da imamo sve potrebne podatke, točke spajanja u ovom slučaju, za crtanje.

Prvi korak je bio kreirati novi tip podatka `Position` koji će kroz konstruktor primati koordinate „*x*“ i „*y*“. Prvo kreiramo dvije instance te klase, a to su `startPosition` i `endPosition` i kao trenutnu vrijednost kreiramo instancu `currentPosition` koja će nam označavati pomak. Sada koristimo „while“ petlju koja se prekida u slučaju kad je `currentPosition` jednak `endPosition.`

```
            public Point [] DrawingConection()
            {
                  Position startPosition = new
Position(dockingStartPositionX, dockingStartPositionY);
                  Position endPosition = new Position(dockingEndPositionX,
dockingEndPositionY);

                  List<Point> pointList = new List<Point>();
                  Position currentPosition = startPosition;
                  while (true)
                  {
                        if (currentPosition.x == endPosition.x &&
currentPosition.y == endPosition.y) break;
```

Za pomak trenutne pozicije kreiramo listu vektora koja će se kretati u četiri smjera: gore, dolje, lijevo i desno. Te pomoću te liste provjeravamo koji od tih smjerova ja najbliže odredištu pomoću pitagorinog poučka.

```
            List<Position> directions = new List<Position>
                        {
                  new Position(currentPosition.x +1,currentPosition.y),
                  new Position(currentPosition.x -1,currentPosition.y),
                  new Position(currentPosition.x ,currentPosition.y+1),
                  new Position(currentPosition.x ,currentPosition.y-1)
                        };

                        double distance = Distance(currentPosition,
endPosition);
                        foreach (Position next in directions)
                        {
                              double newDistance = Distance(next,
endPosition);
```

```
                                    if (newDistance < distance)
                                          currentPosition = next;

private double Distance(Position pos1, Position pos2)
            {
                  return Math.Sqrt((pos1.x - pos2.x) * (pos1.x - pos2.x) +
(pos1.y - pos2.y) * (pos1.y - pos2.y));
            }
```

Prvotno je bilo implementirano da u slučaju kada dođe do tablice veza bi trebala skrenuti, ali u nekim slučajevima je baš ta pozicija bila najbliže odredištu i trenutna pozicija nikada nije bila jednaka završnoj te je petlja zapela na tome koraku i nikada nije bila izvršena pa je napravljeno da veze mogu ići preko tablica odnosno iza njih i svaka veza je svoje boje radi lakšeg raspoznavanja koja veza ide u kojem smjeru. I tako je nađen najbliži put prema odredištu i svaka takva točka je dodana u polje točaka koje su kasnije korištene za crtanje.

Napravljeno je pet tipova takvih točaka. To su:

- Točke iskorištene za grafiku startne tablice
- Točke iskorištene za grafiku tipa veze startne grafike
- Točke iskorištene za grafiku završne tablice
- Točke iskorištene za grafiku tipa veze završne tablice
- Točke iskorištene za grafiku veze od ishodišta do odredišta

```
Point[] pointsEndTable = canvas.MyDrawingPathEndTable(pointX, pointY,
width, height);
Point[] pointsStartTable =canvas.MyDrawingPathStartTable(newPointX,
newPointY, newWidth, newHeight);
Point[] startTabletypePoints =
canvas.TableConnectionStartTable(parentTableConnectionType);
Point[] endTableTypePoints =
canvas.TableConnectionsEndTable(childTableConnectionType,
parentTableConnectionType);
Point[] dokcingPoints = canvas. DrawingConection ();
```

Kako bi svaki aspekt crtanja veza bio u skladu sa ispravnim preporučenim za kreiranje grafike, napravljen je novi „*paintEvent*“ koji služio isključivo za takve slučajeve u kojem se i nalazi sav kod koji je zaslužen za izvedbu svih podataka koji su skupljeni ranije, a to su skup točaka za crtanje.

```
    private void panelERADijagrama_Paint(object sender, PaintEventArgs e)
        {
            try
            {
                int i = 0;
                    base.OnPaint(e);
                    //End table points
                    foreach (var endTablePoint in listOfEndPointsTable)
                    {
                    Pen pen = new Pen(listOfRandomColor.ElementAt(i), 1);
```

```
                    GraphicsPath endLine = new GraphicsPath();
                        endLine.AddLines(endTablePoint);
                        e.Graphics.DrawPath(pen, endLine);
                    i++;
                    }
                i = 0;
                //Start table points
                foreach (var startTablePoint in listOfStartTablePoints)
                    {

                    Pen pen = new Pen(listOfRandomColor.ElementAt(i), 1);
                    GraphicsPath startLine = new GraphicsPath();
                        startLine.AddLines(startTablePoint);
                        e.Graphics.DrawPath(pen, startLine);
                    i++;
                    }
                i = 0;
                    //Start table type
                    foreach (var startTableTypePoint in
listOfStartTableTypePoints)
                    {
                    Pen pen = new Pen(listOfRandomColor.ElementAt(i), 1);
                    GraphicsPath startTableType = new GraphicsPath();
                        startTableType.AddLines(startTableTypePoint);
                        e.Graphics.DrawPath(pen, startTableType);
                    i++;
                    }

                i = 0;
                    //End table type
                    foreach (var endTableTypePoints in
listOfEndTableTypePoints)
                    {
                    Pen pen = new Pen(listOfRandomColor.ElementAt(i), 1);

                    GraphicsPath endTableType = new GraphicsPath();
                        endTableType.AddLines(endTableTypePoints);
                        e.Graphics.DrawPath(pen, endTableType);
                    i++;
                    }
                i = 0;
                    //Docking lines
                    foreach (var dokcingPoints in listOfDockingLinesPoints)
                    {
                    Pen pen = new Pen(listOfRandomColor.ElementAt(i), 1);
                    GraphicsPath dockingLines = new GraphicsPath();
                        dockingLines.AddLines(dokcingPoints);
                        e.Graphics.DrawPath(pen, dockingLines);
                    i++;

            }
            catch {
                System.Console.WriteLine("Error");

            }
        }
```

Sa tim programskim kodom smo završili sa crtanjem veza i cjelokupno generiranje ERA dijagrama.

5. Zaključak

Izrada ovoga rada je bila puna nepredvidivih komplikacija i zahtijevala je vještinu snalažljivosti u novim problemima. Problema je bilo mnogo, ali su se rješavali jedan po jedan. Cilj ovog rada nije bio samo malo bolje se upoznati sa funkcionalnostima modeliranja baze podataka nego mnogo više od toga. Vještine koje su bile bitnije za ovaj zadatak je razmišljanje izvan okvira, snalažljivost, određena razinu znanja matematike, te samo poznavanje programskog jezika koji je izabran za implementaciju u ovom slučaju jezika C#. Vrlo mi je drago što sam uspio ispuniti ovaj zadatak, ali ne mogu reći da sam zadovoljan u potpunosti rezultatom jer zbog kompleksnosti zadatka izabran je lakši način implementacije.

Izradom rada dobio sam malo bolji uvid tj. podsjetio sam se kako točno funkcioniraju baze podataka i kako izgleda njihova implementacija. Ali više od toga sam stekao iskustva u programskom jeziku C# koji mi je do sada bio omiljeni jezik i drago mi je da sam stekao još više iskustva u njemu koje će mi nadam se poslužiti jednom u životu.

Tek sam sada svjestan nakon izrade ove teme zašto je ta tema bila dugo slobodna i zašto niti jedan student je nije htio izabrati, ali svejedno sam ispunio zadatak i mogu reći kako je riješen još jedan problem.

Izrađeni program bi se mogao primijeniti kod studenata ili bilo kojih drugih „učenika“ koji bi htjeli imati uvid u neki vizualni izgled baze podataka zbog njegove jednostavnosti za korištenje bez potrebe za preveliko predznanje koncepta baze podataka.

Popis literature

[1] Wikipedija, „*Sustav za upravljanje bazom podataka*“, 2020. [Na internetu]. Dostupno: https://hr.wikipedia.org/wiki/Sustav_za_upravljanje_bazom_podataka [pristupano 14.7.2020.].

[2] Loomen, „*Sustavi baza podataka*“, 2020. [Na internetu]. Dostupno: https://loomen.carnet.hr/pluginfile.php/295200/mod_resource/content/1/Sustavi%20b aza%20podataka.pdf [pristupano 14.7.2020.].

[3] Microsoft, „*Visual Studio*“, 2019. [Na internetu]. Dostupno: https://visualstudio.microsoft.com/vs/ [pristpano 14.7.2020.].

[4] Geek, „*Desktop aplikacija*“, 2018. [Na internetu]. Dostupno: https://geek.hr/pojmovnik/sto-je-desktop-aplikacija/ [pristupano 14.7.2020.].

[5] Wikipedija, „*Objektno orijentirano programiranje*“, 2020. [Na internetu]. Dostupno: https://hr.wikipedia.org/wiki/Objektno_orijentirano_programiranje [pristupano 14.7.2020.].

[6] Wikipedija, „*Windows Forms*“, 2020. [Na internetu]. Dostupno: https://en.wikipedia.org/wiki/Windows_Forms [pristupano 14.7.2020.].

[7] Wikipedija, „*C#*“, 2020. [Na internetu]. Dostupno: https://bs.wikipedia.org/wiki/C%E2%99%AF [pristupano 14.7.2020.].

[8] M. Maleković i K. Rabuzin, *Uvod u baze podataka,* Varaždin: Fakultet organizacije i informatike. 2016

[9] K. Rabuzin, *Uvod u SQL* : Varaždin, Fakultet organizacije i informatike. 2016

[10] K. Rabuzin, *SQL: napredne teme:* Varaždin, Fakultet organizacije i informatike. 2016

Popis slika