**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Vedran Novak**

# Baza za Hotel napisana u jeziku SQL Server

**PROJEKT IZ KOLEGIJA BAZA PODATAKA 2**

**Varaždin, 2019.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Vedran Novak**

**Matični broj: 43131/14–R**

**Studij: Informacijski sustavi**

**(*ili Poslovni sustavi, Ekonomika poduzetništva, Primjena informacijske tehnologije u poslovanju, Informacijsko i programsko inženjerstvo, Baze podataka i baze znanja, Organizacija poslovnih sustava, Informatika u obrazovanju*)**

# Baza za Hotel napisana u jeziku SQL Server

**PROJEKT IZ KOLEGIJA BAZA PODATAKA 2**

**Mentor/Mentorica:**

Martina Šestak, mag. Inf.

**Varaždin, siječanj 2019.**

*Vedran Novak*

**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**

U ovom seminarskom radu nalazi se sva dokumentacija vezana za napravljeni projekt iz Baze Podataka 2, na temu Hotela. To uključuje opis korištene aplikacijske domene (Visual Studio 2017), prikaz i izrada ERA modela na kojem je i glavni fokus ovog projekta. Opis korištenog alata za modeliranje i administraciju baze podataka, te opcije korištene pri izradi navedene baze. Opis korištenih tehnologija i nakraju, sam prikaz funkcionalnosti razvijenog sučelja.

**Ključne riječi:** Tablica; Triggeri; Složeni, Jednostavni, Upiti, ERA model, Veze;

# Sadržaj

# 1. Uvod

Seminarski rad na temu izrade baze podataka za hotel započet ćemo sa objašnjavanjem sustava za upravljanje bazom podataka kojom ćemo se koristiti kod izrade aplikacijske domene, a to je SQL Server. Što je SQL Server?

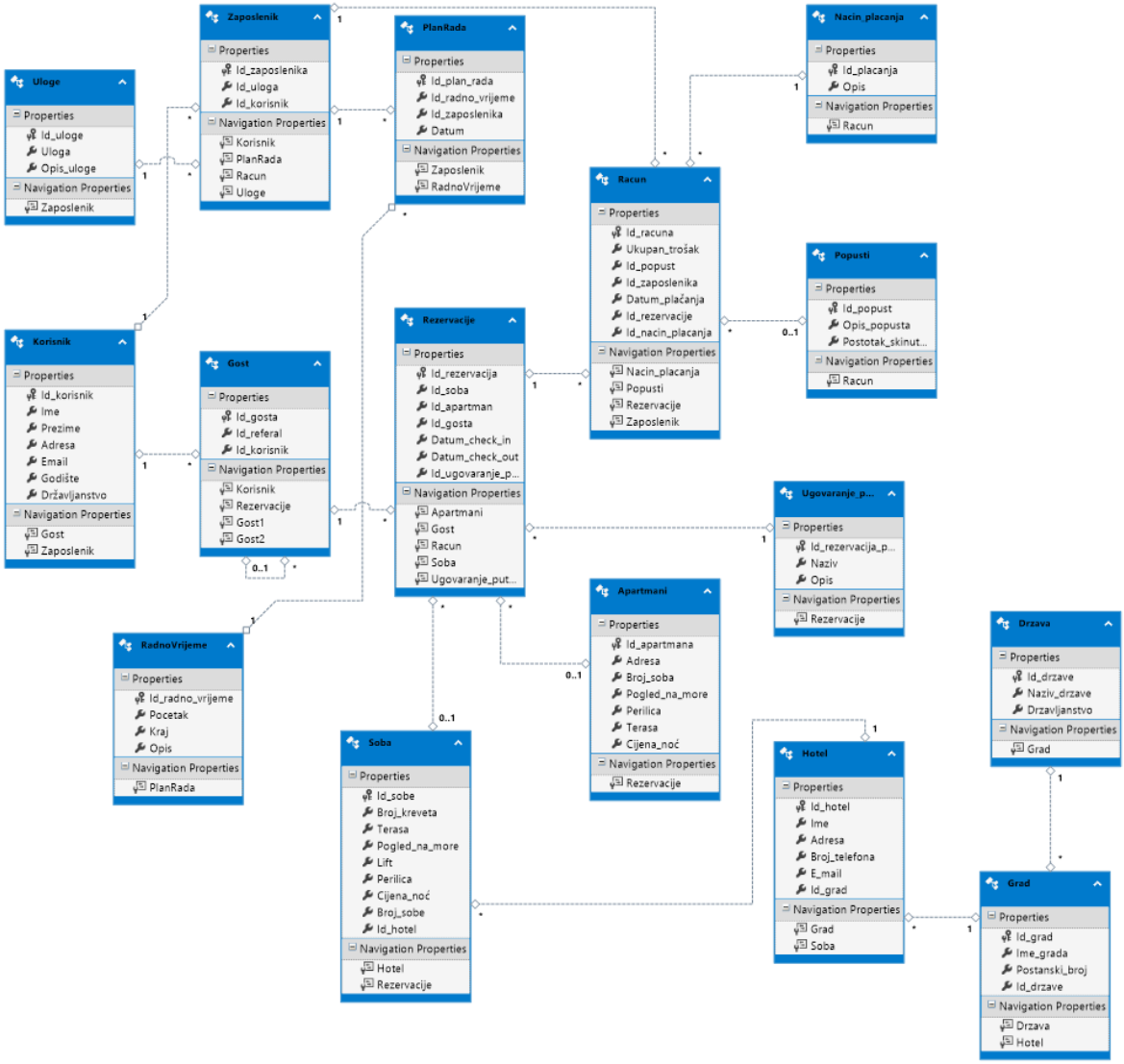
SQL Server razvio je tehnološki div Microsoft Co. Osim, kao što smo već rekli da je to sustav za upravljanjem relacijskim bazama podataka trebamo objasniti i njegovu glavnu funkciju. SQL Server je softverski proizvod čija je glavna funkcija pohranjivanje i dohvaćanje podataka prema potrebama drugih softverskih aplikacija koje se mogu izvoditi na istom ili na drugom računalu koji je na primjer povezan mrežnim putem na prvo računalo. Softverska aplikacija kojom ćemo se mi koristiti zove se „Hotel“. Njena glavna namjena bit će obavljanje samih rezervacija gostiju koji žele doći i ispisivanje računa, koristeći kreiranu bazu podataka, koja će uvelike olakšati upravljanjem glavnih zadaća hotela.

# 2. Opis aplikacijske domene

Naziv aplikacijske domene kao što smo već rekli u uvodu, jest „Hotel“. Izrada baze hotela u koju će se moći upisati gost koji dolazi gostovati u naš hotel. Zaposlenike koji ga ugošćuju. Sam hotel i njegov naziv, adresu i opis. Sve njegove sobe, i gdje s nalaze, šta sve imaju u sebi i koliko dođe noćenje u njima. Također i apartmane, pošto sam uzeo kao primjer hotel u kojem sam sezonski radio koje se nalazilo u samoj staroj jezgri, te pošto se nije smjelo ništa novog renovirati niti graditi, hotel je bio prisiljen proširiti se na način da počne otkupljivati okolne kuće i stanove. Zbog čega puno soba koje hotel ima zapravo nisu dio samog hotela, tj. Glavne zgrade u kojoj se hotel radi, već se nalaze puno dalje, stoga smo ih zvali apartmanima, iako nisu baš bili pošto su i dalje, recimo, bili u sklopu hotela. I dalje, su se sve glavne zadaće obavljale u zgradi hotela, poput check-ina, check-outa, plaćanje računa, rezerviranja apartmana ili soba, održavanje doručka, ručka, odvođenje ili dovođenje gosta s parkirališta do hotela itd. Dalje, kao što i sami znamo, svako poduzeće pa tako i hotel mora imati svoje zaposlenike, bez kojih inače ne bi niti radilo. Također, mora se brinuti o svojim zaposlenicima na način da svaki ima predodređenu ulogu s radnim vremenom i određenim planom rada kad će i koliko raditi kojeg datuma, jer bi inače vladao anarhizam i svi bi radili šta bi htjeli, tj. Poduzeće ne bi moglo pravilno funkcionirati. Ovisno o ulozi zaposlenika, račun će se moći ispisati gostu. Zbog čega ga na primjer zaposlenik koji obavlja ulogu čistaća neće moći ispisati račun gostu, dok recepcioner će moći. Također u našoj bazi će se provjeravati i zabilježavati koji gost je kojem gostu preporučio naš hotel. Te ovisno o tome ili načinu plaćanja, izdati određeni popust na sveukupni trošak gostovanja u našem hotelu. Te pošto je preko preporuke gosta jedan od načina na koji gost može doći do rezervacije sobe ili apartmana u naš hotel, moramo na svaku rezervaciju napisati način na koji je netko rezervirao sobu. Sada ćemo navedenu aplikacijsku domenu prikazati i na izrađenoj bazi podataka, u sljedećim poglavljima

# 3. ERA model

Započnimo prvo sa time što je to ERA model? To je zapravo model entiteta (engl. Entity relationship attribute model) ili skraćeno ERA model. Naš ERA model za bazu koju koristimo u našoj aplikaciji „Hotel“ izgleda ovako:




Slika 1. ERA model baze naše podataka

Sad ćemo objasniti zašto naš ERA model izgleda ovako, i zbog čega smo se odlučili i kako povezati međusobno tablice.

Započet ćemo sa prvom tablicom, a to su 'Korisnik'. Tablica 'Korisnik' u sebi sadrži atribute poput imena osobe, prezimena, adrese stanovanja i Id_državljanstvo koje izvlači iz tablice 'Država', što ćemo malo kasnije spomenuti. Kod izgleda ovako:

```
CREATE TABLE [dbo].[Korisnik] (
    [Id_korisnik]      INT        IDENTITY (1, 1) NOT NULL,
    [Ime]              NCHAR (20) NOT NULL,
    [Prezime]          NCHAR (30) NOT NULL,
    [Adresa]           NCHAR (40) NOT NULL,
    [Email]            NCHAR (40) NULL,
    [Godište]          DATE       NOT NULL,
    [Id_drzavljanstvo] INT        NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_korisnik] ASC),
    CONSTRAINT [FK_Korisnik_ToTable] FOREIGN KEY ([Id_drzavljanstvo]) REFERENCES
[dbo].[Drzava] ([Id_drzave])
);
```

Kao što možemo vidjeti, primarni ključ se nalazi na polju **Id_korisnik**, dok imamo i vanjski ključ koji se nalazi na **Id_drzavljanstvo** koje se referencira na polje **Id_drzave** iz tablice 'Drzava'.

Sljedeće imamo tablicu 'Zaposlenici'. Zaposlenici su glavni razlog zašto hotel uopće posluje. Naime, bez zaposlenika hotel logički ne bi mogao funkcionirati, barem zasada, dok ne dođemo do dovoljne razine automatizacije. Do onda, trebaju nam naši zaposlenici, zbog čega smo im i posvetili cijelu jednu tablicu. Prije nego odemo na sam izgled tablice, moramo opisati čemu služi, kako bi malo olakšali kasnije objašnjavanje. Kao što znamo, u svakoj organizaciji svaki zaposlenik ima svoju predodređenu ulogu koju mora obavljati. Također svaki zaposlenik ima svoje radno vrijeme u kojem obavlja svoju ulogu. Te svaki zaposlenik ima i poneke atribute po kojima se također razlikuje od drugog zaposlenika, kao što su npr. OIB. ime, prezime, email, adresa itd. Naime, da još pojasnimo jednu stvar prije nego pokažemo svoju tablicu za zaposlenika. Dvije osobe mogu imati isto ime i/ili prezime, također mogu živjeti na istoj adresi, ili koristiti iz nekog x-y razloga isti Email. Ali dvije osobe ne mogu biti iste, zbog čega moraju imati bare jedan jedinstveni atribut po čemu se moraju razlikovati. U ovom slučaju to bi bio OIB. Da bi imali isti OIB onda one bi i bile ista osoba. Zbog čega, naš zaposlenik mora imati barem jedno polje koji mora biti jedinstven. U bazi podataka u ovom slučaju atributi iz ovoh konteksta u kontekstu baze podataka bi sačinjavala polja ili stupce, dok cijeli zapis o zaposleniku bi činio slog. Sada možemo i vidjeti kako bi naša tablica 'Zaposlenik' u našem kodu trebala izgledati:

```
CREATE TABLE [dbo].[Zaposlenik] (
    [Id_zaposlenika] INT IDENTITY (1, 1) NOT NULL,
    [Id_uloga]       INT NOT NULL,
    [Id_korisnik]    INT NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_zaposlenika] ASC),
    CONSTRAINT [ulogaFK] FOREIGN KEY ([Id_uloga]) REFERENCES [dbo].[Uloge]
([Id_uloge]),
    CONSTRAINT [zaposlenik_korisnikFK] FOREIGN KEY ([Id_korisnik]) REFERENCES
[dbo].[Korisnik] ([Id_korisnik])
);
```

Kao što vidimo, naše jedinstveno polje u ovom slučaju čini stupac „Id_zaposlenika“, te također čini i primarni ključ (engl. Primary key). Što znači da ćemo njega koristiti kako bi se povezivali na druge tablice, kao oslonac da on označava tog određenog zaposlenika sa tim jedinstvenim

poljem. Osim što je jedinstven i mora biti upisan (NOT NULL), polje može biti tipa podataka INT zbog čega može primiti samo brojčanu vrijednost (integer), koji počinje od 1 i automatski se povećava za 1 zbog IDENTITY (1, 1). U tablici osim jedinstvenog polja koji smo trenutno objasnili možemo vidjeti i ostala polja koji „opisuju“ našeg zaposlenika. To su da zaposlenik mora imati **Ime** koje može imati samo maksimalno 10 karaktera (NCHAR (10)) i zbog uvjeta NOT NULL (stupac u tablici koji MORA imati uneseni podatak), mora imati **Prezime**, mora imati **Email**, mora imati neku ulogu (**Id_uloga**), te može imati neku **Adresu** zbog uvjeta NULL (stupac koji može a i ne mora imati uneseni podatak). Također u tablici možemo vidjeti da se susrećemo sa još jednim novim pojmom a to jest, da imamo i sekundarni ključ (engl. Foreign key) koji se nalazi na **Id_uloga**, a referencira na polje „Id_uloge“ iz tablice „Uloge“. Ako pogledamo li u tablicu „**Uloge**“ možemo doći i do logičkog objašnjenja te poveznice. Sada se moramo vratiti na dio koji smo preskočili. Kao što možemo primijetiti da u našoj tablici nigdje se ne nalaze upravo navedena polja, poput **Ime, Prezime, Adresa**… To je zato što imamo drugu tablicu 'Korisnik' na koju se referenciraju, i iz koje izvlače navedene podatke. Vanjskim ključem, koji smo upravo objasnili, tj. polje **Id_korisnik** referencira na **Id_korisnik** koji se nalazi u tablici 'Korinisk'.

Tablica 'Uloge' koju možemo vidjeti na slici 1. izgleda ovako:

```
CREATE TABLE [dbo].[Uloge] (
    [Id_uloge]   INT          IDENTITY (1, 1) NOT NULL,
    [Uloga]      NCHAR (30)   NOT NULL,
    [Opis uloge] NCHAR (1000) NULL,
    PRIMARY KEY CLUSTERED ([Id_uloge] ASC)
);
```

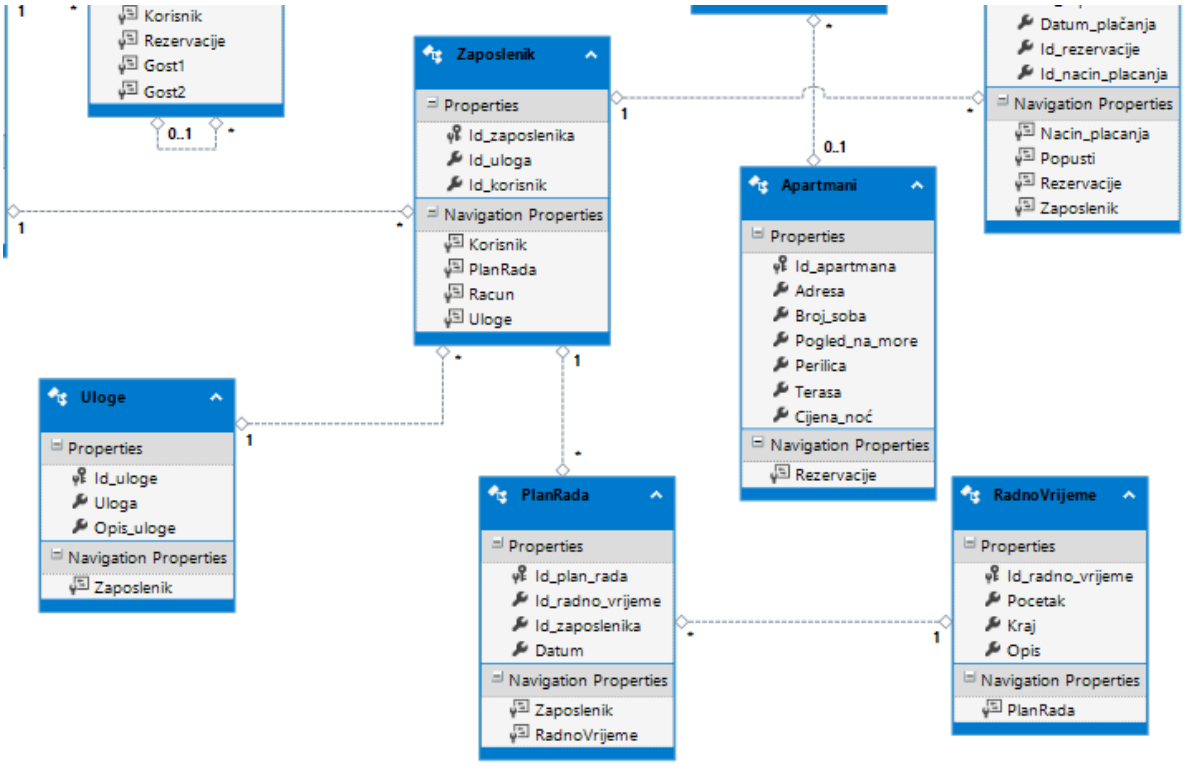
Možemo odmah zaključiti da je tablica jedinstvena, pošto ne vidimo da ima ikakvih sekundarnih ključeva. Te dolazimo odmah do logičkog zaključka da povezanost između tablica 'Zaposlenik' i 'Uloga' glasi ovako: Zaposlenik može imati jednu ulogu, dok uloge mogu imati više zaposlenika. Tj. Jedan zaposlenik kao što smo rekli može obavljati samo jednu ulogu, što znači da ne može izvršavati više poslova odjednom, dok više zaposlenika može dijeliti iste uloge, tj. Obavljati iste poslove. Isto tako, zaposlenik može imati povezane atribute od samo jednog korisnika, dok više korisnika mogu biti zaposlenici našeg hotela.

Dalje, kao što smo rekli, zaposlenik sa svojom ulogom mora imati određeni plan rada otkad do kad će raditi. Zbog čega smo napravili tablice 'PlanRada' i 'RadnoVrijeme':

```
CREATE TABLE [dbo].[RadnoVrijeme] (
    [Id_radno_vrijeme] INT        IDENTITY (1, 1) NOT NULL,
    [Pocetak]          TIME (7)   NOT NULL,
    [Kraj]             TIME (7)   NOT NULL,
    [Opis]             NCHAR (20) NULL,
    PRIMARY KEY CLUSTERED ([Id_radno_vrijeme] ASC)
);
```

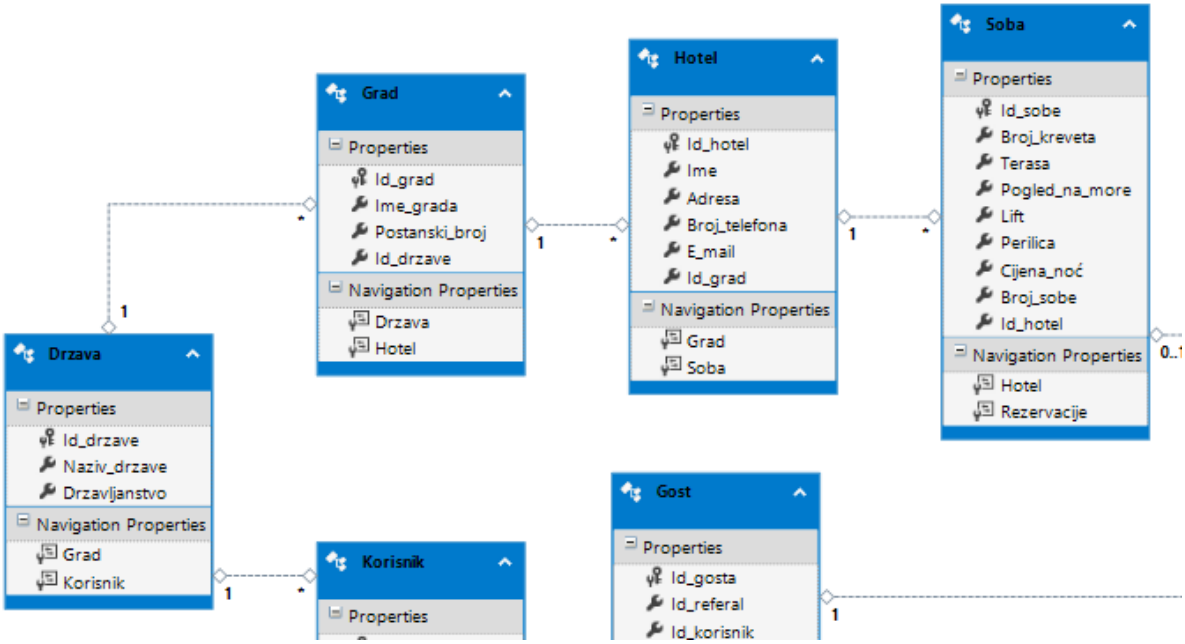
```
CREATE TABLE [dbo].[PlanRada] (
    [Id_plan_rada]     INT  IDENTITY (1, 1) NOT NULL,
    [Id_radno_vrijeme] INT  NOT NULL,
    [Id_zaposlenika]   INT  NOT NULL,
    [Datum]            DATE NOT NULL,
    CONSTRAINT [PK_PlanRada] PRIMARY KEY CLUSTERED ([Id_plan_rada] ASC),
    CONSTRAINT [FK_PlanRada_ToTable_1] FOREIGN KEY ([Id_radno_vrijeme]) REFERENCES
[dbo].[RadnoVrijeme] ([Id_radno_vrijeme]),
    CONSTRAINT [FK_PlanRada_ToTable] FOREIGN KEY ([Id_zaposlenika]) REFERENCES
[dbo].[Zaposlenik] ([Id_zaposlenika])
);
```

Vidimo da tablica 'RadnoVrijeme' je dosta jednostavna, pošto ne uključuje nikakve sekundarne ključeve, stoga se u nju samo upisuju podaci. Poput početka i kraja radnog vremena u obliku sati i minuta zbog uvjeta TIME. Naime, time pokušavamo reći da određeni zaposlenik uvijek ima određeno radno vrijeme, tj. Smjenu npr. Jutarnju, popodnevnu ili večernju u kojoj obavlja svoji posao. Tada npr. Ako bi htjeli napisati da postoji jutarnja smjena onda bi polje **Pocetak** morao poprimiti vrijednost '08:00:00', a **Kraj** '16:00:00', dok polje **Opis** može poprimiti vrijednost „Jutarnja smjena“. Time smo predodredili smjene u kojima zaposlenici obavljaju svoje dužnosti za hotel, dok tablica 'PlanRada' izvršava za razliku 'RadnoVrijeme' puno težu zadaću. Naime, tablica 'PlanRada' kao što već po slici 1. možemo ili kodu možemo primijetiti da je složena tablica, te da ima dva sekundarna ključa. Naime, njezina dva polja se referenciraju na druga polja drugih tablica radi izvršavanje određene zadače. Ako pogledamo u zadnji kod, vidimo **Id_plan_rada** da polje **Id_zaposlenika** referencira se na polje istog naziva **Id_zaposlenika** koji spada u drugu tablicu, tablicu 'Zaposlenik'. Dok **Id_radno_vrijeme** se referencira na tablicu 'RadnoVrijeme' i njezino polje **Id_radno_vrijeme**. Što ta logička veza znači između te tri tablice?




Slika 2. ERA model, veza između tri tablice

Znači da jedan zaposlenika mogu imati više planova rada, dok samo jedan plan rada može imati jednog zaposlenika. Te više plan rada može imati samo jednu smjena, tj. radno vrijeme kako smo nazvali, dok jedan plan rada može imati samo jedno radno vrijeme.  Što ukratko znači da pojedini zaposlenik, ovisno o datumu, može raditi drugačiju smijenu.




Slika 3. ERA model veze između 'Hotel', 'Soba'

Pogledamo li u sliku 3, prvo ćemo obratiti naš pogleda na tablicu 'Grad'. Grad ime atribute poput **Id_grada, Ime_grada**, **poštanski broj** i **Id_drzave**. Dok se **Id_drzave** referencira na tablicu 'Drzava'. Odnos između tablica 'Drzava' i 'Grad' jest sljedeći. Jedna država može imati više gradova, dok jedan grad može pripadati samo jednoj državi. Te isto vrijedi i za hotel, što ćemo i kasnije vidjeti.

```
CREATE TABLE [dbo].[Grad] (
    [Id_grad]        INT        IDENTITY (1, 1) NOT NULL,
    [Ime_grada]      NCHAR (30) NOT NULL,
    [Postanski_broj] INT        NULL,
    [Id_drzave]      INT        NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_grad] ASC),
    CONSTRAINT [FK_Grad_ToTable] FOREIGN KEY ([Id_drzave]) REFERENCES [dbo].[Drzava]
([Id_drzave])
);
```

Sada možemo pogledati u tablicu 'Hotel'. Kao što znamo, hotel mora imati svoje **Ime**, **Adresu** po kojoj ćemo ga naći. **Broj_telefona** kao kontakt broj, **E_mail** kao također jedan od načina kontaktiranja, te zadnje polje **Id_grad** koje se referencira na tablicu 'grad'. Također, može imati još polja ali mi smo naveli samo one koje smatramo najvažnijima. Hotel je složena tablica što možemo zaključiti i preko same slike 3., ali i preko sljedećeg koda:

```
CREATE TABLE [dbo].[Hotel] (
    [Id_hotel]      INT        NOT NULL,
    [Ime]           NCHAR (20) NOT NULL,
    [Adresa]        NCHAR (30) NOT NULL,
    [Broj telefona] NCHAR (20) NOT NULL,
    [E-mail]        NCHAR (20) NOT NULL,
    [Id_grad]       INT        NOT NULL,
```

```
    PRIMARY KEY CLUSTERED ([Id_hotel] ASC),
    CONSTRAINT [CK_Hotel_Email] UNIQUE NONCLUSTERED ([E-mail] ASC),
    CONSTRAINT [FK_Hotel_ToTable] FOREIGN KEY ([Id_grad]) REFERENCES [dbo].[Grad]
([Id_grad])
);
```

Jedan hotel se može nalaziti u više gradova, dok jedan grad može imati više hotela.

Tablici 'Soba' je malo složenija u odnosu na tablicu hotel, pošto nema samo primarni ključ, već i sekundarni ali i par novih pojmova s kojima se prvi put susrećemo.

```
CREATE TABLE [dbo].[Soba] (
    [Id_sobe]        INT       IDENTITY (1, 1) NOT NULL,
    [Broj kreveta]   SMALLINT  NOT NULL,
    [Terasa]         NCHAR (2) NOT NULL,
    [Pogled na more] NCHAR (2) NULL,
    [Lift]           NCHAR (2) NOT NULL,
    [Perilica]       NCHAR (2) NOT NULL,
    [Cijena/noć]     MONEY     NOT NULL,
    [Broj_sobe]      SMALLINT  NOT NULL,
    [Id_hotel]       INT       NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_sobe] ASC),
    CONSTRAINT [FK_Soba_Hotel] FOREIGN KEY ([Id_hotel]) REFERENCES [dbo].[Hotel]
([Id_hotel]),
    CONSTRAINT [CK_Soba_Broj_sobe] CHECK ([Broj_sobe]>=(10) AND [Broj_sobe]<=(50)),
    CONSTRAINT [CK_Soba_Lift] CHECK ([Lift]='Da' OR [Lift]='Ne'),
    CONSTRAINT [CK_Soba_Perilica] CHECK ([Perilica]='Da' OR [Perilica]='Ne'),
    CONSTRAINT [CK_Soba_Pogled_na_more] CHECK ([Pogled na more]='Da' OR [Pogled na
more]='Ne'),
    CONSTRAINT [CK_Soba_Terasa] CHECK ([Terasa]='Da' OR [Terasa]='Ne')
);
```

'Soba' se preko polja **Id_hotel** referencira na polje **Id_hotel** u tablici 'Hotel'. Što logički znači da: Jedan Hotel može imati više soba, dok jedna soba može pripadati samo jednom hotelu. Pogledajmo sliku 3. radi boljeg razumijevanja ovog izraza!

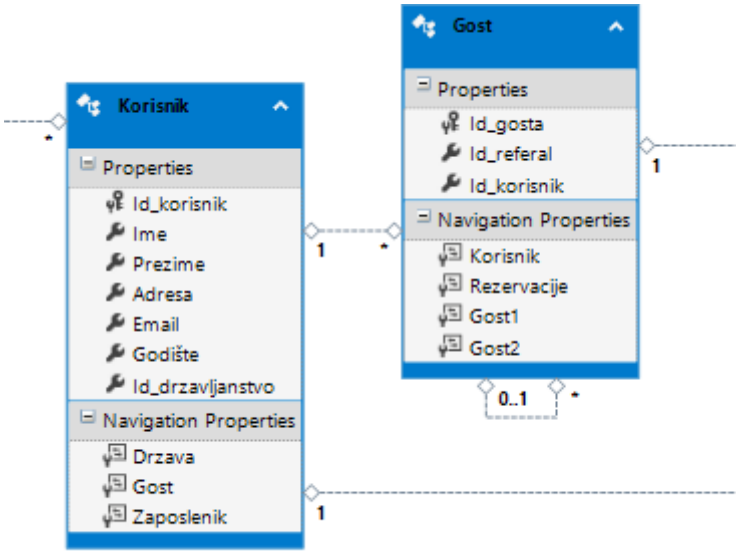
Nadalje, kao što smo rekli, prvi put se susrećemo sa novim pojmovima, kao što vidimo u tablici 'Soba' imamo jako puno CONSTRAINT CHECK-ova. Oni služe za provjeravanje dali vrijednost upisana u tablicu odgovara našim predodređenim uvjetima. Sa uvjetom CHECK možemo lakše izbjeći greške pri upisivanju podataka u tablici radi boljeg ostvarenja funkcionalnosti baze kojeu želimo postići. Na primjer.

```
CONSTRAINT [CK_Soba_Lift] CHECK ([Lift]='Da' OR [Lift]='Ne'),
```

Postavlja CONSTRAINT naziva **CK_Soba_Lift** koji provjerava dali je u polju **Lift** iz tablice 'Soba' zapisano „Da“ ili „Ne“. U slučaju da nije zapisano ništa od navedenog, izbacuje error 16, 1. Isti postupak provjeravanja vrijedi i za sljedeće CONSTRAINT-ove.

```
CONSTRAINT [CK_Soba_Broj_sobe] CHECK ([Broj_sobe]>=(10) AND [Broj_sobe]<=(50)),
CONSTRAINT [CK_Soba_Lift] CHECK ([Lift]='Da' OR [Lift]='Ne'),
CONSTRAINT [CK_Soba_Perilica] CHECK ([Perilica]='Da' OR [Perilica]='Ne'),
CONSTRAINT [CK_Soba_Pogled_na_more] CHECK ([Pogled na more]='Da' OR [Pogled na
more]='Ne'),
CONSTRAINT [CK_Soba_Terasa] CHECK ([Terasa]='Da' OR [Terasa]='Ne')
```

Dalje, susrećemo se sa novom unarnom vezom u ERA modelu kao što možemo vidjeti na sljedećoj slici.



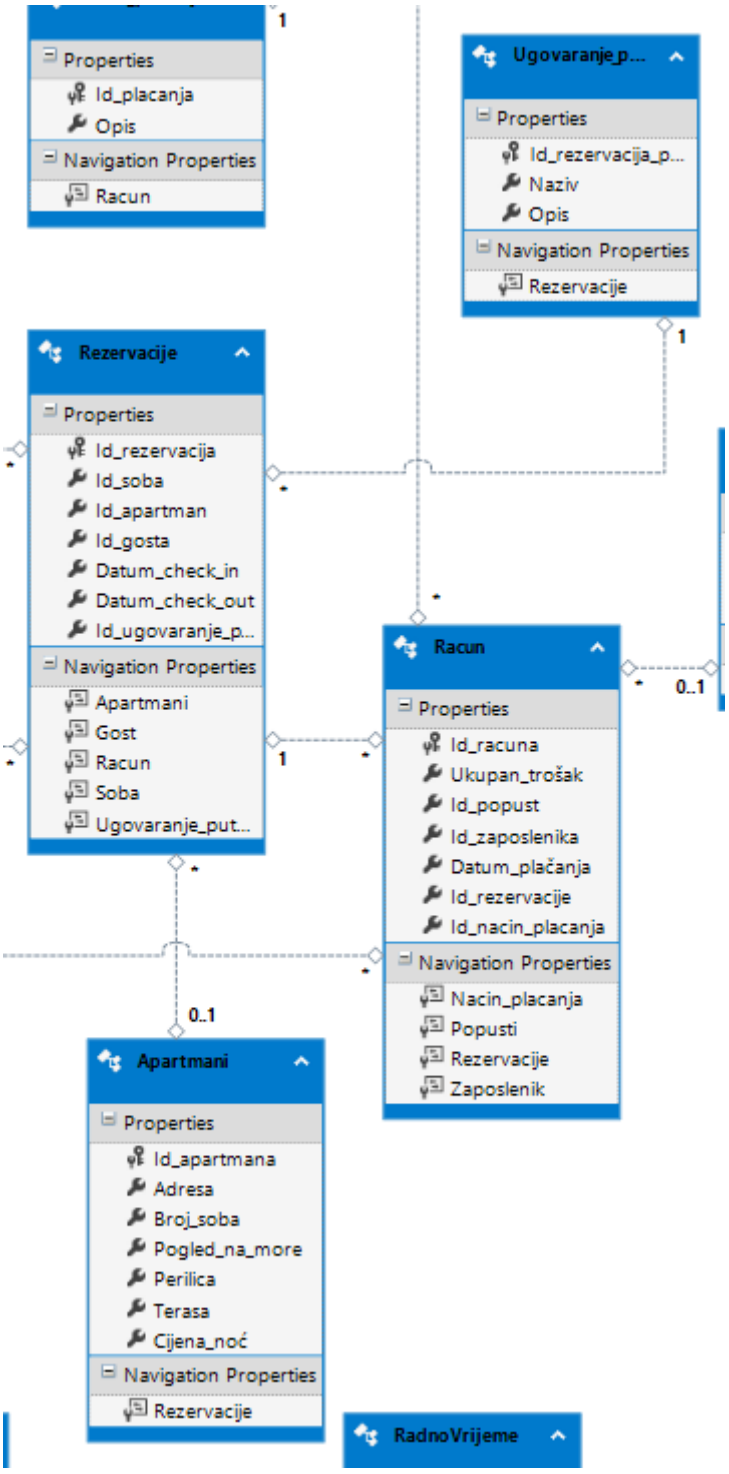

Slika 4. ERA model unarna veza

Kao što vrijedi za vezu 'Korisnik' – 'Zaposlenik'. 'Gost' se referencira na tablicu 'Korinsik' te iz nje izvlači razne podatke. Logička veza jest sljedeća: Gost može imati samo jednog korisnika, tj ime, prezime, adresu, itd. Dok više korisnika mogu biti gosti. Unarna veza označava da samo jedan zapis iz tablice može biti povezan samo s jednim zapisom iz te tablice. Što je nama u ovom slučaju bilo od koristi kod kreiranja polja **Id_referal** u našoj tablici 'Gost' koja kao što samo njezino ime kaže, treba imati rekurzivnu funkcionalnost da gost koji je već bio gostovao u našem hotelu može preporučiti drugom gostu naš Hotel. Zbog unarne veze kao što smo napisali ranije, i pošto tablica mora imati rekurzivnu funkcionalnost naš kod mora izgledati ovako:

```
CREATE TABLE [dbo].[Gost] (
    [Id_gosta]    INT NOT NULL,
    [Id_referal]  INT NULL,
    [Id_korisnik] INT NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_gosta] ASC),
    CONSTRAINT [gost_korisnikFK] FOREIGN KEY ([Id_korisnik]) REFERENCES
[dbo].[Korisnik] ([Id_korisnik]),
    CONSTRAINT [referalFK] FOREIGN KEY ([Id_referal]) REFERENCES [dbo].[Gost]
([Id_gosta])
);
GO
CREATE TRIGGER [dbo].[Trigger_Gost_Postoji_Referal]
    ON [dbo].[Gost]
    AFTER INSERT
    AS
    BEGIN
              PRINT 'Preporucio ga je gost: '
              SELECT k.Ime, k.Prezime FROM Gost g
              JOIN   Korisnik k
              ON            k.Id_korisnik = g.Id_korisnik
              WHERE  EXISTS ( SELECT 1 FROM inserted i WHERE i.Id_referal =
g.Id_gosta);
```

END Trenutno ćemo ignorirati naredbu CREATE TRIGGER, pošto ćemo se kasnije vratiti da ju objasnimo, i trenutno se samo fokusirati na samu tablicu 'Gost'. Pogledamo li u sekundarni ključ, možemo vidjeti da polje **Id_referal** se referencira na samog sebe, tj. Na polje **Id_gost** u tablicu 'Gost'. Logički, kao što smo već rekli, to znači da zapis iz polja **Id_referal** može biti povezan samo s jednim zapisom iz te tablice, u ovom slučaju to je zapis **Id_gost**. Znači, Svaki gost može preporučiti drugom gostu naš hotel, te više gosta mogu imati preporuku od istog gosta.

Dalje, da pojasnim, zašto na sljedećoj slici ćete vidjeti tablicu 'Apartman', iako pišemo bazu za hotel. Naime, u hotelu za koji sam radio prošle sezone, nalazio se u staroj jezgri starog grada, koji ima jako ograničenu površinu, te pošto je zabranjeno izgraditi ili renovirati zgrade u tom dijelu grada, hotel se odlučio na opciju proširivanja na način da otkupi susjedne kuće. I na taj način se uspio proširiti po cijelom starom gradu, zbog čega su mnoge sobe ostale nepovezane od glavne zgrade hotela iako se plaćanje, check-in, check-out, rezervacije i ostale zadaće stvari ostale izvršavati u glavnoj zgradi hotela. Stoga, sobe koje se ne nalaze u hotelu nazivamo apartmanima! Na slici 1 možemo vidjeti kako je to povezano, iako ćemo malo bolje još objasniti kasnije.




Slika 5. Tablice 'Apartmani' i 'Ugovaranje_putem'

Obje tablice spadaju u jednostavnije, pošto nemaju sekundarne ključeve, tj. Ne referenciraju se na druge tablice. Stoga ćemo samo preletjeti kroz njih. Kao što smo maloprije objasnili zašto uopće imamo tablicu 'Apartmani', tablica 'Ugovaranje_putem' služi radi zapisivanja **Naziva** i **Opis** načina na koji su gosti ugovorili svoju rezervaciju za taj hotel. Kodovi izgledaju sljedeće:

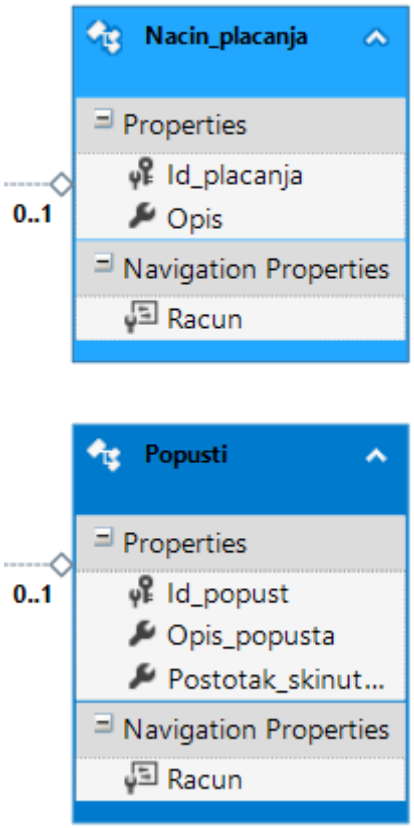
Tablica 'Apartmani' u kojoj ćemo gledati samo tablicu pošto objašnjavamo samo ERA model, trigger ćemo zasada ignorirati;

```
CREATE TABLE [dbo].[Apartmani] (
    [Id_apartmana]   INT        IDENTITY (1, 1) NOT NULL,
    [Adresa]         NCHAR (50) NOT NULL,
    [Broj soba]      INT        NOT NULL,
    [Pogled na more] NCHAR (2)  NOT NULL,
    [Perilica]       NCHAR (2)  NOT NULL,
    [Terasa]         NCHAR (2)  NOT NULL,
    [Cijena/noć]     MONEY      NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_apartmana] ASC),
    CONSTRAINT [CK_Apartmani_Perilica] CHECK ([Perilica]='Da' OR [Perilica]='Ne'),
    CONSTRAINT [CK_Apartmani_terasa] CHECK ([Terasa]='Da' OR [Terasa]='Ne'),
    CONSTRAINT [CK_Apartmani_Pogled_na_more] CHECK ([Pogled na more]='Da' OR [Pogled
na more]='Ne')
);

GO

CREATE TRIGGER [dbo].[Trigger_Apartmani]
   ON [dbo].[Apartmani]
   FOR INSERT
   AS
   BEGIN
       SELECT * FROM inserted;
   END
```

I tablica 'Ugovaranje_putem'_

```
 CREATE TABLE [dbo].[Ugovaranje_putem] (
    [Id_rezervacija_putem] INT         IDENTITY (1, 1) NOT NULL,
    [Naziv]                NCHAR (20)  NOT NULL,
    [Opis]                 NCHAR (100) NULL,
    PRIMARY KEY CLUSTERED ([Id_rezervacija_putem] ASC)
);
```

Sljedeće, imamo također dvije jednostavne koje možemo vidjeti na sljedećoj slici:

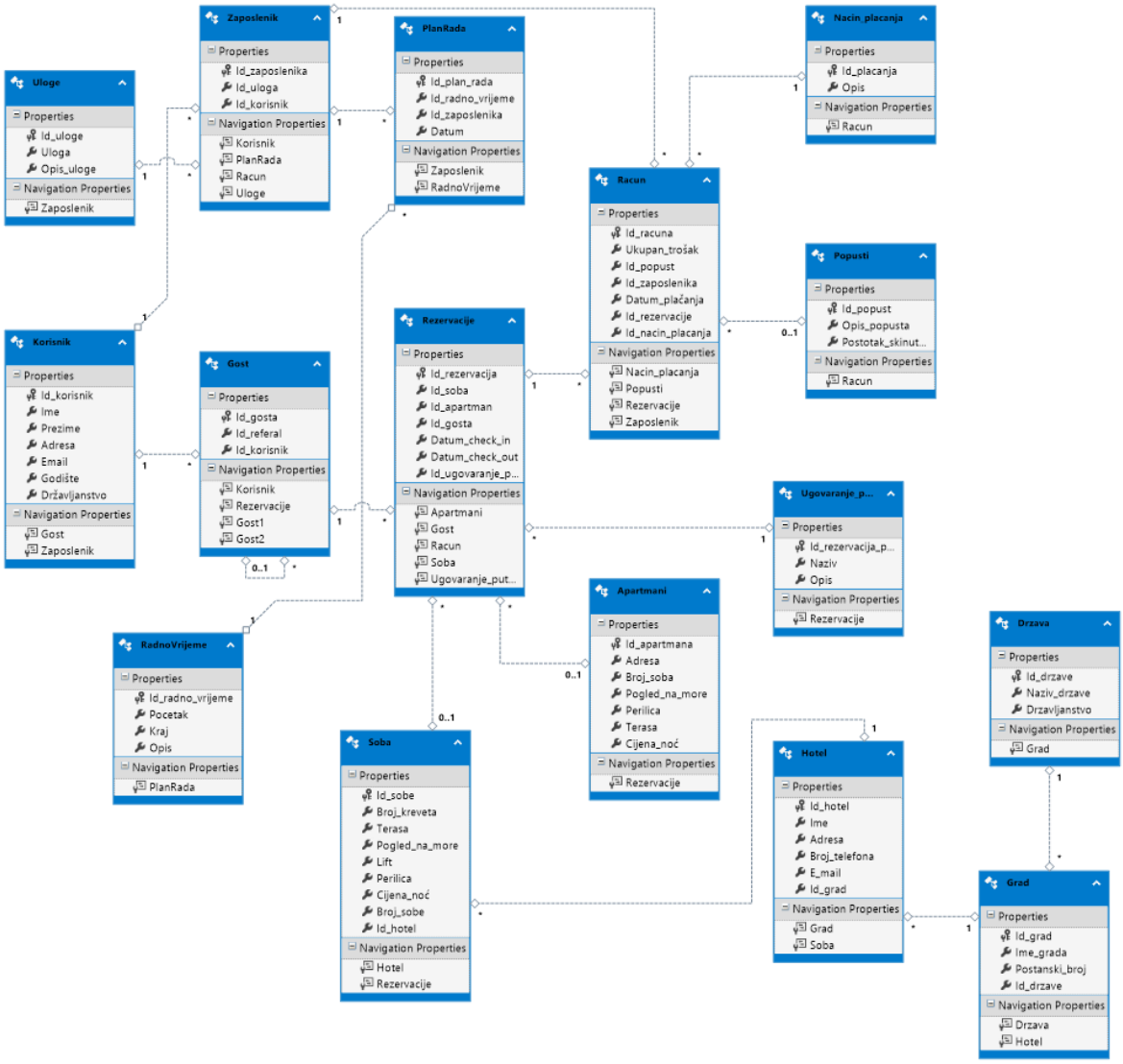



Slika 6. Tablice 'Nacin_placanja' i 'Popusti'

Obje tablice su jako jednostavne, kasnije ćemo objasniti zašto ih imamo dok budemo obradili tablicu 'Racun', zasada ćemo samo pokazati njihove kodove:

```
CREATE TABLE [dbo].[Popusti] (
    [Id_popust]               INT            IDENTITY (1, 1) NOT NULL,
    [Opis popusta]            NCHAR (100)    NULL,
    [Postotak skinute cijene] DECIMAL (2, 2) NULL,
    PRIMARY KEY CLUSTERED ([Id_popust] ASC)
);

CREATE TABLE [dbo].[Nacin_placanja] (
    [Id_placanja] INT         IDENTITY (1, 1) NOT NULL,
    [Opis]        NCHAR (100) NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_placanja] ASC)
);
```

Sada ćemo se vratiti na sliku 1, kako bi mogli objasniti tablice 'Rezervacija' i 'Racun' i njihovu logiku i povezanost sa drugim tablicama.




Slika 1, 7. ERA model naše baze za hotel

Prije nego pogledamo kako su povezane preostale tablice 'Rezervacije' i 'Racun' moramo prvo objasniti logiku pri stvaranju tih dviju tablica. Naime, u planu jest da Gost hotela može rezervirati sobu ili apartman određenog datuma i određenog trajanja, također ta rezervacija ima različite načine na koji se mogu izvršiti, npr, putem aplikacije Booking.com, zvanjem hotela, Air b'n'b, preko neke treće agencije itd. Nakon što je obavljena rezervacija, i odmor gostu se bliži kraju, gost je primoran platiti usluge koje mu je hotel pružio. Tu dolazi do nastajanja nove tablice, a to je tablica 'Racun'. Gostu na računu će biti ispisana rezervacija, datum plačanja, ime zaposlenika koji mu je ispisao račun te također i eventualne popuste koje je ostvario i način na koji je platio taj račun. Sada možemo pogledati kodove tablica 'Rezervacije' i 'Racun':

```
CREATE TABLE [dbo].[Rezervacije] (
    [Id_rezervacija]      INT  IDENTITY (1, 1) NOT NULL,
    [Id_soba]             INT  NULL,
    [Id_apartman]         INT  NULL,
    [Id_gosta]            INT  NOT NULL,
    [Datum check-in]      DATE NOT NULL,
    [Datum check-out]     DATE NOT NULL,
    [Id_ugovaranje_putem] INT  NOT NULL,
    PRIMARY KEY CLUSTERED ([Id_rezervacija] ASC),
    CONSTRAINT [IdUgovaranjePutemFK] FOREIGN KEY ([Id_ugovaranje_putem]) REFERENCES
[dbo].[Ugovaranje_putem] ([Id_rezervacija_putem]),
    CONSTRAINT [IdApartmanFK] FOREIGN KEY ([Id_apartman]) REFERENCES [dbo].[Apartmani]
([Id_apartmana]),
    CONSTRAINT [IdSobaFK] FOREIGN KEY ([Id_soba]) REFERENCES [dbo].[Soba] ([Id_sobe]),
    CONSTRAINT [IdGostaFK] FOREIGN KEY ([Id_gosta]) REFERENCES [dbo].[Gost]
([Id_gosta]),
    CONSTRAINT [CK_Rezervacije_Provjera_check_in_veci_od_check_out] CHECK ([Datum
check-out]>[Datum check-in])
);
```

Kao što vidimo, tablica 'Rezervacije' nije jednostavna pošto vidimo da u sebi ima nekoliko sekundarnih ključeva. Također ima i jedan CHECK koji provjerava da se u polju **Datum check-in** nije upisao datum koji je veći od zapisa u polju **Datum check-out**. Pošto je to nemoguće i ne smije se to dogoditi.

```
CONSTRAINT [CK_Rezervacije_Provjera_check_in_veci_od_check_out] CHECK ([Datum check-
out]>[Datum check-in])
```

Nadalje, moramo objasniti logiku povezanih tablica sa tablicom 'Rezervacije'. Sada ćemo objasniti i logiku između veza u našem ERA modelu s tablicom 'Rezervacije'. Logika: Može se barem jedna soba rezervirati, ali i ne mora, isto važi i za apartman, koji se može a i ne mora rezervirati. Dok više soba i/ili apartmana mogu biti rezervirana. Rezervacija može biti na ime samo jednog gosta dok više gosta mogu obaviti rezervaciju. Te ugovaranje rezervacije se može izvršiti na više načina dok jedna rezervacija može biti ugovorena na samo jedan od tih način. Po ovoj logici možemo i zaključiti da većina veza u ERA modelu sa našom tablicom povezane su binarnom vezom jedan naprema više što smo sada i pokazati.
Sljedeće imamo tablicu 'Racun'

```
CREATE TABLE [dbo].[Racun] (
    [Id_racuna]         INT      IDENTITY (1000, 1) NOT NULL,
    [Ukupan trošak]     MONEY    NULL,
    [Id_popust]         INT      NULL,
    [Id_zaposlenika]    INT      NOT NULL,
    [Datum plačanja]    DATETIME NOT NULL,
    [Id_rezervacije]    INT      NOT NULL,
    [Id_nacin_placanja] INT      NULL,
    PRIMARY KEY CLUSTERED ([Id_racuna] ASC),
    CONSTRAINT [zaposlenikiFK] FOREIGN KEY ([Id_zaposlenika]) REFERENCES
[dbo].[Zaposlenik] ([Id_zaposlenika]),
    CONSTRAINT [popustFK] FOREIGN KEY ([Id_popust]) REFERENCES [dbo].[Popusti]
([Id_popust]),
```

```
    CONSTRAINT [nacin_placanjaFK] FOREIGN KEY ([Id_nacin_placanja]) REFERENCES
[dbo].[Nacin_placanja] ([Id_placanja]),
    CONSTRAINT [rezervacijeFK] FOREIGN KEY ([Id_rezervacije]) REFERENCES
[dbo].[Rezervacije] ([Id_rezervacija]),
    CONSTRAINT [CK_Racun_Datum_placanja] CHECK ([Datum plačanja]>=getdate())
);
```

Kao što vidimo tablicu 'Racun' možemo shvaćati kao zadnjom tablicom, pošto je povezana na neki način sa svim drugim tablicama. Tablica 'Racun' kao što sam naziv govori, ispisuje račun. Zbog čega u tablici moramo imati par bitnih polja koji bi se morali nalaziti na svakom računu, a to su: imamo dva polja koje korisnik ne može sam upisati već vrijednost primaju automatski, a to su **Id_racuna** i **Ukupan trošak** u kojem, kao što smo maloprije rekli, vrijednost je zapisana automatski, zbog triggera koji ćemo kasnije spomenuti. Nadalje, mora imati **Id_zaposlenika** što označava zaposlenika koji je ispisao račun, **Datum plačanja** koji zbog CHECK CONSTRAINT-a ne može biti manji od trenutnog vremena.

```
CONSTRAINT [CK_Racun_Datum_placanja] CHECK ([Datum plačanja]>=getdate())
```

Također, imamo još dva polja, a to su **Id_rezervacije,** i zadnji **Id_nacin_placanja** (koji može biti upisan, ali i ne mora). Osim polja, ima i par sekundarnih ključeva koji se povezuju na druge tablice, što ćemo objasniti odmah i objasniti. Logika između povezivanja navedene tablice sa drugima jest sljedeća: Jedan račun može imati jedan ili niti jedan popust, isti popust može biti na više računa. Isto vrijedi i za vezu između 'Racun' i 'Nacin_placanja' pošto je jedan naprema više, tj. Jedan račun se može platiti na jedan način dok na više načina se mogu platiti svaki račun. Isto zaposlenik može ispisati više računa, dok jedan račun može biti ispisan samo od jednog zaposlenika. Te može biti više rezervacija na računu, dok samo jedna rezervacija može biti ispisana na računu. Što također čini 1:M vezu. Sljedeće ćemo pokazati sve triggere koje imamo u našoj bazi, a zatim i par jednostavnih i složenih upita koje smo napravili.

# 4. Triggeri

Okidač (engl. Trigger) je nešto što se poziva prije, nakon i/ili poslije nekog upisa, ažuriranja ili brisanja iz baze. Te služe za upis, ažuriranje i/ili brisanje nekog zapisa u bazi. Oni uvelike olakšavaju način na koji se pohranjuje novi zapis u neku tablicu. Pošto se pozivaju, naravno ovisno o tome kada je to specificirano, prije, nakon i/ili poslije nekog upisa najčešće radi provjere valjanosti upisa, provjere sa drugom bazom, ažuriranja vrijednosti u nekom polju, itd. U SQL Serveru on je zapisan uz sam kod koji je i vezan za tu tablicu. Znači na istom mjestu, što uvelike olakšava traženje gdje je zapisan koji okidač, ako nam se ne da tražiti po samom upitu gdje se koji nalazi preko naredbe

```
select * from sys.triggers
```

Sada ćemo pokazati sve naše napravljene okidače i objasniti koju zadaću svaki obavlja.

## 4.1. Okidači u bazi Hotel

Prvo ćemo pojasniti malo lakše okidače u našoj bazi, te ćemo polako doći do malo kompliciranijih.

S prvim okidačem se odmah susrećemo u tablici 'Apartmani' koji glasi ovako:

```
CREATE TRIGGER [dbo].[Trigger_Apartmani]
   ON [dbo].[Apartmani]
   FOR INSERT
   AS
   BEGIN
       SELECT * FROM inserted;
   END
```

Prvo, FOR INSERT znači da DML okidač samo kada su sve operacije specificirane u SQL naredbi uspješno pokrenute. Tek tada će se ispaliti i izvršiti naredbu specificiranu u našem okidaču. A ona se nalazi točno između BEGIN i END. Ona ispisuje sve novo ubačene apartmane u tablicu. Tablica 'inserted' u ovom slučaju je View iz kojeg možemo dohvatiti sve nove podatke koje želimo ubaciti u našu tablicu.

Sljedeći okidač nam se nalazi u tablici 'Gost'.

```
CREATE TRIGGER [dbo].[Trigger_Gost_Postoji_Referal]
    ON [dbo].[Gost]
    AFTER INSERT
    AS
    BEGIN
              PRINT 'Preporucio ga je gost: '
              SELECT k.Ime, k.Prezime FROM Gost g
              JOIN   Korisnik k
              ON            k.Id_korisnik = g.Id_korisnik
              WHERE  EXISTS ( SELECT 1 FROM inserted i WHERE i.Id_referal =
g.Id_gosta);

    END
```

On nam također okida tek nakon što su sve operacije specificirane u SQL naredbi uspješno izvršene. Te, nakon što ubacimo novi slog u našu tablicu, ispisuje poruku i prikazuje ako uopće postoji, koji **Ime** i **Prezime** gosta koji je preporučio novom gostu naš hotel.

Sad dolazimo do malo kompliciranijih okidača. Prvo ćemo prikazati okidač koji se nalazi u tablici 'Rezervacije', te on glasi:

```
CREATE TRIGGER [dbo].[Trigger_Rezervacije_vec_rezervirano]
    ON [dbo].[Rezervacije]
    INSTEAD OF INSERT
    AS
    BEGIN
       DECLARE @datumIn                 date,  @datumNovogGosta     date
       DECLARE @datumOut                date,  @datumNovogGostaOut date
       DECLARE @IdApartman              int,   @IdNoviApartman      int
       DECLARE @IdSoba                  int,   @IdNovaSoba          int
       DECLARE @proba                   date

              SELECT @IdNoviApartman      = [Id_apartman],
                     @IdNovaSoba          = [Id_soba],
                     @datumNovogGosta     = [Datum check-in],
                     @datumNovogGostaOut  = [Datum check-out]
              FROM   inserted

              SELECT *
              INTO #tempTable
              FROM inserted

              SELECT @datumIn      = r.[Datum check-in],
                     @IdApartman   = r.[Id_apartman],
                     @IdSoba       = r.[Id_soba],
                     @datumOut     = r.[Datum check-out]
              FROM   Rezervacije r
              JOIN   #tempTable t ON r.Id_soba = t.Id_soba

              IF
              (@IdSoba = @IdNovaSoba OR @IdNoviApartman = @IdApartman)
              AND((@datumIn < @datumNovogGosta AND @datumOut > @datumNovogGosta)
              OR (@datumIn < @datumNovogGostaOut AND @datumOut > @datumNovogGostaOut)
              OR (@datumIn  > @datumNovogGosta AND @datumOut < @datumNovogGostaOut))
                     BEGIN
                            RAISERROR ('Apartman/sobu koju je gost rezervirao za taj
                            datum je vec rezervirana/zauzeta!', 16, 1)
                     END
              ELSE
                     BEGIN
```

```
                    INSERT INTO [Rezervacije] (Id_soba, Id_apartman, Id_gosta,
                    [Datum check-in], [Datum check-out], Id_ugovaranje_putem)
                    SELECT Id_soba, Id_apartman, Id_gosta, [Datum check-in],
                    [Datum check-out], Id_ugovaranje_putem
                    FROM inserted
                END
    END
```

Ovaj okidač se izvršava prije i umjesto nove upisane vrijednosti. Kao što možemo vidjeti, otišao sam pisati navedeni okidač na dosta neobičan način u odnosu koji smo radili u PostgreSQL-u. Zahvaljujući SQL Serveru koji mi je malo olakšao pisanje samog okidača pošto sam mogao koristiti varijable u koje sam mogao spremati određene vrijednosti na koje sam se kasnije mogao pozivati. Zbog čega imamo na početku toliko puno DECLARE funkcija sa @imenom varijable i tipom podataka koji se može spremiti u tu varijablu. Nego, prije nego krenemo s objašnjavanjem što svaka linija koda radi, moramo objasniti što smo htjeli postići sa navedenim okidačem. Okidač **Trigger_Rezervacije_vec_rezervirano** služi da se provjeri ako je već neki drugi gost rezervirao taj termin s tom istom sobom ili apartmanom, te ako je, u tom slučaju ispisuje poruka sa pogreškom da je navedena soba ili apartman već rezervirana. Dok u slučaju da nije, upisuje se nova rezervacija, tj. slog u tablicu. Nakon što samo unaprijed specificirao sve varijable i njihove tip koji može sadržati podataka, počeli smo svaku varijablu puniti. Prvo smo napunili varijable iz pogleda (engl. View) u kojima nam se nalaze novo upisani podaci, te zatim i podacima koji već postoje u toj tablici, pod uvjetom da ima identično polje **Id_soba** kao i novo uneseni podatak. Taj uvjet smo postigli tako što smo pogled zapisali u privremenu tablicu **#tempTable**, pošto se pogledom ne može manipulirati kao što se može i tablicom. Nakon podosta dugog IF izjava (engl. Statementa) koji provjerava datume dolaska i odlaska ako se poklapaju sa trenutnim u samoj bazi, te ako je izjava točan, označava da je apartman ili soba već rezervirana te ispisuje RAISERROR sa određenom porukom. Dok ako izjava nije istinit, tj. Nije rezervirano, novi slog upisuje se u tablicu.

Te sad imamo i zadnji okidač, **Trigger_Racun_azuriranje_ukupne_cijene:**

```
CREATE TRIGGER [dbo].[Trigger_Racun_azuriranje_ukupne_cijene]
    ON [dbo].[Racun]
    INSTEAD OF INSERT
    AS
    BEGIN
       DECLARE      @popust             DECIMAL (2, 2)
       DECLARE      @cijenaSobe         MONEY
       DECLARE      @cijenaApartman     MONEY
       DECLARE      @ukupnaCijena       MONEY
       DECLARE @Idrezervacija     INT,   @IdNovaRezervacija   INT

       SELECT *
       INTO   #tempTable
       FROM   inserted

       SELECT @IdNovaRezervacija   = Id_rezervacije
       FROM   #tempTable
```

```
        SELECT @Idrezervacija              = r.Id_rezervacije
        FROM   Racun r, #tempTable t
        WHERE  t.Id_rezervacije = r.Id_rezervacije

        SELECT @popust = [Postotak skinute cijene]
        FROM   Popusti p
        JOIN   #tempTable t
        ON     p.Id_popust   = t.Id_popust

        SELECT @cijenaSobe = s.[Cijena/noć] * DATEDIFF(DAY,r.[Datum check-in],r.[Datum
        check-out])
        FROM   Soba s, Rezervacije r, #tempTable t
        WHERE  t.Id_rezervacije = r.Id_rezervacija AND r.Id_soba = s.Id_sobe

        SELECT @cijenaApartman = a.[Cijena/noć] * DATEDIFF(DAY,r.[Datum check-in],
               r.[Datum check-out])
        FROM   Apartmani a, Rezervacije r, #tempTable t
        WHERE  t.Id_rezervacije = r.Id_rezervacija AND r.Id_apartman = a.Id_apartmana

        IF
               @popust IS NULL AND @cijenaSobe IS NOT NULL
               BEGIN
                      INSERT INTO Racun([Ukupan trošak], Id_popust, Id_zaposlenika,
                      [Datum plačanja], Id_rezervacije, Id_nacin_placanja)
                      SELECT @cijenaSobe, Id_popust, Id_zaposlenika, [Datum plačanja],
                      Id_rezervacije, Id_nacin_placanja
                      FROM inserted
               END
        IF
               @popust IS NOT NULL AND @cijenaSobe IS NOT NULL
               BEGIN
                      SELECT @ukupnaCijena = @cijenaSobe - (@cijenaSobe * @popust)
                      INSERT INTO Racun([Ukupan trošak], Id_popust, Id_zaposlenika,
[Datum plačanja], Id_rezervacije, Id_nacin_placanja)
                      SELECT @ukupnaCijena, Id_popust, Id_zaposlenika, [Datum plačanja],
                      Id_rezervacije, Id_nacin_placanja
                      FROM inserted
               END
        IF
               @popust IS NULL AND @cijenaApartman IS NOT NULL
               BEGIN
                      INSERT INTO Racun([Ukupan trošak], Id_popust, Id_zaposlenika,
                      [Datum plačanja], Id_rezervacije, Id_nacin_placanja)
                      SELECT @cijenaApartman, Id_popust, Id_zaposlenika, [Datum
                      plačanja], Id_rezervacije, Id_nacin_placanja
                      FROM inserted
               END
        IF
               @popust IS NOT NULL AND @cijenaApartman IS NOT NULL
               BEGIN
                      SELECT @ukupnaCijena = @cijenaApartman - (@cijenaApartman *
                      @popust)
                      INSERT INTO Racun([Ukupan trošak], Id_popust, Id_zaposlenika,
                      [Datum plačanja], Id_rezervacije, Id_nacin_placanja)
                      SELECT @ukupnaCijena, Id_popust, Id_zaposlenika, [Datum plačanja],
                      Id_rezervacije, Id_nacin_placanja
                      FROM inserted
               END

        IF
               (@cijenaSobe IS NULL AND @cijenaApartman IS NULL) OR (@Idrezervacija =
               @IdNovaRezervacija)
```

```
		BEGIN
			RAISERROR ('Niste upisali ID sobe niti ID apartmana!!', 16, 1)
		END
END
```

Navedeni okidač također se izvršava prije i umjesto nove upisane vrijednosti. Njegova zadaća jest da ažurira i/ili sam upiše vrijednost polja **Ukupan trošak** u tablici 'Racun' ovisno o koliko dana je koji gost gostovao u našem hotelu. Te u slučaju da je gost ostvario popust, oduzeti od sume iznos za navedeni popust. Sad ćemo i objasniti kako smo to ostvarili. Prvo smo deklarirali par varijabli na koje ćemo se kasnije pozivati. U varijabli @popust stavili smo vrijednost **Postotak skinute cijene** iz tablice 'Popusti', stoga da njezin **Id_popusti** mora odgovarati istom polju koji je unesen. Te kasnije samo ako postoji uneseni će se navedeni popust oduzeti od ukupne sume koju gost mora platiti. Sljedeće što smo napravili jest u varijablu @cijenaSobe i @cijenaApartmana tipa MONEY upisali smo ukupnu sumu, tj. cijenu koju gost treba platiti, koju smo dobili množenjem te sume s funkcijom DATEDIFF koja nam je dala broj dana od **Datum check-in** do **Datum check-out**. Također, morali smo i povezati tablice 'Soba', 'Apartman' i #**tempTable** kako bi mogli uopće i dobiti navedeni iznos. To smo uradili iz razloga što smo iz svake od navedenih tablica morali izvaditi određeni zapis u određenom od navedenih polja u kodu radi dobivanja ukupne sume koštanja broja svih noćenja u hotelu ili apartmanu. Nakon što smo to naveli, imamo par IF izjava kojima provjeravamo dali postoji ili ne postoji popust, te ovisno o toj izjavi se oduzima ili ne popust od ukupne cijene. Dok u slučaju da rezervacija nije upisana, okidač izbacuje Error sa porukom da nešto nije uredu.

# 5. Upiti

U ovom poglavlju navesti ćemo par upita koji će nam poslužiti pri korištenju naše baze podataka. Prvo ćemo prikazati par jednostavnih pa ćemo preći na složenije. Upiti su pisani u SQLQuery, te navedeni program će uz projekt biti objavljen na stranici moodla kako bi mogli pogledati navedeni program, a i navedenu bazu o kojoj već toliko dugo pričamo.

## 5.1. Jednostavni

Prvo ćemo pojasniti malo jednostavne upite:

```
--Ispisuje sve korisnike
SELECT * FROM Korisnik;

--Ispisuje sve hotele
SELECT * FROM Hotel;

--Ispisuje sve goste
SELECT * FROM Gost;

--Gosti koji imaju barem jedan referal!
SELECT * FROM Gost
WHERE [Id_referal] IS NOT NULL;

SELECT * FROM Korisnik;

--Pretraga specifičnog zaposlenika sa određenim ID-em
SELECT * FROM Zaposlenik
WHERE Id_zaposlenika = 16;

--Pretraga zaposlenika s ID-eovima od 10 do 16
SELECT * FROM Zaposlenik
WHERE Id_zaposlenika BETWEEN 10 AND 16;

--Pretraga svih uloga
SELECT u.[Opis uloge], u.Uloga  FROM Uloge u

--Pretraga svih uloga koji koje izvršavaju recepcioneri
SELECT * FROM Uloge
WHERE [Uloga] LIKE '%recepcioner%'

--Pretraga soba i apartmana kojima noćenje dođe manje od 170€
SELECT * FROM Soba s
WHERE s.[Cijena/noć] < 170;

--Pretraga svih soba koje imaju pogled na more i da noćenje dođe manje od 180€
SELECT Broj_sobe AS "Broj sobe", [Cijena/noć] AS "Cijena noćenja"
FROM Soba
WHERE [Pogled na more] = 'Da' AND [Cijena/noć] > 180;
```

## 5.2. Složeni

Sada ćemo pokazati i složenije:

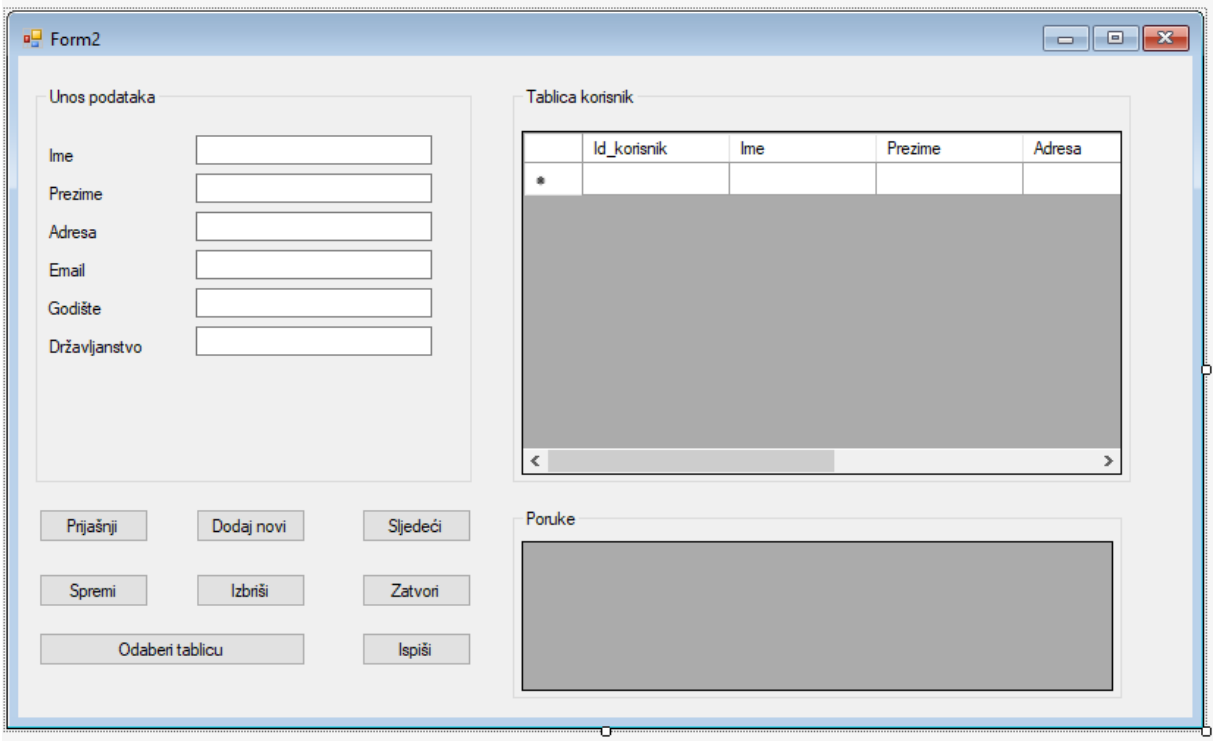
```
--Prikazuje Ime, Prezime i Adresu iz zablice zaposlenika koji imaju ID jednak 1, 5, 6,
8 i/ili 10,
--te poredani su uzlazno
SELECT k.Ime, k.Prezime, k.Adresa
FROM Zaposlenik z
JOIN Korisnik k
ON k.Id_korisnik = z.Id_korisnik
WHERE Id_zaposlenika IN (5, 6, 8, 10)
ORDER BY Ime ASC;

--Ukupan broj gostiju
SELECT COUNT(*) AS 'Ukupan broj gositju' FROM Gost;

--Ukupan broj gostiju koji nisu dobili nikakvu preporuku od drugog gosta
SELECT COUNT(*) AS 'Broj gostiju bez dobivenih preporuka' FROM Gost
WHERE Id_referal IS NULL;

--Ukupan broj gostiju koji su dobili preporuku od drugog gosta
SELECT COUNT(Id_referal) AS 'Broj gostiju koji su dobili preporuku od drugog gosta'
FROM Gost;

--Najsuplja soba s noćenjem
SELECT MAX([Cijena/noć]) AS 'Najveća cijena sobe' FROMSoba;

--Najjeftinija soba s noćenjem
SELECT MIN([Cijena/noć]) AS 'Najmanja cijena sobe' FROM      Soba;

--Nađi ime i prezime gosta koji je plaio racun i rezervirao apartman/sobu!
NADOPUNI PODATKE U TABLICI!!!
SELECT k.[Ime] AS "Ime gosta", k.[Prezime] AS "Prezime gosta", k.Godište--,
g.Državljanstvo
FROM   Gost g JOIN Rezervacije r
ON            g.Id_gosta = r.Id_gosta
JOIN   Racun ra
ON            ra.Id_rezervacije = r.Id_rezervacija
JOIN   Korisnik k
ON            g.Id_korisnik = k.Id_korisnik

--Nađi sve o zaposlenicima
SELECT k.Ime AS "Ime Zaposlenika", k.Prezime AS "Prezime Zaposlenika",
              k.Adresa, k.Email, u.Uloga, u.[Opis uloge]
FROM   Zaposlenik z JOIN Uloge u
ON            z.Id_uloga = u.Id_uloge
JOIN   Korisnik k
ON            z.Id_korisnik = k.Id_korisnik

--Nađi sve o zaposleniku koji je ispisao račun gostu
SELECT k.Ime, k.Prezime, k.Adresa, r.Id_racuna AS "ID ispisanog racuna", r.[Ukupan
trošak] AS "Cijena za uplatiti",
              r.[Datum plačanja] AS "Datum ispisanog racuna", u.Uloga, u.[Opis uloge]
FROM   Zaposlenik z, Racun r, Uloge u, Korisnik k
WHERE  z.Id_zaposlenika = r.Id_zaposlenika AND z.Id_uloga = u.Id_uloge AND
k.Id_korisnik = z.Id_korisnik

--Nađi sve o ispisanog računu!
SELECT k.Ime, k.Prezime, k.Adresa, r.Id_racuna AS "ID ispisanog racuna", r.[Ukupan
trošak] AS "Cijena za uplatiti",
```

```
            r.[Datum plačanja] AS "Datum ispisanog racuna"
FROM   Zaposlenik z JOIN Racun r
ON          z.Id_zaposlenika = r.Id_zaposlenika
JOIN   Korisnik k
ON          k.Id_korisnik = z.Id_korisnik

--Ostvareni popust na isplati racuna!
SELECT r.Id_racuna,
            (s.[Cijena/noć]*DATEDIFF(DAY,rezer.[Datum check-in],rezer.[Datum check-
out])) AS "Cijena Soba bez uključenog popousta",
            r.[Ukupan trošak], p.[Opis popusta], p.[Postotak skinute cijene]
FROM   Popusti p JOIN Racun r
ON          p.Id_popust = r.Id_popust
JOIN   Rezervacije rezer
ON          rezer.Id_rezervacija = r.Id_rezervacije
JOIN   Soba s
ON          s.Id_sobe = rezer.Id_soba

--U kojem hotelu se nalazi koja soba
SELECT s.Broj_sobe AS "Broj Sobe", h.Ime AS "Naziv Hotela", g.Ime_grada AS "Grad u
kojem se nalazi hotel", h.Adresa AS "Adresa hotela"
FROM   Soba s join Hotel h
ON          s.Id_hotel = h.Id_hotel
JOIN   Grad g
ON          g.Id_grad = h.Id_grad
```

Kao što možemo vidjeti koristili smo samo jednu samo jednu funkciju u našim upitima, a to je funkcija DATEDIFF(day, … , …) koja nam daje broj dana između dva datuma. Nju smo također ako se prisjećate koristili i u okidaču. Kako bi ju kasnije mogli pomnožiti sa cijenom noćenja da bi mogli dobiti ukupnu cijenu koju gost mora platiti na računu. Ovdje smo je iskoristili da bi mogli vidjeti ostvareni popust koji je gost ostvario na računu.

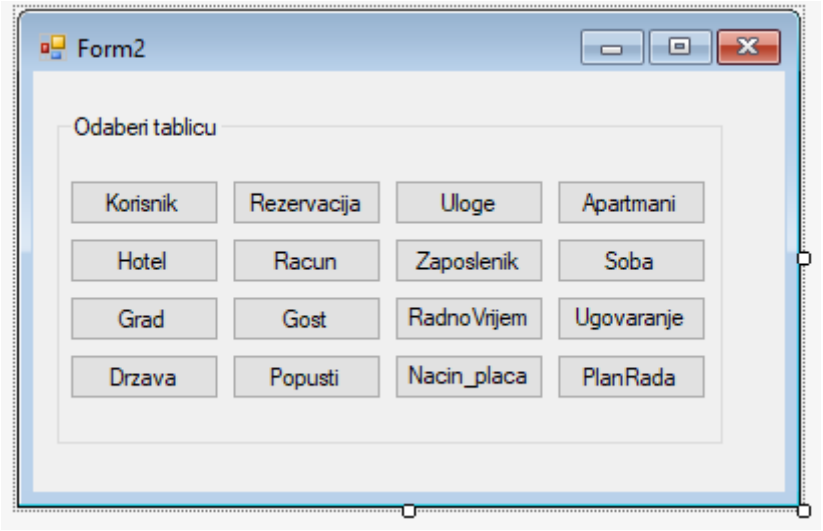
# 6. Forma

Sljedeće ćemo prikazati dvije forme koje smo napravili u jeziku C#, u našem Visual Studio 2018. okruženju. Prva tablica će nam služiti za upisivanje, ispisivanje, dodavanje, spremanje i brisanje pojedinih podataka u naše tablice. Također će nam dozvoljavati da ih se i prebacujemo sa tablice na tablicu. U navedenoj formi možemo i vidjeti samu tablicu koju koristimo i sve unesene podatke u njoj. Također pri unošenju novih podataka, u slučaju aktiviranja određenog trigger-a ili CHECK CONSTRAINT (ako tablica uopće ima), ispisat će nam poruku koji se trigger aktivirao i koji rezultat je donio. Navedena forma izgleda sljedeće:

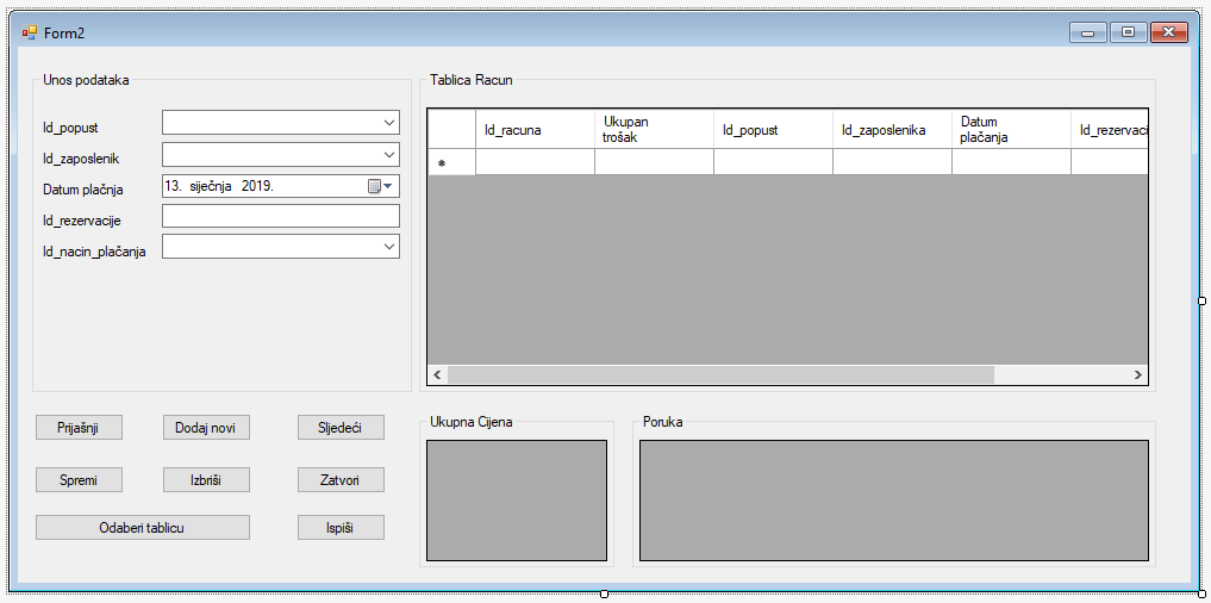



Slika 8. Forma za modeliranje tablicom Korisnik

Do sljedeće forme dolazimo klikom na gumb Odaberi tablicu, te kad se odlučimo na odabir jedne, npr. Mi ćemo odabrati tablicu 'Racun', možemo krenuti raditi sa tom tablicom, odnosno formom.



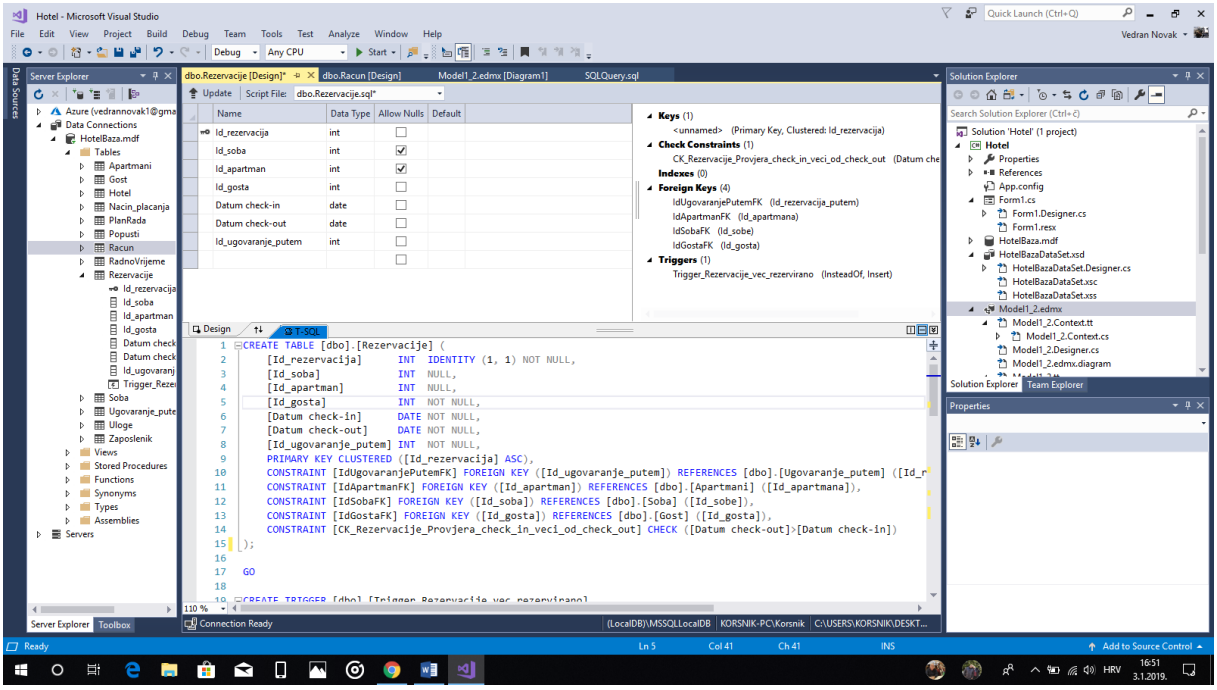

Slika 9. Forma za odabir tablice




Slika 10. Forma za modeliranjem tablicom Racun

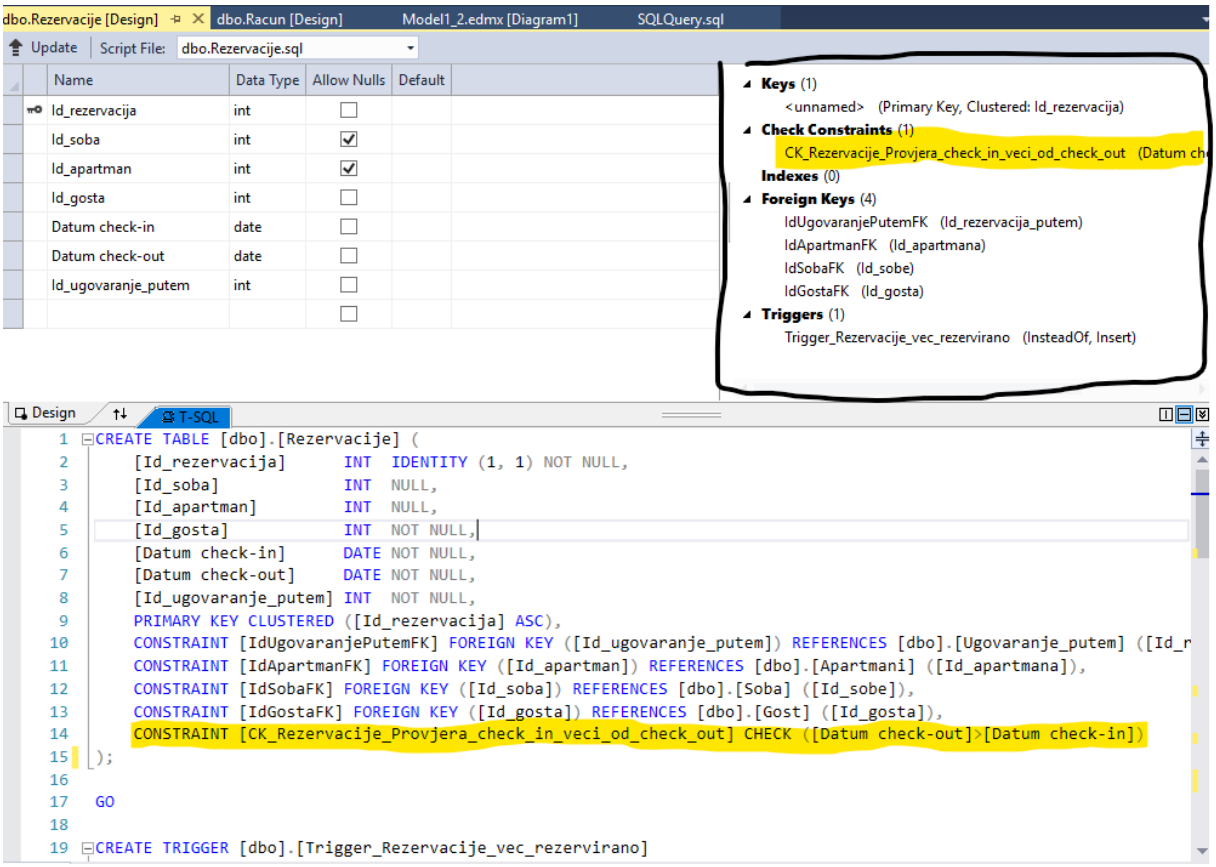
# 7. Zaključak

U radu smo koristili jezik SQL Server u integriranom razvojnom okruženju Visual Studio 2017. Visual Studio 2017 ima mnogo korisnih svojstva (engl. Feature) koje su nam podosta olakšale pisanje same baze podataka poput izvrsnog IntelliSense, sa unaprijed napravljenom tablicom u koju samo upisujemo polja i tip podataka te nam se odmah ispisuje i njezin kod ispod u SQL Server jeziku. Mogućnost generiranja dijagrama ERA modela radi provjere naše logike i lakše preglednosti. Te izvrsno napravljeni UI-a sa odlično postavljenim menijima sa svake strane, kako bi u svakom trenutku se mogli snaći gdje ste i gdje se nalazi ono što vam treba. Također, spajanje na bazu pisanu u SQL Server jeziku je jednostavno i brzo. Na sljedećoj slici ćemo pokazati kako izgleda naš IDE iz Microsofta u kojem smo radili:



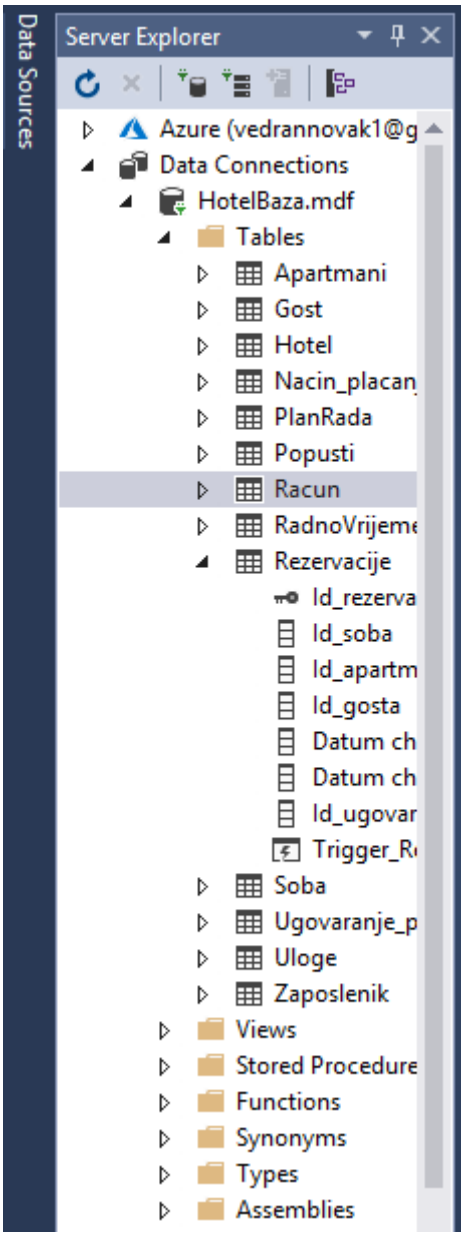

Slika 11. Integrirano razvojno okruženje (IDE) Visual Studio 2017

Desno od tablice možemo vidjeti i svojstva koja tablica ima, u ovom slučaju gledamo svojstva iz tablice 'Rezervacija'. Možemo vidjeti njezin primarni ključ, njezine sekundarne ključeve, CHECK constraintse i okidače. Također u njima ih možemo i kreirati, te kod njihovog kreiranja u donjem dijelu prozora T-SQL generirat će nam se njihova sintaksa kojom možemo manipulirati u našu koristi, tj. Ovisno što želimo postići ili napraviti.



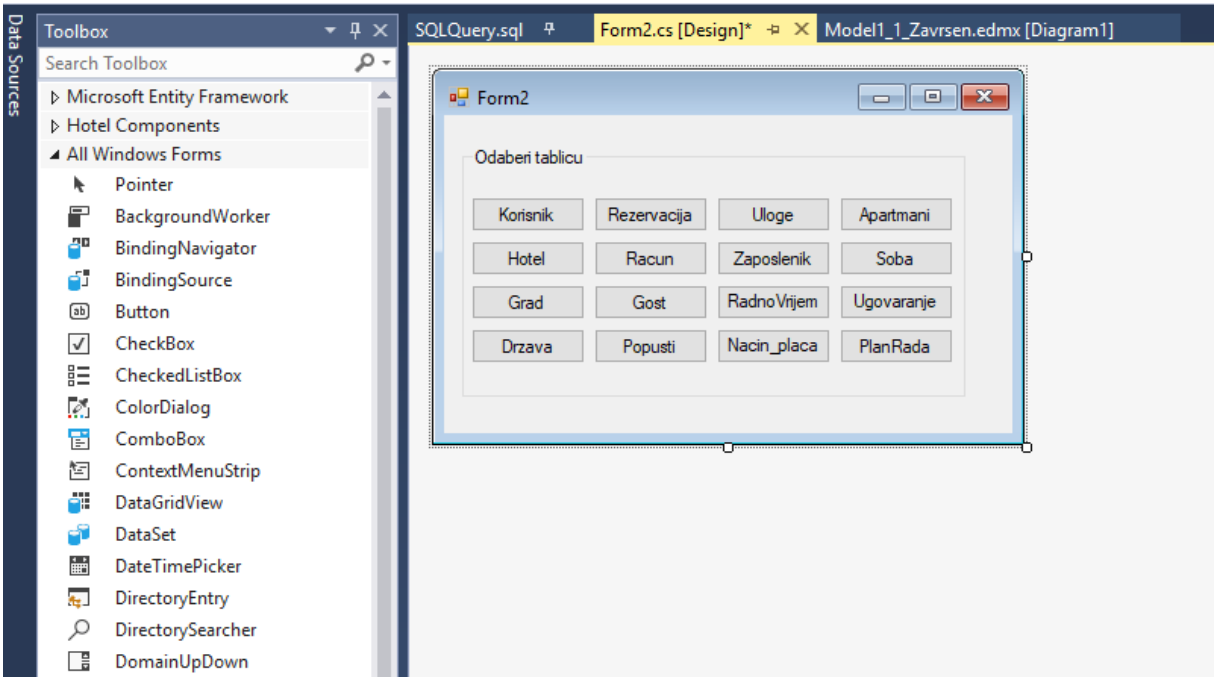

Slika 12. Prikaz raznih svojstva koje nam Visual Studio 2017 nudi

Na slici 9. pogledajmo u obojane dijelove slike. Kliknemo li desnim klikom na gornji **Check Constraints** stvori nam se padajući meni koji nam nudi da stvorimo novi CHECK koji automatski generira sintaksu u T-SQL-u. Te njega kasnije možemo modificirati, kao što i jesmo, na način da mu promijenimo ime ali i funkcionalnost (napomena, prije nije imao nikakvu funkciju). Isto nam je i sa sekundarnim ključevima i okidačima. Generira nam se sintaksa ali na nama je da promijenimo sve ostalo. Također, kao što smo maloprije spomenuli imamo i padajući istraživač (engl. Explorer) koji nam uvelike olakšava snalaženje po samoj bazi. Možemo ga vidjeti i na sljedećoj slici.




Slika 13. Istraživač sa lijeve strane

U njemu kao što možemo vidjeti, nalaze se sve bitne kreirane stavke u našoj bazi podataka, poput tablica, pogleda, funkcija, tipova podataka, spremljenih procedura itd.




Slika 14. Kreiranje forma

Također pri izradi forma, navedeni meni sa lijeve strane nam se promijenio u alatnu traku (engl. Toolbox) koju smo također koristili radi lakšeg dizajniranja sučelja naše aplikacije. Funkcionirao je na principu „Drag and Drop“, dok smo pojedino dugme i određene prozorčiće morali sami programirati. Iako nam je i to uvelike olakšano radi automatskog generiranja koda. Jedino što je bilo malo teže, kod kreiranja formi, jest samo spajanje na sam folder od navedene baze, pošto nisam koristio IDE SQL Server, već Visual Studio 2017, sa proširenjem za SQL, tj. radom sa bazom podataka. Zbog čega sam morao modificirati konekciju, ne na bazu već na samu datoteku.

Visual Studio je općenito izvrsno okruženje, u kojem se ne mora samo baza raditi. Mi smo osim baze podataka, koristili IDE za kreiranje i jednostavnog programa koji je spojen na našu bazu te u nju upisuje i ispisuje podatke iz naše baze.

# Popis literature

Geoffrey Sparks, sparks@sparxsystems.com.au . Database Modeling in UML. Preuzeto 26.12.2018. https://elf.foi.hr/pluginfile.php/10631/mod_resource/content/1/Database_Modeling_In_UML%5B1%5D.pdf

Burak Guzel, (2010.), SQL for Beginners. Preuzeto 27.12.2019. https://bit.ly/2EgLt0m

SQL Turtorial, Pogledano 26.12.2018, https://www.w3schools.com/SQl/sql_ref_sqlserver.asp

Martina Šestak, (2018). Pogledano 26.12.2018, https://elf.foi.hr/course/view.php?id=99

# Popis slika