Sveučilište J. J. Strossmayer u Osijeku
Odjel za Matematiku,
preddiplomski studij Matematika i računarstvo

MODERNI SUSTAVI BAZE PODATAKA

ZAVRŠNI PROJEKT: HOTEL

Profesor:
doc. dr. sc. Slobodan Jelić

Student:
Josip Pavičić

Asistenti:
Ena Pribisalić
Mateja Đumić

Osijek, lipanj 2020.

SADRŽAJ

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.

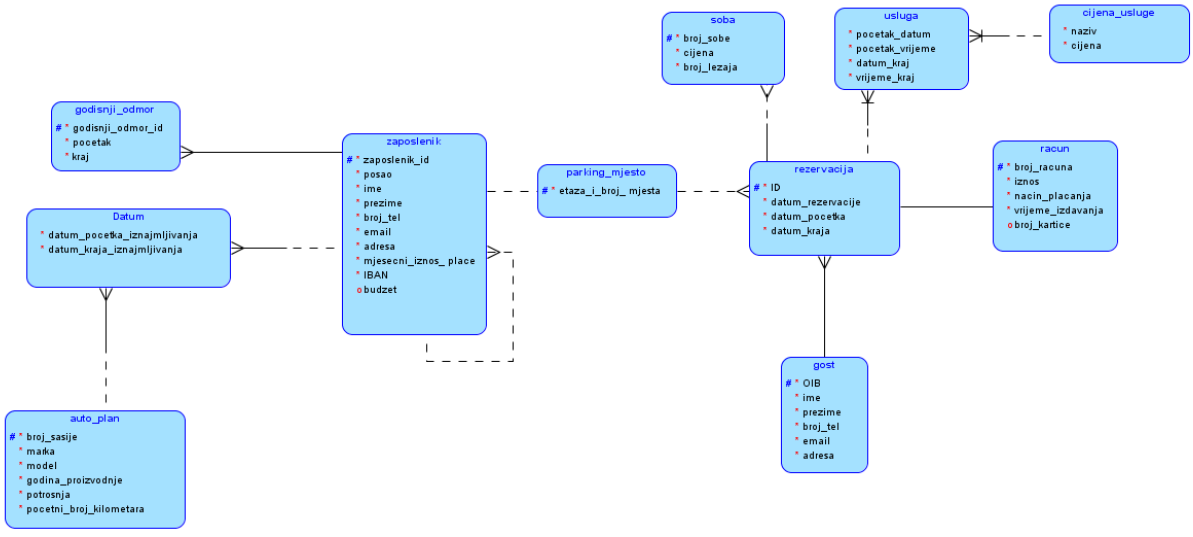
2. MEV(**M**odel **E**ntiteta i **V**eza) i relacijski model

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

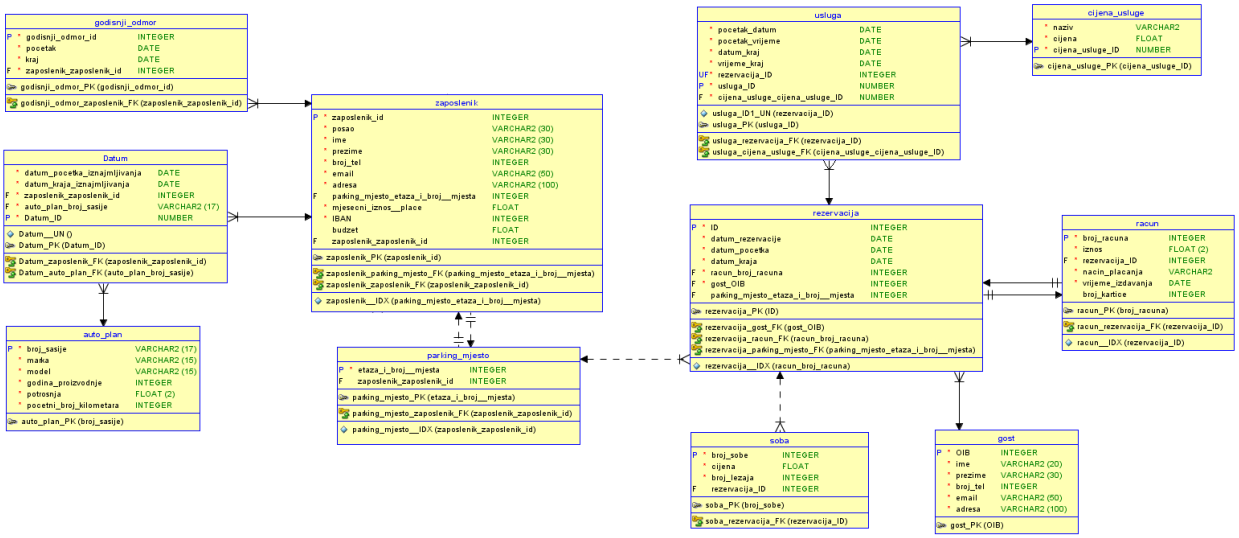
Model hotela sastoji se od 11 entiteta. Entiteti su povezani vezama od kojih su dvije više naprema više veze koje su rastavljene entitetima datum i usluga na četiri jedan naprema više veze, dakle imamo sveukupno devet jedan naprema više veza i dvije jedan naprema jedan veze koje nam govore sljedeće:

- 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.

```
CREATE TABLE rezervacija
(
    rezervacija_id INTEGER CONSTRAINT rezervacija_pk PRIMARY KEY,
    datum_rezervacije DATE NOT NULL,
    datum_pocetka DATE NOT NULL,
    datum_kraja DATE NOT NULL,
    broj_racuna INTEGER NOT NULL CONSTRAINT rezervacija_broj_racuna_fk REFERENCES
        racun(broj_racuna),
    OIB INTEGER NOT NULL CONSTRAINT rezervacija_OIB_fk REFERENCES
        gost(OIB),
    parking_mjesto_id VARCHAR(5) CONSTRAINT rezervacija_parking_mjesto_fk REFERENCES
        parking_mjesto(etaza_i_br_mjesta),
    broj_sobe INTEGER CONSTRAINT rezervacija_broj_sobe_fk REFERENCES
        soba(broj_sobe)
);
```

Slika 3.1. tablica rezervacija

```
CREATE TABLE auto_plan
(
    broj_sasije VARCHAR(17) CONSTRAINT broj_sasije_pk PRIMARY KEY,
    marka VARCHAR(15) NOT NULL,
    model VARCHAR(15) NOT NULL,
    godina_proizvodnje INTEGER NOT NULL,
    potrosnja FLOAT NOT NULL,
    pocetni_broj_kilometara INTEGER NOT NULL
);
```

Slika 3.2. tablica auto_plan

```
CREATE TABLE racun
(
    broj_racuna INTEGER CONSTRAINT broj_racuna_pk PRIMARY KEY,
    iznos FLOAT NOT NULL,
    nacin_placanja VARCHAR(20) NOT NULL,
    vrijeme_izdavanja DATE NOT NULL,
    broj_kartice INTEGER
);
```

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.

```
INSERT INTO datum(datum_id, datum_pocetka_iznajmljivanja, datum_kraja_iznajmljivanja, zaposlenik_id, auto_plan_id)
VALUES (7 , '01-DECEMBER-2017','26-AUGUST-2020', 29 , 'WP0ZZZ99ZTS392124');
```

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.

```
INSERT INTO auto_plan
VALUES ('JHMES86902S205033','Audi','A6', 2020, 10.7, 54150);
```

Slika 4.2. unos vrijdnosti u tablicu auto_plan

5. Upiti

Postoje jednostavni upiti nad jednom tablicom, složeni upiti s više tablica, podupiti i ugnježđeni 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.

```
--ispis potrosenih dana godisnjeg odmora 2020 godine, where
SELECT z.ime, z.prezime, godo.kraj - godo.pocetak AS "Godisnji u 2020", z.zaposlenik_id
FROM zaposlenik z, godisnji_odmor godo
WHERE godo.zaposlenik_id = z.zaposlenik_id
AND godo.pocetak BETWEEN '01-JANUARY-2020' AND '31-DECEMBER-2020';
```

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.

```sql
--podupit, svi zaposlenici s vecom placom od prosjecne
SELECT ime, prezime
FROM zaposlenik
WHERE mjesecni_iznos_place >
(
    SELECT AVG(mjesecni_iznos_place)
    FROM zaposlenik
);
```

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*;

```
--procedura za racunaje iznosa racuna, update
CREATE OR REPLACE PROCEDURE p_iznos_racuna_update
(v_rez_id IN NUMBER) AS

c1 INTEGER;
c2 INTEGER;
br_rac INTEGER;

BEGIN
    SELECT DISTINCT(rez.datum_kraja - rez.datum_pocetka) * s.cijena INTO c1
    FROM rezervacija rez INNER JOIN soba s USING(broj_sobe)
    WHERE rezervacija_id = v_rez_id;

    SELECT SUM( (TO_NUMBER(u.datum_kraj - u.pocetak_datum)* 24 + (u.vrijeme_kraj - u.pocetak_vrijeme)/100) * cu.cijena) INTO c2
    FROM rezervacija rez INNER JOIN usluga u USING(rezervacija_id) INNER JOIN cijena_usluge cu
    USING(cijena_usluge_id)
    WHERE rezervacija_id = v_rez_id;

    SELECT broj_racuna INTO br_rac
    FROM rezervacija rez INNER JOIN racun rac USING(broj_racuna)
    WHERE rezervacija_id = v_rez_id;

UPDATE racun SET iznos = c1 + c2 WHERE broj_racuna = br_rac;

COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;


END p_iznos_racuna_update;
/
```

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

```
--trigger za update racuna
CREATE OR REPLACE TRIGGER t_minus_racun
AFTER UPDATE OF iznos ON racun
FOR EACH ROW
WHEN (new.iznos < 0)
BEGIN
    raise_application_error(0, 'Iznos racuna je negativan');
END;
/
```

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

```
--triger na razini tablice
CREATE OR REPLACE TRIGGER t_auto_plan
AFTER UPDATE OF pocetni_broj_kilometara ON auto_plan
BEGIN
    DBMS_OUTPUT.PUT_LINE('Promjenjen broj kilometara' );
END;
/
```

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.

```
CREATE INDEX i_auto_plan
ON auto_plan(marka, model);
```

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.