Sveučilište J. J. Strossmayer u Osijeku

Odjel za Matematiku,

preddiplomski studij Matematika i računarstvo

MODERNI SUSTAVI BAZE PODATAKA

ZAVRŠNI PROJEKT: HOTEL

Profesor: Student:

doc. dr. sc. Slobodan Jelić Josip Pavičić

Asistenti:

Ena Pribisalić

Mateja Đumić

Osijek, lipanj 2020.

SADRŽAJ

1. Uvod.....................................................................................................2
2. Model entiteta i veza(MEV) i relacijski model......................................3
   1. MEV hotela.................................................................................3
   2. Relacijski model hotela..............................................................5
3. Kreiranje tablica...................................................................................7
4. Unosi....................................................................................................8
5. Upiti......................................................................................................9
6. Procedure...........................................................................................10
7. Okidači................................................................................................12
8. Indexi..................................................................................................14
9. Zaključak.............................................................................................15

**1.** Uvod

Hotel je ustanova namjenjena pružanju usluga najčešće kratkrotrajnog smještaja i prehrane svojim gostima, korisnicima usluge. Ovisno o željama i potrebama gostiju, hotel pruža djelomičnu ili potpunu uslugu koja obuhvaća spavanje, prehranu, zabavu i sve ostalo prilagođeno potrebama gostiju. Hotelske sobe dijele se prema broju ležaja koje su, ovisno o kategoriji, opremljene svim potrebnim za potpunu uslugu i odmor gostiju o čemu se brinu zaposlenici hotela. Hotel u svom sastavu obično ima recepciju, bar i restoran, također mogu imati otvoreni i zatvoreni bazen, noćni bar, kockarnicu, saunu, frizerski salon, suvenirnicu itd.

Cilj mog projekta je realističan prikaz baze podataka tj. organiziranog skupa podataka mog zamišljenog hotela na osnovi prethodnog opisa, na kojem mogu praktično implementirati i pospješiti stečeno znanje iz predmeta moderni sustavi baze podataka.

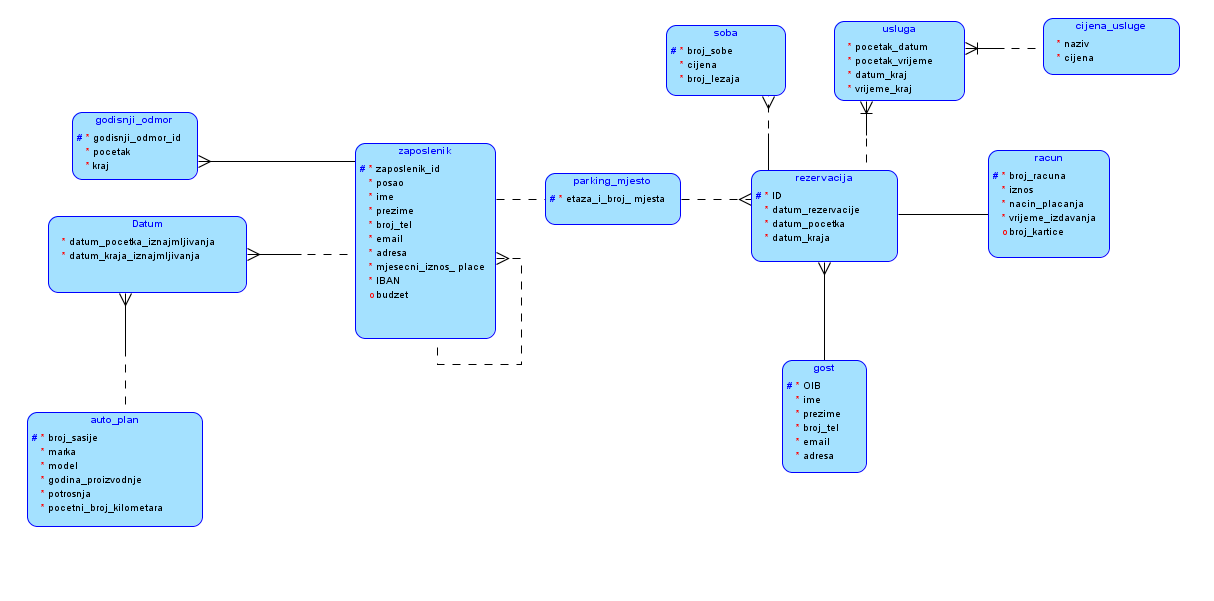
**2.** MEV(**M**odel **E**ntiteta i **V**eza) i relacijski model

Proces dizajniranja baze podataka počinje s analizom informacija koje moramo pohraniti u bazu i koje su relacije između komponenata podataka tih informacija. Za dizajniranje koristio sam MEV i Oracle-ov Data Modeler.

MEV je vizualno prirode s pravokutnicima koji predstavljaju entitete koji unutar sebe sadrže atribute te strelicama koje predstavljaju relacije između atributa. Često napravimo model entiteta i veza te ga prevedemo u relacijski model za implementaciju baze.

Relacijski model je koristan zato što ima koncept relacija, dvodimenzionalnu tablicu u kojoj su podaci organizirani. Također on podržava programski jezik visokog nivoa zvan SQL (**S**tructured **Q**uery **L**anguage).

**a.** MEV hotela



**Slika 2a.** Model entiteta i veza

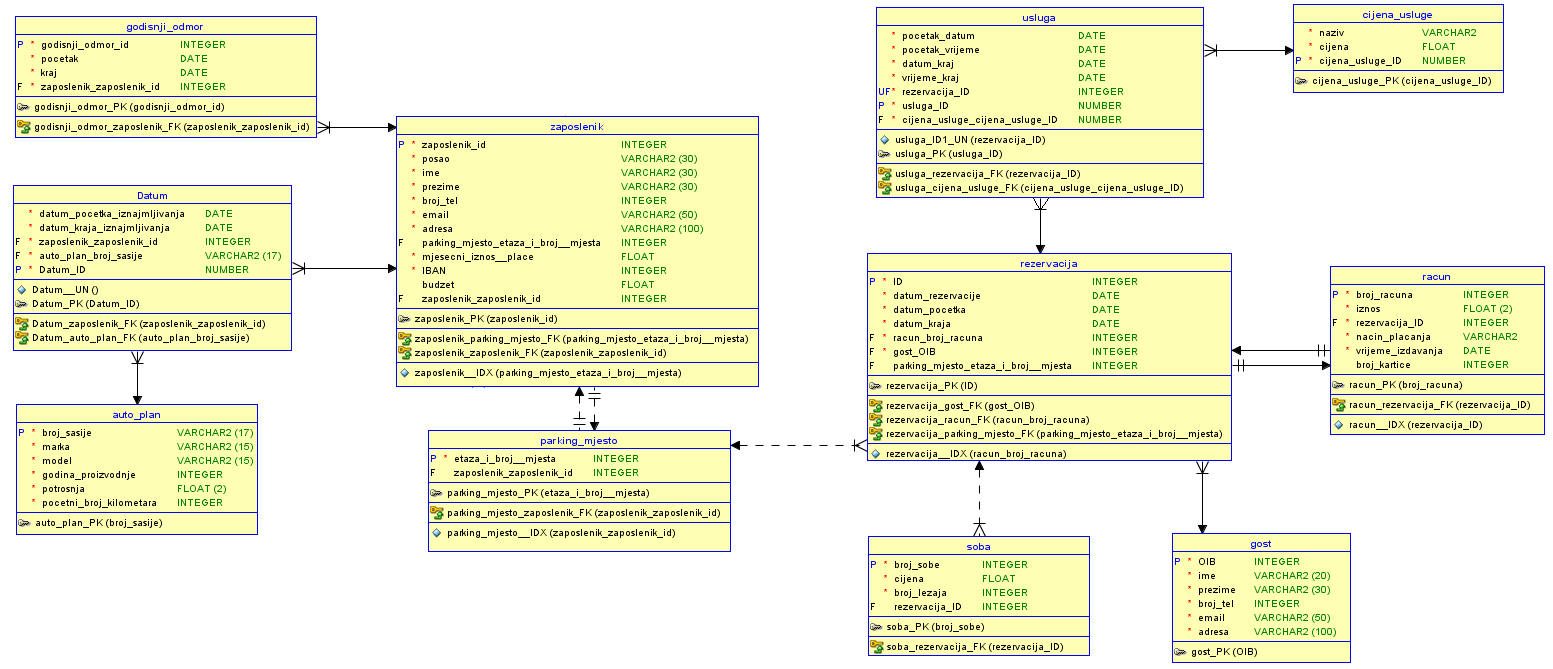
Model hotela sastoji se od 11 entiteta. Entiteti su povezani vezama od kojih su dvije više naprema više veze koje su rastavljene entitetima datum i usluga na četiri jedan naprema više veze, dakle imamo sveukupno devet jedan naprema više veza i dvije jedan naprema jedan veze koje nam govore sljedeće:

* Svaki zaposlenik mora imati godišnji odmor, kojeg može rastaviti na više manjih. Ako godišnji odmor postoji, on mora imati zaposlenika kojem je dodijeljen. (1:N)
* Zaposlenik može iznajmiti službene aute. Jedan auto može biti iznajmljen od strane više zaposlenika. (M:N)
* Svaki zaposlenik može imati parkirno mjesto rezervirano za njega. Parkirno mjesto može biti dodijeljeno zaposleniku. (1:1)
* Svaki zaposlenik može imati nadređenog i biti nadređen drugima. (1:N)
* Svaki gost mora imati jednu ili više rezervacija. Svaka rezervacija ima gosta. (1:N)
* Svaka rezervacija mora imati jedan račun. Svaki račun mora pripadati jednoj rezervaciji. (1:1)
* Svaka rezervacija mora imati jednu ili više soba. Svaka soba može imati rezervaciju. (1:N)
* Svaka rezervacija može imati više usluga. Svaka usluga može biti dodijeljena prema više rezervacija. (M:N)

Kratko objašnjenje entiteta:

* Većina entiteta je gore objašnjena.
* Treba naglasiti da su M:N veze rastavljene s dodatnim entitetom(datum, usluga) na 1:N veze.
* Opcionalnost atributa *broj\_kartice* znači ako je vrijednost atributa NULL da je plaćeno gotovinom, u suprotnom je plaćeno karticom
* Vrijeme početka i kraja usluge nam služi da pratimo iskorištenost istih te za računanje računa jer su usluge plaćene po satima

**b.** Relacijski model



**Slika 2b.** Relacijski model

Tumač:

\* → obvezan atribut

◦ → opcionalan atribut

P → primarni ključ

F → strani ključ

UF → jedinstveni strani ključ

-- → komentar

Sintaksa:

*ime\_entiteta*

(

[opis atributa(obvezna, primarni kljuc..)] *naziv\_atributa* [tip podatka u atributu]

.

.

.

)

Bit će opisani entiteti datum i rezervacija kao primjer. Ostali su analogni.

datum(

P \* Datum\_id NUMBER, --jedinstveni identifikator datuma

\* datum\_pocetka\_iznajmljivanja DATE, --atribut tipa DATE

\* datum\_kraja\_iznajmljivanja DATE, --atribut tipa DATE

F \* zaposlenik\_zaposlenik\_id INTEGER, --strani ključ koji pokazuje na zaposlenika koji

je iznajmio auto

F \* auto\_plan\_broj\_sasije VARCHAR2(17) --strani ključ koji pokazuje na auto koji je

iznajmljen

)

rezervacija(

P \* ID INTEGER, --jedinstveni identifikator rezervacije

\* datum\_rezervacije DATE, --atribut tipa DATE

\* datum\_pocetka DATE, --atribut tipa DATE

\* datum\_kraja DATE, -- atribut tipa DATE

F \* racun\_broj\_racuna INTEGER, --strani ključ koji pokazuje na račun

rezervacije

F \* gost\_OIB INTEGER, --strani ključ koji pokazuje čija je

rezervacija

F parking\_mjesto\_etaza\_i\_broj\_mjesta INTEGER --strani ključ koji pokazuje parking

mjesto koje je dodijeljeno rezervaciji

)

**3.** Kreiranje tablica

Sintaksa za kreiranje tablica u SQL-u:

CREATE TABLE *ime\_tablice*

(

*stupac\_1* tip podatka,

*stupac\_2* tip podatka,

*stupac\_3* tip podatka,

...

*stupac\_n* tip podatka

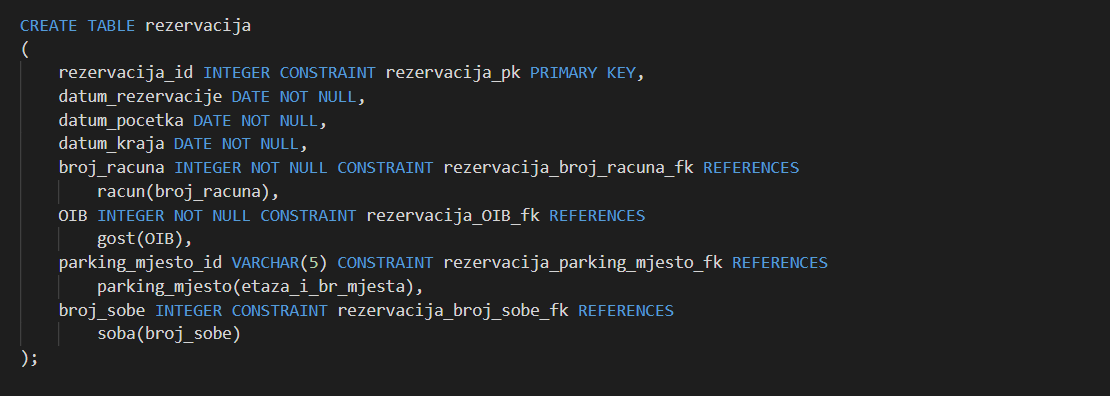
);

Brisanje tablice:

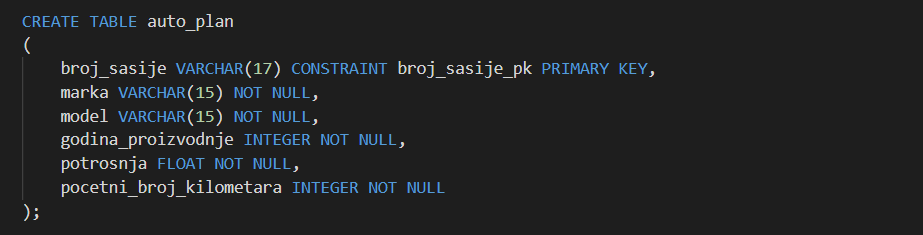
DROP TABLE ime\_tablice [CASCADE CONSTRAINTS];

Tipovi podataka u SQL-u: NUMBER, VARCHAR, DATE, DATETIME, INEGER, BOOL...

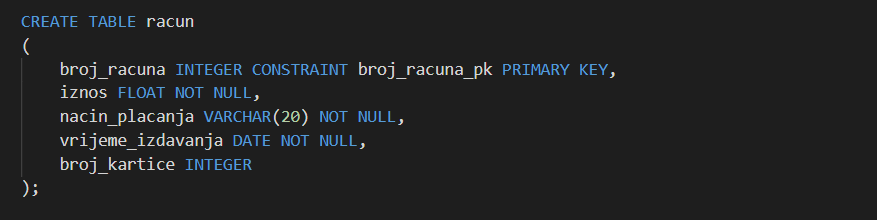
Primjer 3.1.



**Slika 3.1.** tablica rezervacija



**Slika 3.2.** tablica auto\_plan



**Slika 3.3.** tablica racun

**4.** Unosi

Unosi služe za popunjavanje naše baze podataka s podacima, odnosno vrijednostima.

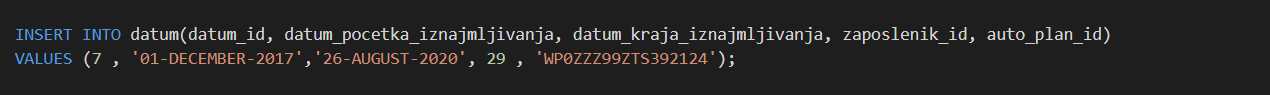
Sintaksa za unose u SQL-u ako želimo unijeti određene stupce:

INSERT INTO ime\_tablice( *stupac\_1*, *stupac\_2*, *stupac\_3*, ... )

VALUES ( *vrijednost\_1*, *vrijednost\_2*, *vrijednost\_3*, ... );

Prvi navedeni stupac, u ovom slučaju *stupac\_1*, dobiva prvu navedenu vrijednost, u ovom slučaju *vrijednost\_1.* Analogno za drugi, treći...

Primjer 4.1.



**Slika 4.1.** unos vrijdnosti u tablicu datum

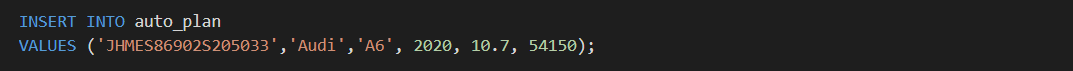
Sintaksa za unose u SQL-u ako želimo unijeti sve stupce koje su u tablici:

INSERT INTO ime\_tablice

VALUES ( *vrijednost\_1*, *vrijednost\_2*, *vrijednost\_3*, ... );

U ovom slučaju trebamo paziti da se redoslijed stupaca tablice podudara sa željenim vrijednostima.

Primjer 4.2.



**Slika 4.2.** unos vrijdnosti u tablicu auto\_plan

**5. Upiti**

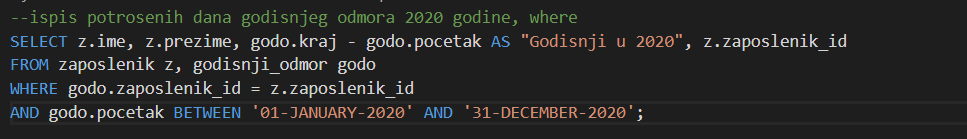
Postoje jednostavni upiti nad jednom tablicom, složeni upiti s više tablica, podupiti i ugnježdeni upiti. Upiti imaju više funkcija. Najpoznatija je prikupljanje podataka iz tablice ili tablica. Rezultat upita je jedna tablica sa svim informacijama koje smo tražili. Većinom ne želimo prikupiti sve informacije iz nekih tablica, stoga možemo filtrirati podatke na one koje želimo.

Sintaksa upita:

SELECT *stupac\_1*, *supac\_2*, ...

FROM *ime\_tablice*;

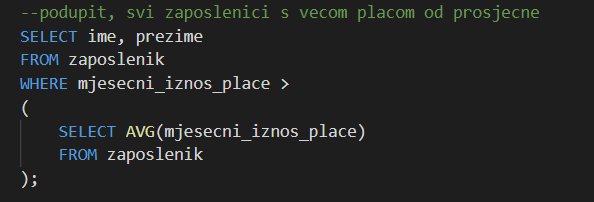
Primjer 5.1.



**Slika 5.1.** Upit

Na slici 5.1. vidimo korištenje WHERE klauzule s kojom postavljamo uvjet upitu. Prvo spojimo tablice *zaposlenik* i *godisnji\_odmor* preko stranog kljuca *zaposlenik\_id* koji je pohranjen u tablici *godisnji\_odmor*. Zatim postavimo uvjet da je godišnji počeo u rasponu od prvog do zadnje dana u godini, uključujući i prvi i zadnji dan.

Primjer 5.2.



**Slika 5.2.** Podupit

Na slici 5.2. vidimo korištenje podupita koji nam prvo vrati prosječan iznos plaće pomoću agregirajuće funkcije AVG() zatim vanjski upit uspoređuje svaki *mjesecni\_iznos\_place* iz tablice *zaposlenik* s tim prosjekom i vraća nam sve zaposlenike koji imaju plaću veću od prosječne.

**6.** Procedure

Svaka komercijala baza podataka ima način spremanja procedura koje možemo koristiti SQL upitima ili drugim SQL iskazima.

Procedure su pisane u jednostavnom proceduralnom jeziku i dopuštaju nam pozivanje istih u samoj bazi podataka. Možemo o njima razmišljati kao o funkcijama ili metodama. One mogu biti pokrenute pomoću okidača, drugih procedura ili aplikacija na Javi, PHP itd.

Prednosti:

* povećavaju performanse aplikacije
* smanjuju promet podataka između baze podataka i aplikacije
* možemo ih koristiti više puta tako što napišemo pozovemo proceduru, ne moramo ju iznova pisati

Nedostatci:

* Spremljene procedure mogu zauzeti puno memorije

Sintaksa sa kreiranje procedure:

CREATE [OR REPLACE] PROCEDURE *ime\_procedure*

[ ( *parametar\_name* [IN | OUT | IN OUT] *type*, [...]) ]

{ IS | AS }

[*blok za deklaraciju*]

BEGIN

*izvršavanje programa*

[EXCEPTION

*blok za iznimke*]

END [*ime\_procedure*];

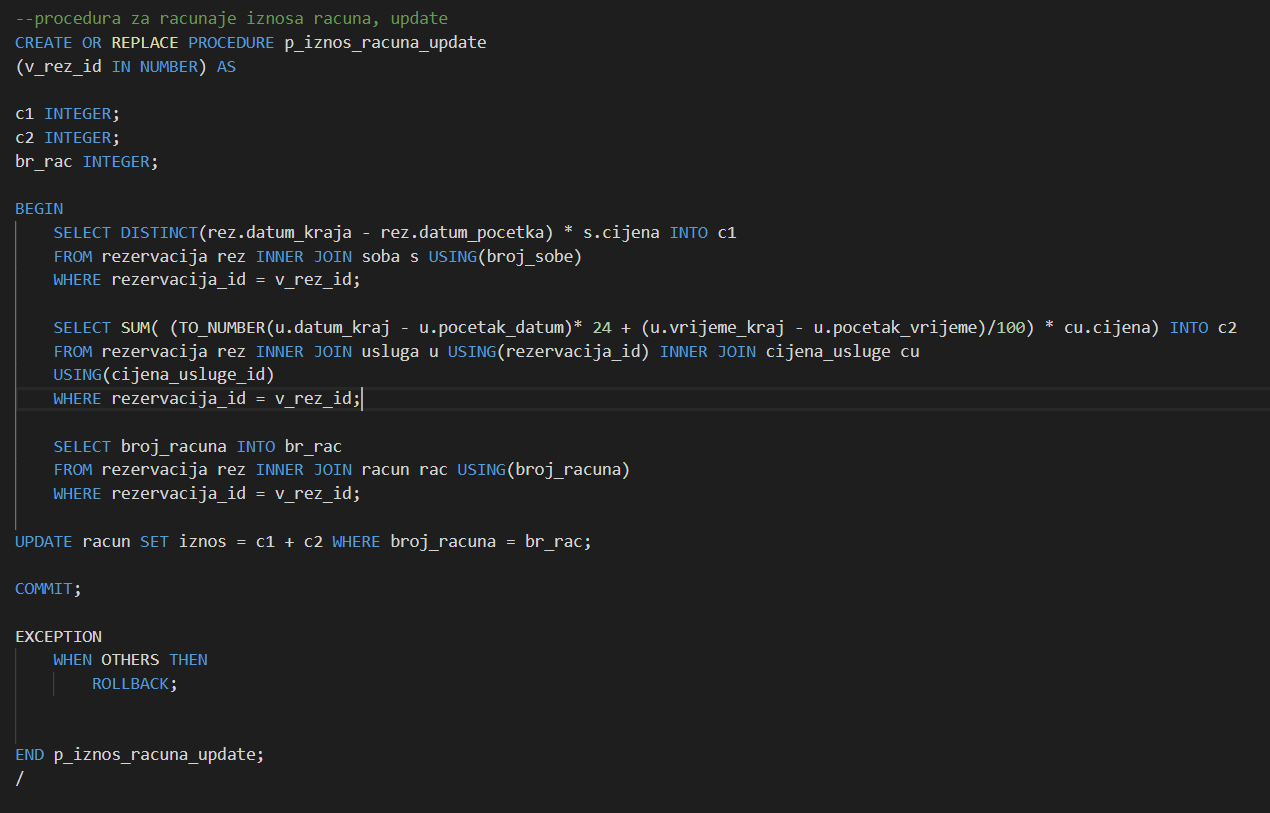
/

Poziv procedure:

CALL *ime\_procedure*( *parametar*, ... );

Brisanje procedure:

DROP PROCEDURE *ime\_procdure*;



**Slika 6.1.** Procedura za ažuriranje iznosa računa

Na slici 6.1. vidimo kreiranje procedure koja računa troškove gosta i ažurira iznos računa.

Procedura prihvaća jednu vrijednost i sprema ju u *v\_rez\_id*

Imamo 3 deklaracije varijabli *c1*, *c2*, i *br\_rac*.

* U *c1* spremamo vrijednost prvog upita koji nam vraća koliki iznos kost treba platiti za najam sobe.
* U *c2* spremamo vrijednost drugog upita koji nam vraća iznos usluga koje je gost koristio.
* U *br\_rac* spremamo vrijednost trećeg upita koji nam vraća broj računa gosta s obzirom na ID rezervacije.
* *v\_rez\_id* pohranjuje id rezervacije za koju računamo iznos računa

Zatim ažuriramo *iznos* u računu koji ima *broj\_racuna* jednak *br\_rac* za vrijednost *c1 + c2.*

Na kraju naredbom COMMIT trajno spremimo promjene u bazu. Ako se dogodi neka greška tijekom izvođenja ove procedure izvršava se EXCEPTION blok koji izvrši ROLLBACK tj. poništi sve promjene koje smo napravili.

**7.** Okidači

Okidači su posebne vrste procedura koje se automatski pokreću na neki događaj, točnije na JMP naredbe ( INSERT, UPDATE ili DELETE ).

Dijelimo ih prema:

* vremenu pokretanja
  + BEFORE
  + AFER
* prema načinu djelovanja
  + ROW-LEVEL – poziva se za svaki redak u tablici čija je modifikacija pokrenula okidač
  + STATEMENT-LEVEL – poziva se jednom prilikom izvršavanja naredbe zbog koje se okidač pokrenuo bez obzira na broj redaka koji će biti promijenjeni

Razlike između ROW i STATEMENT LEVEL okidača:

* ROW-LEVEL okidači mogu pristupiti starim i novim vrijednostima redaka ako su pokrenuti zbog UPDATE naredbe
* izvršavanje ROW-LEVEL okidača može se ograničiti samo na određeni skup redaka koji zadovoljavaju nekakav uvjet

Sintaksa:

CREATE [OR REPLACE] TRIGGER ime\_okidača

{BEFORE | AFTER | INSTEAD OF | FOR} događaj\_okidača

ON ime\_tablice

[FOR EACH ROW]

[{FORWARD | REVERSE} CROSSEDITION]

[{FOLLOWS | PRECEDES} schema.other\_trigger}

[{ENABLE | DISABLE}]

[WHEN uvjet okidača]]

BEGIN

blok\_okidača

END ime\_okidača;

FORWARD – okidač se pokreće prije „okidajuće“ naredbe

AFTER – okidač se pokreće poslije „okidajuće“ naredbe

INSTEAD OF – okidač se pokreće *umjesto* „okidače“ naredbe

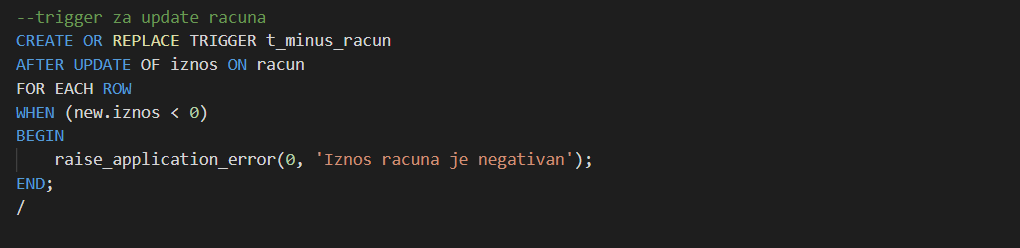
FOR EACH ROW – radi se o ROW-LEVEL okidaču, ako se ispusti ova ključna riječ, onda se radi o

STATEMENT-LEVEL okidaču

{FOLLOWS | PRECEDES} schema.other\_trigger određuje hoće li se okidač pokrenuti prije ili poslije okidača schema.other\_trigger koji se može nalaziti i u nekoj drugoj shemi

{ENABLE | DISABLE} određuje je li okidač inicijalno (prilikom kreiranja) omogućen ili ne.

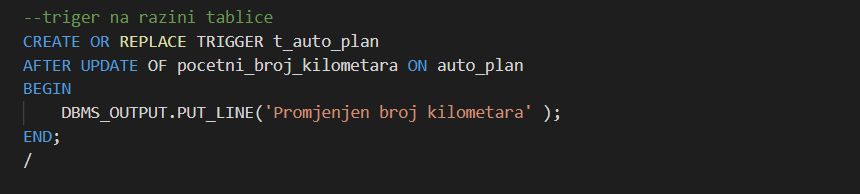
Primjer 7.1. Okidač na razini retka, ROW-LEVEL



**Slika 7.1.** Okidač na razini retka, ROW-LEVEL

Okidač *t\_minus\_racun* pokreće se nakon ažuriranja vrijednosti *iznos* iz tablice *racun* u negativan iznos. Ispisat će nam „Iznos racuna je negativan“ te ažuriranje neće biti napravljeno.

Primjer 7.2. Okidač na razini tablice, STATEMENT-LEVEL



**Slika 7.2.** Okidač na razini tablice, STATEMENT-LEVEL

Okidač *t\_auto\_plan* pokreće se nakon ažuriranja vrijednosti *pocetni\_broj\_kilometara* iz tablice *auto\_plan.* Ispisat će se „Promjenjen broj kilometara“.

**8.** Indeksi

Indeksi nam služe u ubrzavanju pretraživanja pogotovo kad imamo veliki broj različitih podatak spremljenih u našoj bazi podataka. Pogodni su za stupce koji sadrže veliki broj različitih vrijednosti. Njihov način pretraživanja je temeljni da strukturi podataka koju zovemo B-stablo.

Prednosti:

* Ubrzan pristup podacima

Nedostatci:

* Vrijeme potrebno za dodavanje indeksa za svaki redak
* dodatni memorijski prostor potreban za pohranjivanje indeksa

Sintaksa:

CREATE [UNIQUE] INDEX ime\_indeksa ON

ime\_tablice (ime\_stupca[, ime\_stupca ... ])

TABLESPACE tab\_razmak;

UNIQUE – vrijednost u indeksiranom stupcu mora biti jedinstvena

ime\_indeksa – naziv indeksa

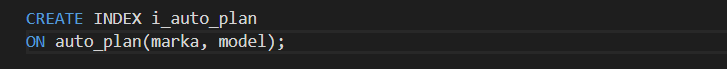
ime\_tablice – naziv tablice u kojoj se nalazi indeksirani stupac

ime\_stupca – naziv stupca koji se indeksira

TABLESPACE – naziv tabličnog prostora u koji se sprema indeks (preporučuje se da bude različit

od tabličnog prostora u koji se spremaju tablice zbog performansi)

Primjer 8.1.



**Slika 8.1.** Indeks

Na slici 8.1. vidimo primjer indeksiranja stupaca *marka* i *model* u tablici *auto\_plan*.

**9.** Zaključak

Cilj ovog seminarskog rada je ponuditi sva potrebna znanja uz primjere za lakše razumijevanja projekta Hotel.

Cilj projekta bio je po prvi puta samostalno napraviti bazu podataka za bilo što u svrhu samousavršavanja vlastitog znanja SQL-a i općenito znanje o bazama podataka. Cjelokupni dojam projekta je pozitivan jer je doista koristan u više segmenata, od organizacije vlastitog vremena do učenja novih stvari.