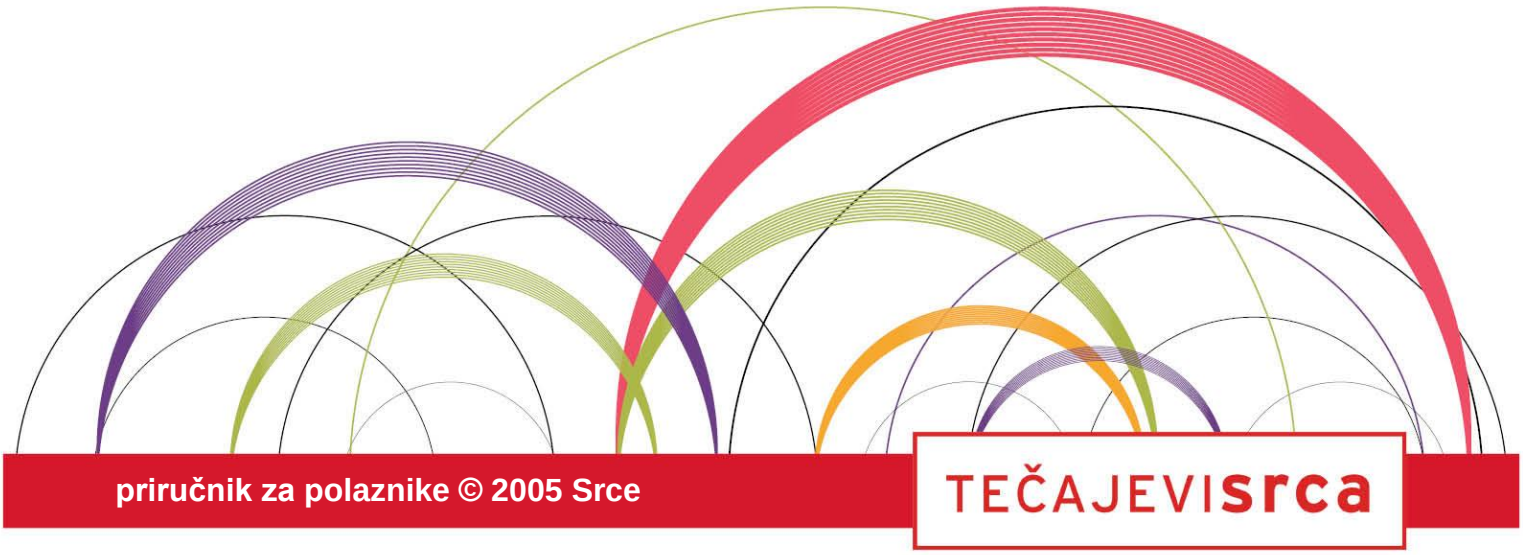
Uvod u SQL

D300



priručnik za polaznike © 2005 Srce

TEČAJEVIsrca

srce
Sveučilište u Zagrebu
Sveučilišni računski centar

Ovu inačicu priručnika izradio je autorski tim Srca u sastavu:

Autor: Mladen Vedriš

Recenzent: Nenad Milanović

Urednica: Sabina Rako

Lektorica: Jasna Novak Milić

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

tecajevi@srce.hr

ISBN: 978-953-7138-42-2 (meki uvez)
ISBN: 978-953-7138-47-9 (PDF)

Verzija priručnika D300-20141203

Sadržaj

UVOD 1

1. OBJEKTI, RELACIJE I BAZE PODATAKA 2

1.1. Sustavi i modeli 2
1.2. Matematički pojam relacije 4
1.3. Relacijski model i relacijska baza podataka 6
1.4. Relacijski upitni jezici 7
1.5. Definiranje relacija 8
1.6. Elementarna selekcija 9
1.7. Dodavanje redaka u relaciju 10
1.8. Selekcija uz klauzulu WHERE 11
1.9. Promjena podataka u relaciji 16
1.10. Stvaranje novih relacija selekcijom 17
1.11. Promjena strukture relacije 19
1.12. Brisanje redaka iz relacije 20
1.13. Uklanjanje relacije iz baze podataka 21
1.14. Pitanja za ponavljanje 21

2. KOMPLEKSNE SELEKCIJE 22

2.1. Logički operatori u klauzuli WHERE 22
2.2. Kondicionalni operatori IN i BETWEEN 24
2.3. Klauzula ORDER BY 26
2.4. Aliasi i skalarne funkcije 28
2.5. Agregatne funkcije 30
2.6. Klazula GROUP BY 32
2.7. Klauzula HAVING 33
2.8. Klauzula COMPUTE 34
2.9. Pitanja za ponavljanje 35

3. OPERACIJE NAD RELACIJAMA 36

3.1. Projekcija i restrikcija 36
3.2. Unija relacija 38
3.3. Presjek i razlika relacija 40
3.4. Descartesov produkt relacija 41
3.5. Spoj relacija po zajedničkom atributu 42
3.6. Definiranje pogleda 45
3.7. Spajanje projekcija i problem gubljenja informacija 46
3.8. Pitanja za ponavljanje 53

4. ELEMENTI TEORIJE NORMALIZACIJE 54

4.1. Definicija funkcionalne zavisnosti 54
4.2. Ponavljajuća skupina i prva normalna forma 57
4.3. Nepotpune funkcionalne zavisnosti i druga normalna forma 59
4.4. Tranzitivne zavisnosti i treća normalna forma 62
4.5. Više normalne forme 64
4.6. Nenormalizirani relacijski model 65
4.7. Problem samogovorećih ključeva 66
4.8. Pitanja za ponavljanje 70

5. OBLIKOVANJE RELACIJSKE BAZE PODATAKA 71

5.1. Model entiteta i veza 71
5.2. Entitetni tipovi 71
5.3. Atributni tipovi 72
5.4. Tipovi veza 73
5.5. Dijagram entiteta i veza 75
5.6. Prevođenje ER modela u relacijski model 76

5.7. DEFINIRANJE INDEKSA 80
5.8. SUSTAVI ZA UPRAVLJANJE RELACIJSKIM BAZAMA PODATAKA 81
5.9. PITANJA ZA PONAVLJANJE 83
DODATAK 1. SINTAKTIČKI ELEMENTI JEZIKA SQL 84
DODATAK 2. FORMIRANJE TABLICA IZ PRIMJERA 87
DODATAK 3. MS SQL SERVER MANAGEMENT STUDIO EXPRESS 92
LITERATURA 95
LINKOVI 95

Uvod

Cilj je tečaja Uvod u SQL omogućiti polazniku da napravi značajan iskorak u svojem poimanju informacijskih sustava zasnovanih na relacijskim bazama podataka te da od korisnika početnika postane napredni korisnik ovih sustava koji razumije suštinske mehanizme njihovog rada i zna se njima služiti.

SQL (*Structured Query Language*) pripada skupini takozvanih neproceduralnih upitnih jezika i služi za dohvat, promjenu, dodavanje i brisanje podataka, kao i za oblikovanje strukture podataka unutar relacijskih baza.

Zamišljeno je da ovaj tečaj bude most između, nazovimo ih tako, akademskih tečajeva, kakvi se predaju na fakultetima, i praktičnih tečajeva, kakvi se mogu slušati u popularnim informatičkim učilištima. Ovdje će se naći mnoštvo primjera kroz koje se uči vještina rada s podacima, ali nisu zanemareni ni strogo formalni koncepti na kojima se sve zasniva, a bez kojih ne može biti pravog razumijevanja ni svladavanja vještine.

Kako bi se sačuvala preciznost i konzistentnost u izlaganju, ponegdje će se rabiti formalno matematičko izražavanje, no na polaznikovo predznanje ili iskustvo s ovakvim konceptima ne postavljaju se nikakvi posebni zahtjevi. Svaka formalna definicija popraćena je i neformalnim objašnjenjem kao i primjerima koje će polaznici zajedno s predavačem prolaziti zajedno s predavačem, učvršćujući tako i proširujući korak po korak svoje znanje i vještinu.

Tečaj je napravljen tako da bude što nezavisniji od samog alata koji se rabi za pristup bazi podataka. Polaznici će tijekom tečaja za praktične primjere rabiti *MS SQL Management Studio Express* za koji su osnovne upute dane u Dodatku 3, ali način na koji će stjecati svoja nova znanja neće ih vezati uz taj alat. Na taj se način želi omogućiti dostizanje potrebne fleksibilnosti i nezavisnosti u odnosu na sustave koji se koriste upitnim jezikom SQL.

1. Objekti, relacije i baze podataka

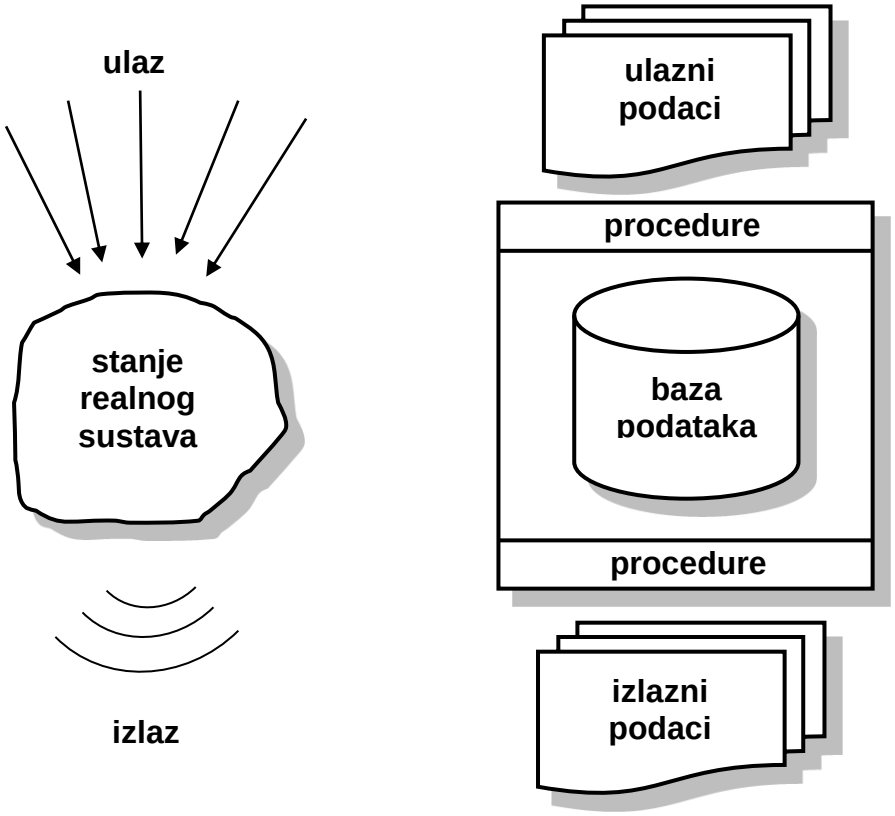
Sažetak

U ovom poglavlju naučit će se osnovni pojmovi o realnim i informacijskim sustavima, bazama podataka i upitnim jeziicima. Naročito će se govoriti o

- odnosu između realnog i informacijskog sustava
- relacijskim bazama podataka i relacijskim upitnim jezicima
- relacijama i njihovom predstavljanju pomoću tablica
- osnovama jezika SQL
- stvaranju i brisanju relacija
- unošenju, pretraživanju i brisanju podataka unutar relacije.

1.1. Sustavi i modeli

Svijet koji nas okružuje, bilo da se radi o vanjskom svijetu materijalnih stvari ili unutrašnjem svijetu apstraktnih pojmova, dostupan je našoj spoznaji putem svojih **objekata** i **procesa** koje naši osjeti, intuicija, misli i osjećaji mogu zapaziti.



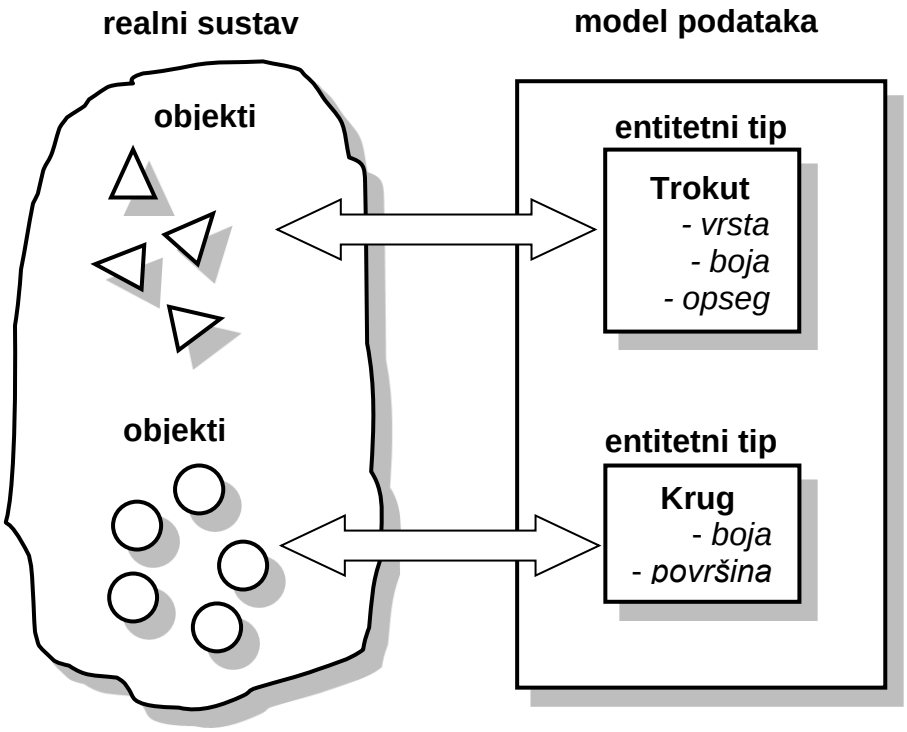
Želimo li sustavno promatrati neku cjelinu svijeta, odnosno, neki **realni sustav**, napravit ćemo **bazu podataka**, organiziranu kolekciju podataka o objektima tog sustava te je snabdjeti programima koji odražavaju njegove procese. Tako dobiveni sustav zvat ćemo **informacijskim sustavom**.

Za izgradnju informacijskog sustava trebat će nam **model podataka**, zamišljena konstrukcija sastavljena od misaonih obrazaca prema kojima ćemo bilježiti podatke o objektima realnog sustava te **model procesa**, skup pravila koja opisuju ponašanje realnog sustava i na temelju kojih možemo definirati procedure, odnosno programe u informacijskom sustavu. Možemo reći da su procedure u informacijskom sustavu slika procesa u realnom sustavu.

Baza podataka odražava stanje realnog sustava i predstavlja jezgru informacijskog sustava. Model podataka na kojem se sustav temelji manje je promjenjiv od modela procesa koji čini podlogu programima koji upravljaju bazom podataka. Zato se prilikom projektiranja informacijskih sustava u pravilu polazi od modela podataka.

Prilikom izgradnje modela podataka tražit ćemo objekte (entitete) realnog sustava koji su međusobno usporedivi po nekim obilježjima. Zamisao, odnosno ideja koja na razini modela podataka predstavlja klasu međusobno usporedivih objekata naziva se **entitetnim tipom**. Na primjer, među objektima na slici uočavamo trokute i krugove koji su predstavljeni entitetnim tipovima **Trokut** i **Krug**.

Svako obilježje (svojstvo) koje se može dodijeliti objektima realnog sustava zove se **atribut**. Na primjer **boja** je atribut nekog objekta. Ideju koja na razini modela podataka predstavlja atribut po kojem uspoređujemo objekte jednog entitetnog tipa zovemo **atributni tip**. Na primjer, atributni tip je **boja** kao svojstvo po kojem uspoređujemo objekte predstavljene entitetnim tipom **Trokut** . Kao što vidimo, razlika između atributa i atributnog tipa vrlo je suptilna i u praksi je uobičajeno izjednačavati ta dva pojma. Reći ćemo, dakle, da entitetni tip **Trokut** kao jedan od svojih atributa ima **boju**.



Skupovi vrijednosti koje pojedini atributi mogu poprimiti nazivaju se **domenama**.

Među raznim objektima postoje i veze. One veze za koje zaključimo da su nam zbog nekog razloga zanimljive predstavit ćemo na razini modela konstrukcijom koju nazivamo **tip veze**.

Entitetne tipove i njihove atribute, kao i tipove veza možemo na razini baze podataka prikazati na različite načine, ali jedan je od najpogodnijih prikaz pomoću matematičkih konstrukcija koje nazivmo relacijama. Bazu podataka koja se zasniva na relacijama zovemo relacijskom bazom podataka.

1.2. Matematički pojam relacije

Promatrajmo, kao primjer, kolekciju raznih geometrijskih likova pa izdvojimo među njima neke trokute i pokušajmo prepoznati njihova obilježja.

Jedno je od obilježja trokuta koje možemo promatrati **vrsta**. Trokuti mogu biti jednakostranični, jednakokračni i raznostranični.

Pretpostavimo da smo naše trokute obojili. Svojstvo po kojem ih sada možemo razlikovati je **boja**. To može biti bilo koja boja iz vidljivog spektra, ali mi ćemo pretpostaviti da smo na raspolaganju imali samo crvenu, zelenu, plavu, žutu, bijelu, narančastu i ljubičastu boju.

Nadalje, trokuti imaju **opseg**. On se izražava realnim brojevima koji predstavljaju neku mjeru duljine (npr. centimetri). Pretpostavimo da želimo promatrati samo trokute čiji se opseg može izraziti cjelobrojnom vrijednošću, dakle brojevima 1, 2, 3 itd.

Zapišimo sada, malo formalnije, skupove iz kojih naši atributi mogu poprimati vrijednosti. Definirat ćemo, dakle, njihove domene.

$$D_{vrsta} = \{\text{jednakostraničan, jednakokračan, raznostraničan}\}$$

$$D_{boja} = \{\text{crvena, zelena, plava, žuta, bijela, narančasta, ljubičasta}\}$$

$$D_{opseg} = \{1, 2, 3, ...\}$$

Primjećujemo da su domene atributa **vrsta** i **boja** konačni skupovi, dok je domena atributa **opseg** beskonačna.

Da bismo opisali neki pojedinačni trokut, moramo navesti njegovu vrstu, boju i opseg, odnosno, možemo se poslužiti uređenim trojkama od kojih prva predstavlja vrstu, druga boju, a treća opseg. Prva će komponenta trojke pripadati, dakle, skupu D_{vrsta}, druga skupu D_{boja}, a treća skupu D_{opseg}.

Skup svih takvih uređenih trojki nazivamo **Descartesovim produktom** navedenih skupova i označavamo kao

$$D_{vrsta} \times D_{boja} \times D_{opseg}$$

Precizno matematički zapisano,

$$D_{vrsta} \times D_{boja} \times D_{opseg} = \{(x_1, x_2, x_3) \mid x_1 \in D_{vrsta}, x_2 \in D_{boja}, x_3 \in D_{opseg}\}$$

Na primjer, uređenom trojkom

(jednakokračan, plava, 3)

opisan je jedan konkretni trokut. Neki drugi opisan je uređenom trojkom

(raznostraničan, crvena, 100).

Zatim možemo imati

(jednakostraničan, bijela, 34)

i, recimo,

(raznostraničan, zelena, 12).

Primijetimo da u našem modelu trokute istog oblika koji su jednako obojeni i imaju isti opseg ne možemo razlikovati! Model je, naime, uvijek bolja ili lošija aproksimacija, nikad točan opis stvarnog svijeta.

Skup koji smo upravo definirali očito je podskup Descartesovog produkta $D_{vrsta} \times D_{boja} \times D_{opseg}$. Kako svaki podskup Descartesovog produkta skupova nazivamo **relacijom**, mi smo zapravo definirali relaciju **Trokut** s atributima **vrsta**, **boja**, **opseg**.

Ova nas razmatranja vode do matematičkih pojmova Descartesova produkta i relacije koje ćemo sada uvesti općenito.

Definicija. Neka su D_1, ..., D_n skupovi. **Descartesovim produktom** $D_1 \times ... \times D_n$, nazivamo skup svih uređenih n-torki $(x_1,..., x_n)$ za koje vrijedi da je x_1 pripada skupu D_1, dok x_2 pripada skupu D_2 i tako dalje. Simbolički,

$$D_1 \times ... \times D_n = \{(x_1,..., x_n) \mid x_1 \in D_1,..., x_n \in D_n\}$$

Svaki podskup R Descartesovog produkta $D_1 \times ... \times D_n$ nazivamo **relacijom** nad skupovima $D_1,...,D_n$. Ako su pritom D_1, ..., D_n domene atributa A_1, ..., A_n, onda kažemo da je R relacija s atributima A_1, ..., A_n i pišemo $R = R(A_1, ..., A_n)$. Izraz $R(A_1, ..., A_n)$ zovemo **relacijskom shemom** relacije R.

U našem slučaju promatrali smo Descartesov produkt

$$D_{vrsta} \times D_{boja} \times D_{opseg}$$

i njegov podskup, dakle, relaciju,

Trokut = {(jednakokračan, plava, 3),
(raznostraničan, crvena, 100),
(jednakostraničan, bijela, 34),
(raznostraničan, zelena, 12)}

Svaki objekt možemo, dakle opisati nekom uređenom n-torkom vrijednosti njegovih atributa. Prema tome, relacije su pogodno sredstvo za opisivanje skupova objekata koji imaju zajednička obilježja, odnosno koji pripadaju istom entitetnom tipu.

Iz načina kako smo zapisali elemente relacije **Trokut** vidljivo je da se oni mogu prikazati i u tablici čiji bi svaki redak predstavljao po jednu n-torku, a svaki stupac po jedan atribut relacije. Takva bi tablica bila sljedećeg oblika:

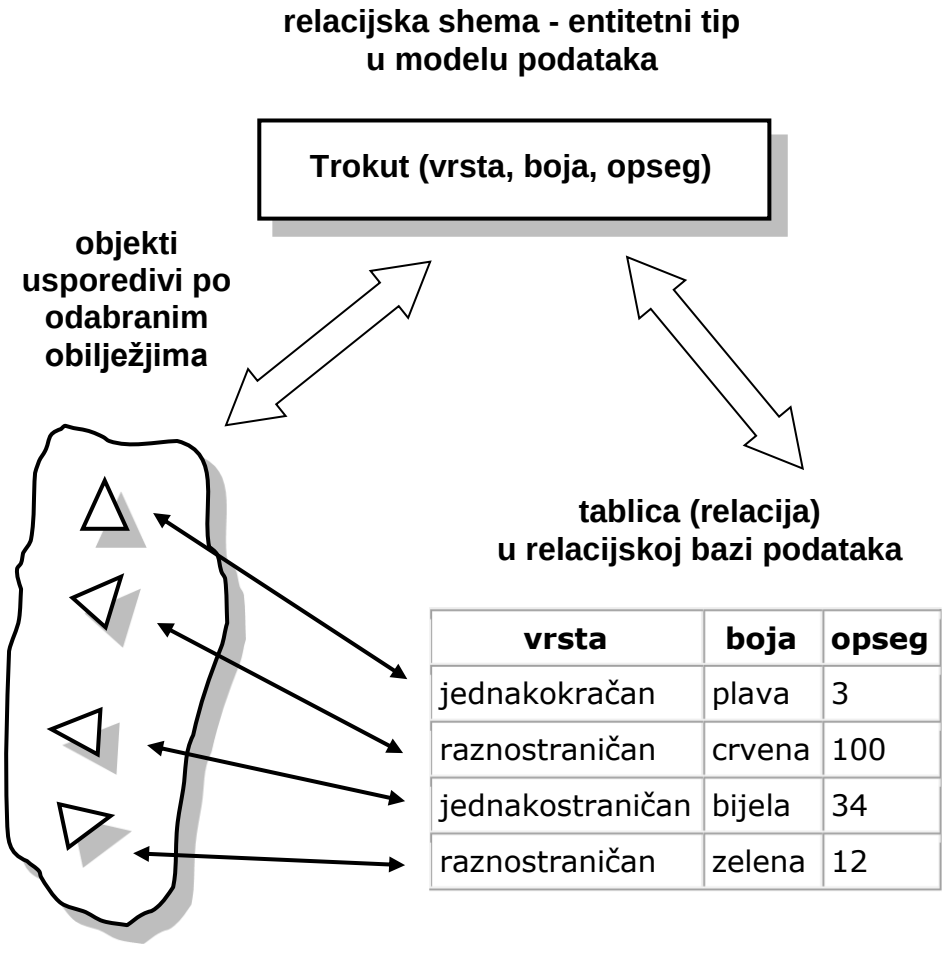
Trokut		
vrsta	**boja**	**opseg**
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

Na taj način tablice možemo u potpunosti izjednačiti s relacijama, a stupce tablica s atributima relacija. Zbog toga u daljnjem izlaganju nećemo ni praviti razlike među njima.

1.3. Relacijski model i relacijska baza podataka

Skup relacija koje opisuju promatrani realni sustav čini relacijsku bazu podataka. Tablicama, odnosno relacijama mogu se, naime, opisati svi potrebni aspekti realnog sustava, dakle, ne samo njegovi objekti, već i veze među njima. O tome će više riječi biti u poglavlju o oblikovanju relacijskih baza podataka. Tablice s pravilno odabranim atributima dovoljne su da bi se opisalo stanje nekog realnog sustava.

Baza podataka je, kao što smo već istaknuli, organizirana kolekcija podataka koja opisuje stanje realnog sustava. Organizacija podataka postiže se uporabom različitih struktura podataka kao što su tablice, popisi, pokazivači, stabla i slično. Relacijska je baza u odnosu na druge vrste baza podataka posebna po tome što su u njoj relacije, odnosno tablice, jedina struktura podataka kojom se služimo.



vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

Svaka tablica unutar relacijske baze podataka predstavlja kolekciju istovrsnih objekata i jednoznačno je određena svojim imenom. Svaki stupac tablice predstavlja neki atribut koji je zajednički svim objektima iz promatrane kolekcije, a svaki redak predstavlja po jedan primjerak objekta iz te kolekcije. Svaki stupac unutar tablice određen je svojim imenom koje je jedinstveno na razini tablice (ali ne mora biti jedinstveno na razini baze podataka) i ima svoj tip podataka, tj. skup vrijednosti koje taj atribut može poprimiti.

Strukturu relacije opisujemo, kao što znamo, njezinom relacijskom shemom. Skup svih relacijskih shema neke relacijske baze podataka možemo nazvati **relacijskim modelom podataka** promatranog sustava.

Klasi objekata realnog sustava koji su usporedivi po odabranim obilježjima odgovara entitetni tip u modelu podataka, odnosno relacijska shema, ako se radi o relacijskom modelu podataka. Pojedinoj vrsti odabranih obilježja (atributa) u realnom sustavu odgovara po jedan atributni tip u modelu podataka.

Svakom entitetnom tipu u modelu podataka, odnosno relacijskoj shemi ako se radi o relacijskom modelu podataka, odgovara jedna tablica (relacija) u relacijskoj bazi podataka. Svakom atributnom tipu promatranog entitetnog tipa u modelu podataka odgovara po jedan stupac u odgovarajućoj tablici relacijske baze podataka.

Svakm pojedinačnom objektu realnog sustava odgovara po jedan redak (n-torka) u odgovarajućoj tablici relacijske baze podataka. Objekte koji su opisani istom n-torkom ne možemo razlikovati. Oni za nas predstavljaju jedinstveni objekt. Zato u tablice koje izjednačavamo s relacijama ne unosimo kopije istih redaka.

Primijetimo da je svejedno kojim redoslijedom navodimo atribute u tablici. Tako je **Trokut(vrsta, boja, opseg)** isto što i **Trokut(boja, vrsta, opseg)**. Isto tako, budući da je relacija skup, svejedno je kojim su su redoslijedom navedene njezine n-torke, dakle poredak redaka u tablici sa stanovišta relacijske teorije nije važan.

1.4. Relacijski upitni jezici

Nad relacijama se, kao što ćemo uskoro vidjeti, mogu definirati razne operacije. Drugim riječima, nad njima se može definirati algebra koju zovemo **relacijska algebra**. Rezultat niza operacija relacijske algebre nad nekim skupom relacija uvijek će biti nova relacija, odnosno skup n-torki.

Jedan je od načina na koji se dolazi do tog rezultata je eksplicitno navođenje potrebnog niza operacija. Taj način nazivamo **proceduralnim**. Proceduralni pristup je zastupljen u klasičnom programiranju, a njegovu suštinu čini zadavanje *postupka* – kako što napraviti.

Drugi je način davanje određene specifikacije koja jednoznačno određuje skup n-torki tražene relacije ne vodeći računa o operacijama koje je potrebno izvršiti. Takav način nazivamo **neproceduralnim**, a njegovu suštinu čini zadavanje *cilja* – što želimo dobiti. U slučaju relacijskih baza podataka, specifikacija rezultata koji se želi dobiti uobičajeno se naziva **relacijskim upitom**. Prevođenje relacijskog upita u odgovarajući niz programskih koraka, odnosno operacija relacijske algebre obavlja se automatski.

Za postavljanje relacijskih upita konstruirani su različiti **relacijski upitni jezici**. Među njima je najrašireniji **SQL – Structured Query Language**. Njegov je prepoznatljivi osnovni upitni blok

```
SELECT <popis atributa>
FROM <popis relacija>
WHERE <uvjet>
```

SQL omogućava pregledno i jednostavno izvođenje složenih operacija relacijske algebre te lagano komuniciranje s različitim relacijskim bazama podataka.

SQL je upitni jezik koji se rabi za komunikaciju s relacijskim bazama podataka. Smatra se standardnim jezikom za sustave upravljanja relacijskim bazama podataka (RDBMS – *Relational Database Management Systems*).

Neki od poznatijih RDBMS-a koji se koriste SQL-om su: *Oracle*, *Sybase*, *Microsoft SQL Server*, *Access*, *Ingres*. Iako većina sustava za upravljanje bazama podataka rabi SQL, većina također ima i svoje vlastite dodatke i proširenja. Međutim, standardne SQL naredbe, kao što su Select, Insert, Update, Delete, Create i Drop mogu pokriti gotovo sve što je potrebno za uobičajeni rad s bazama podataka

1.5. Definiranje relacija

Za stvaranje tablica koristimo se naredbom CREATE TABLE, koja ima ovakvu sintaksu.

CREATE TABLE	CREATE TABLE ime_relacije (ime_atributa1 tip1, ime_atributa2 tip2, ...)

Nakon ključnih riječi CREATE TABLE slijedi ime koje želimo dodijeliti novostvorenoj tablici, a zatim u zagradi popis definicija njezinih stupaca (atributa) pri čemu se uz svaki atribut navodi i njegov tip i eventualna ograničenja. Definicije se unutar popisa standardno odvajaju zarezom, a na kraju svake naredbe u jeziku SQL stavlja se točka-zarez (;).

Imena tablica i njezinih stupaca počinju slovom nakon kojeg mogu slijediti slova, brojke ili podcrte (_). Standardna duljina imena ne smije premašiti 30 znakova. Imena ne mogu biti rezervirane riječi kao što su SELECT, CREATE, KEY i slično. Možemo definirati po volji mnogo stupaca.

Tipovi podataka određuju skupove (domene) iz kojih će pojedini atributi poprimati vrijednosti, odnosno kakvi su podaci dozvoljeni u pojedinim stupcima. Na primjer, ako u nekom stupcu želimo imati tekstovne podatke, tip tog atributa može biti char ili varchar, a ako želimo upisivati brojeve, tip može biti number

U sljedećoj tablici navedene su neki karakteristični tipovi podataka:

char(size)	string (niz znakova) fiksne duljine. Duljina je navedena u zagradama. Maksimalna duljina je 255 byteova.
varchar(size)	string promjenjive duljine. Maksimalna duljina navedena je u zagradama.
int	cjelobrojna vrijednost.
date	datumska vrijednost.
decimal(size,d)	numerička vrijednost s maksimalnim brojem znamenki danim u size, pri čemu je d decimalnih znamenki.

Ograničenja (*constraints*) su pravila koja su povezana s pojedinim stupcima, a odnose se na podatke koji će se unositi u pojedini stupac. Na primjer ograničenje UNIQUE znači da se u tablici ne mogu pojaviti dva retka s jednakim vrijednostima u danom stupcu; NOT NULL znači da se prilikom unošenja podataka ovaj stupac ne može ostaviti praznim; PRIMARY KEY

znači da će za svaki redak (slog) tablice, vrijednost unešena u ovaj stupac predstavljati jedinstveni identifikator tog sloga.

Primjer:

```
create table Trokut
(
vrsta varchar(20) not null,
boja varchar(20),
opseg int
);
```

Sustav će izraditi tablicu **Trokut**,(**vrsta**, **boja**, **opseg**) i odgovoriti porukom koja će, ovisno o razlikama u implementaciji, biti ovakvog oblika:

Trokut created:

vrsta	**boja**	**opseg**

Zadatak. Izradite relaciju **Krug(boja, površina)** s atributima **boja** tipa varchar(20) i **površina** tipa decimal(4,2).

1.6. Elementarna selekcija

Kako bismo mogli vidjeti podatke u našim relacijama, poslužit ćemo se selekcijom. To je osnovni upit u jeziku SQL čija pojednostavnjena sintaksa izgleda ovako:

SELECT	SELECT popis_atributa FROM ime_relacije

SELECT *	SELECT * FROM ime_relacije

Zvjezdicom se skraćeno označava popis svih atributa relacije. Za do sada stvorene relacije **Trokut** i **Krug** mogli bismo postaviti ove upite:

```
select vrsta, boja from Trokut;
select površina from Krug;
select * from Trokut;
```

Kako su relacije za sada prazne, ovi bi upiti dali prazan skup rezultata.

1.7. Dodavanje redaka u relaciju

Za dodavanje redaka u tablicu rabi se naredba INSERT.

Nakon ključnih riječi INSERT INTO upisuje se ime postojeće tablice, a nakon toga se u zagradi navodi popis stupaca u koje pri unosu retka želimo zadati vrijednost. Popis ne mora sadržati sve stupce tablice, već samo odabrane. Među njima se, naravno, moraju nalaziti i oni koji imaju ograničenje NOT NULL.

Nakon zatvorene zagrade navodi se, također u zagradi, popis odgovarajućih vrijednosti za izabrane stupce. Vrijednosti moraju biti navedene onim redoslijedom kojim su navedeni i izabrani stupci.

Naredba se INSERT INTO može koristiti u skraćenom obliku ako se navode vrijednosti svih atributa u poretku kako su definirani u relaciji.

Vrijednosti tekstovnih atributa (stringovi) navode se u jednostrukim navodnicima ('), dok se numeričke vrijednosti upisuju bez navodnika.

Pogledajmo sintaksu:

INSERT INTO	INSERT INTO ime_relacije VALUES (vrijednost1, vrijednost2,....) *ili* INSERT INTO ime_relacije (ime_atributa1, ime_atributa2,...) VALUES (vrijednost1, vrijednost2,....)

Primjeri:

```
insert into Trokut
(vrsta, boja, opseg)
values
('jednakokračan', 'plava', 3)
;
```

Rezultat ove naredbe bit će unos retka o jednakokračnom trokutu plave boje s opsegom 3.

```
insert into Trokut
(vrsta, boja)
values
('raznostraničan', 'crvena')
;
```

Ova će naredba unijeti redak o raznostraničnom trokutu crvene boje s nedefiniranim opsegom.

```
insert into Trokut
(vrsta)
```

```
values
('jednakostraničan')
;
```

Unijeli smo jednakostraničan trokut nedefinirane boje i nedefiniranog opsega.

```
insert into Trokut
(vrsta, boja, opseg)
values
('raznostraničan', 'zelena', 2)
;
```

Unijeli smo raznostraničan trokut zelene boje i opsega 3.

Zadatak. Unesite u relaciju **Krug** retke za tri različita kruga tako da bude

Krug	
boja	**površina**
plava	16.45
crvena	15.30
narančasta	30.22

1.8. Selekcija uz klauzulu WHERE

Naredba SELECT koristi se za postavljanje upita nad relacijskom bazom podataka te za dohvat odabranih podataka koji odgovaraju kriterijima navedenim u upitu. Pogledajmo sintaksu jednostavne SELECT naredbe:

WHERE	SELECT popis_atributa FROM popis_relacija WHERE uvjet

Naredba počinje rezerviranom riječi SELECT nakon koje slijede imena stupaca čije podatke želimo prikazati (projekcija). Ako želimo prikazati sve stupce iz tablice, stavit ćemo znak '*' umjesto popisa stupaca.

Slijedi rezervirana riječ FROM, a iza nje ime promatrane tablice.

Ovako formiranom naredbom SELECT izdvojili bismo sve retke za odabrane stupce tablice. Želimo li se ograničiti samo na neke retke (restrikcija), dodat ćemo i klauzulu WHERE koja sadrži logički uvjet za izdvajanje pojedinih redaka.

Primjer:

```
select * from Trokut;
```

Rezultat ove naredbe bit će popis do sada unesenih redaka tablice **Trokut**:

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	*NULL*
jednakostraničan	*NULL*	*NULL*
raznostraničan	zelena	2

Važno je primijetiti da je rezultat izvođenja naredbe SELECT također tablica. U nekim se implementacijama nedefinirane vrijednosti, takozvane *null* vrijednosti, prikazuju kao praznine, a u nekim se eksplicitno ispisuje NULL ili slična oznaka.

Opcionalna klauzula WHERE, odnosno uvjet koji ona sadrži, određuje koje će vrijednosti podataka ili koji retci biti odabrani ili prikazani kao rezultat izvođenja SELECT naredbe. Time se, dakle, implementira operacija restrikcije. Sintaksa naredbe SELECT dopušta proizvoljne kombinacije restrikcije i projekcije.

Uvjet je izraz koji sadrži operatore uspoređivanja kao što su:

= Jednako
> Veće
< Manje
>= Veće ili jednako
<= Manje ili jednako
<> Različito
LIKE Slično (vidjeti primjer)

Operatori jednakosti (=) i različitosti (<>) primjenjivi su na sve vrste vrijednosti, dok operatori numeričkog uspoređivanja(<, <=, >, >=) imaju smisla samo za numeričke vrijednosti. Pogledajmo, na primjer, ranije spomenutu restrikciju relacije **Trokut** po uvjetu opseg ≥ 2:

```
select * from Trokut
where opseg >= 2;
```

vrsta	boja	opseg
jednakokracan	plava	3
raznostraničan	zelena	2

Slijedi još nekoliko primjera restrikcije.

```
select * from Trokut
where vrsta = 'raznostraničan';
```

vrsta	boja	opseg
raznostraničan	crvena	*NULL*
raznostraničan	zelena	2

Operator pronalaženja uzorka u string atributima LIKE može se koristiti u uvjetnoj selekciji. Izdvojit će one retke koji su „slični“ zadanom uzorku. Znak postotka (%) koristi se kao znak koji stoji umjesto proizvoljnog niza znakova

LIKE	SELECT popis_atributa FROM ime_relacije WHERE ime_atributa LIKE uzorak%

Na primjer:

```
select * from Trokut
where vrsta LIKE 'jednako%';
```

Dobit ćemo podatke za one trokute iz tablice čija vrsta počinje stringom 'Jednako', dakle sve jednakostranične i jednakokračne trokute. Dobit ćemo:

vrsta	boja	opseg
jednakokračan	plava	3
jednakostraničan	*NULL*	*NULL*

Možemo postaviti sljedeći upit:

```
select * from Trokut
where vrsta LIKE '%straničan';
```

Dobit ćemo podatke za trokute čija vrsta završava stringom 'straničan', dakle sve jednakostranične i raznostranične trokute. Imamo:

vrsta	boja	opseg
raznostraničan	crvena	*NULL*
jednakostraničan	*NULL*	*NULL*
raznostraničan	zelena	2

Sljedeći upit izdvojio bi trokute kojima se vrijednost u stupcu boja slaže sa stringom 'crvena':

```
select * from Trokut
where boja = 'crvena';
```

vrsta	boja	opseg
raznostraničan	crvena	*NULL*

Pogledajmo i neke upite koji rade sa *null* vrijednostima.

```
select * from Trokut
where opseg is null;
```

vrsta	boja	opseg
raznostraničan	crvena	*NULL*
jednakostraničan	*NULL*	*NULL*

Dobili smo trokute kojima nije definiran opseg. Treba primijetiti da klasična jednakost ovdje ne bi dala rezultate, jer se ona ne može definirati nad neodređenim vrijednostima. Pogledajmo još upit koji će izdvojiti trokute čiji opseg nije *null*.

```
select * from Trokut
where boja is not null;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	*NULL*
raznostraničan	zelena	2

Navedimo još nekoliko primjera standardnih upita nad relacijom **Trokut**:

```
select boja, opseg from Trokut;

select vrsta, boja, from Trokut
where opseg < 3;

select opseg from trokut
where vrsta LIKE 'r%';

select * from Trokut;

select vrsta, boja, opseg from Trokut
where vrsta LIKE '%kračan';

select * from Trokut
where boja LIKE '%en%';

select * from Trokut
where opseg >= 2;
```

1.9. Promjena podataka u relaciji

Za ažuriranje, odnosno mijenjanje onih redaka koji odgovaraju zadanom kriteriju, koristi se naredba UPDATE. Odabir redaka koje treba promijeniti postiže se prikladno odabranom klauzulom WHERE.

UPDATE	UPDATE ime_relacije SET ime_atributa1=nova_vrijednost1 [, ime_atributa2=nova_vrijednost2, ...] WHERE ime_atributa=neka_vrijednost

Primjeri:

Postavimo opseg crvenog trokuta na 100.

```
update Trokut
set opseg = 100
where boja = 'crvena';
```

Dodijelimo jednakostraničnim trokutima bijelu boju i postavimo im opseg na 34.

```
update Trokut
set boja = 'bijela', opseg=34
where vrsta = 'jednakostraničan';
```

Povećajmo zelenim raznostraničnim trokutima opseg za 10.

```
update Trokut
set opseg = opseg+10
where vrsta='raznostraničan' and boja='zelena';
```

Pregledajmo sada tablicu jednostavnom naredbom SELECT.

```
select * from Trokut;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

1.10. Stvaranje novih relacija selekcijom

Rezultat selekcije uvijek je nova tablica, ali se ona smatra privremenom. Želimo li rezultat selekcije sačuvati kao novu, trajnu relaciju u bazi podataka, koristit ćemo se naredbom za izradu tablice pomoću selekcije.

SELECT INTO (koristi se za stvaranja kopija relacija)	SELECT * INTO ime_nove_relacije FROM ime_izvorne_relacije *ili* SELECT popis_atributa INTO ime_nove_relacije FROM ime_izvorne_relacije

```
select vrsta, boja
into Trokut1
from Trokut;
```

Pogledajmo relaciju **Trokut1** koju smo maloprije izradili.

```
select * from Trokut1;
```

Trokut1	
vrsta	**boja**
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela
raznostraničan	zelena

```
select *
into Krug1
from Krug;
```

Ovom naredbom stvorili smo relaciju **Krug1**. Pogledajmo je.

```
select * from Krug1;
```

Krug1	
boja	**površina**
plava	16.45
crvena	15.30
narančasta	30.22

Neke implementacije jezika SQL omogućavaju izradu novih relacija pomoću naredbe CREATE TABLE ... AS SELECT .... U takvim sustavima napisali bismo:

```
create table Trokut1 as
select * from Trokut;
```

Rezultat ovog upita bio bi isti kao u prethodnom primjeru. No, valja napomenuti da u različitim sustavima ova operacija može biti različito definirana.

1.11. Promjena strukture relacije

Struktura tablice može se mijenjati dodavanjem, mijenjanjem ili brisanjem stupaca te raznim dodavanjima indeksa, ključeva i slično. U tu svrhu koristi se naredba ALTER TABLE s klauzulom

ALTER TABLE (dodavanje atributa)	ALTER TABLE ime_relacije ADD ime_atributa tip

ALTER TABLE (brisanje atributa)	ALTER TABLE ime_relacije DROP COLUMN ime_atributa

```
alter table Trokut1
add površina decimal(4,2);

alter table Trokut1
drop column površina;
```

Neki sustavi dopuštaju i promjenu postojećih atributa pomoću klauzule ALTER COLUMN.

ALTER TABLE (promjena atributa)	ALTER TABLE ime_relacije ALTER COLUMN ime_atributa tip

```
alter table Krug1
alter column površina int;

select * from Krug1;
```

Krug1	
boja	**površina**
plava	16
crvena	15
narančasta	30

Primjećujemo da smo promjenom tipa podataka za atribut površina izgubili informaciju o decimalnim mjestima.

1.12. Brisanje redaka iz relacije

Naredba DELETE koristi se za brisanje zapisa, odnosno redaka iz tablice.

DELETE FROM	DELETE FROM ime_relacije (**Napomena:** brišu se svi retci) *ili* DELETE FROM ime_relacije WHERE uvjet

Napomena: ako se u naredbi DELETE izostavi klauzula WHERE, svi će retci biti izbrisani. Isti učinak imat će naredba TRUNCATE TABLE.

TRUNCATE TABLE (briše se samo sadržaj relacije)	TRUNCATE TABLE ime_relacije

Primjeri:

```
delete from Trokut1
where vrsta = 'raznostraničan';
```

Ova naredba izbrisala je dva retka s podacima o raznostraničnim trokutima.

```
delete from Trokut1
where vrsta = 'jednakokračan' or boja = 'bijela';
```

Ova naredba izbrisala je preostale retke iz relacije **Trokut1**.

```
Truncate table Krug1;
```

Ova naredba izbrisala je sve retke iz relacije **Krug1**. Rezultat bi bio isti da smo izveli naredbu DELETE bez klauzule WHERE.

1.13. Uklanjanje relacije iz baze podataka

Želimo li kompletnu relaciju zajedno s njezinom definicijskom shemom ukloniti iz baze podataka, izvršit ćemo naredbu DROP TABLE.

DROP TABLE	DROP TABLE ime_relacije

Primjer.

```
drop table Trokut1, Krug1;
```

Rezultat je ove naredbe potpuno uklanjanje relacija **Trokut1** i **Krug1** iz baze podataka.

1.14. Pitanja za ponavljanje

1. Što opisuje baza podataka u informacijskom sustavu?
2. Kako nazivamo obilježja koja dodjeljujemo objektima realnog svijeta?
3. U čemu je razlika između proceduralnih i neproceduralnih jezika?
4. Kako izgleda osnovni upitni blok jezika SQL?
5. Kako izgleda naredba za dodavanje redaka u relaciju?
6. Kojom se naredbom ažuriraju vrijednosti atributa u relaciji?
7. Čemu služi operator LIKE?
8. U čemu je razlika između naredbi DELETE FROM i TRUNCATE TABLE?

2. Kompleksne selekcije

Sažetak

Osnovna znanja o pretraživanju podataka proširit ćemo uvođenjem dodatnih tehnika i sintaktičkih mogućnosti koje nam stoje na raspolaganju u jeziku SQL. To su

- logički i kondicionalni operatori
- razvrstavanje (sortiranje) podataka klauzulom ORDER BY
- aliasi, odnosno zamjenska imena relacija i atributa
- računanje raznih vrijednosti pomoću skalarne i agregatne funkcije
- grupiranje podataka klauzulama GROUP BY i HAVING.

2.1. Logički operatori u klauzuli WHERE

U klauzuli WHERE možemo navesti i više od jednog logičkog uvjeta, na način da ih razdvojimo logičkim operatorima konjunkcije (AND) ili disjunkcije (OR). Sintaksa tog razdvajanja izgleda ovako:

AND / OR	SELECT popis_atributa FROM popis_relacija WHERE uvjet AND|OR uvjet

Razdvojimo li dva uvjeta operatorom AND, izdvojit ćemo time one retke iz relacije koji ispunjavaju oba uvjeta. Razdvojimo li ih operatorom OR, izdvojit ćemo retke koji zadovoljavaju bar jedan od navednih uvjeta

```
select * from Trokut
where vrsta = 'raznostraničan'
and boja = 'crvena';
```

vrsta	boja	opseg
raznostraničan	crvena	100

```
select * from Trokut
where vrsta = 'jednakokračan'
or boja = 'crvena';
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100

Uvjet se može negirati operatorom NOT.

```
select * from Trokut
where not (vrsta = 'raznostraničan');
```

vrsta	boja	opseg
jednakokračan	plava	3
jednakostraničan	bijela	34

Logički operatori se mogu kombinirati vodeći računa o pravilu prednosti koje kaže da se najprije izvršava operator NOT, zatim AND pa na kraju OR, kao i u uobičajenom logičkom računu. Želimo li taj prioritet promijeniti, koristit ćemo se zagradama.

```
select * from Trokut
where ((opseg < 10) or (boja = 'crvena'))
and (vrsta = 'raznostraničan');
```

vrsta	boja	opseg
raznostraničan	crvena	100

Kad bismo izostavili zagrade oko disjunkcije, imali bismo

```
select * from Trokut
where (opseg < 10) or
(boja = 'crvena') and (vrsta = 'raznostraničan');
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100

Rezultati su, dakle, različiti.

2.2. Kondicionalni operatori IN i BETWEEN

Operator IN omogućava ispitivanje nalazi li se neka vrijednost u zadanom skupu vrijednosti.

IN	SELECT popis_atributa FROM popis_relacija WHERE ime_atributa IN (vrijednost1,vrijednost2,..)

```
select * from Trokut
where boja in ('bijela', 'zelena');
```

vrsta	boja	opseg
jednakostraničan	bijela	34
raznostraničan	zelena	12

Za selekciju vrijednosti koje su izvan zadanog skupa operator IN možemo kombinirati s logičkim operatorom NOT.

```
select * from Trokut
where boja not in ('bijela', 'zelena');
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100

Većina implementacija jezika SQL omogućava kombiniranje operatora IN s takozvanim **podupitom** (*subquery*). Skup vrijednosti na koje se odnosi operator IN može se, naime, definirati i pomoću selekcije.

```
select * from Trokut
where boja in (
  select boja from Krug
  );
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100

Isti rezultat dobili bismo uz uporabu ključne riječi ANY na sljedeći način:

```
select * from Trokut
where boja = any (
  select boja from Krug
  );
```

Napomena. U nekim sustavima podupiti nisu podržani.

Operator BETWEEN...AND koristi se za ispitivanje nalazi li se neka vrijednost unutar zadanih granica nekog uređenog skupa. Može se, na primjer, zadati da vrijednost bude između neka dva broja, ili, abecedno, između neke dvije riječi.

BETWEEN	SELECT popis_atributa FROM popis_relacija WHERE ime_atributa BETWEEN vrijednost1 AND vrijednost2

```
select * from Trokut
where opseg between 30 and 40;
```

vrsta	boja	opseg
jednakostraničan	bijela	34

```
select * from Trokut
where boja
between 'bijela' and 'plava';
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34

Ovim smo upitom izdvojili one retke za koje je vrijednost atributa **boja** abecedno između (uključeno) riječi 'bijela' i riječi 'plava'. Valja napomenuti da je u različitim implementacijama jezika SQL operator BETWEEN riješen na različite načine u pogledu uključivanja ili isključivanja graničnih vrijednosti. Prije uporabe ovog operatora potrebno je pažljivo ispitati kako se on ponaša u danom sustavu.

Za selekciju vrijednosti koje su izvan zadanih granica možemo operator BETWEEN...AND kombinirati s logičkim operatorom NOT.

```
select * from Trokut
where boja
not between 'bijela' and 'plava';
```

vrsta	boja	opseg
raznostraničan	zelena	12

2.3. Klauzula ORDER BY

Klauzula OREDR BY koristi se za dobivanje ispisa ratvrstanog po zadanim atributima.

ORDER BY	SELECT popis_atributa FROM ime_relacije ORDER BY ime_atributa [ASC|DESC]

Želimo li dobiti podatke o trokutima ispisane po veličini njihovih opsega, od najmanjeg do najvećeg, postavili bismo sljedeći upit:

```
select * from Trokut
order by opseg;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	zelena	12
jednakostraničan	bijela	34
raznostraničan	crvena	100

Razvrstati možemo i po više atributa. Na primjer, želimo li ispisati trokute abecedno po vrstama, a unutar toga po veličini opsega, postupit ćemo ovako:

```
select * from Trokut
order by vrsta, opseg;
```

vrsta	boja	opseg
jednakokračan	plava	3
jednakostraničan	bijela	34
raznostraničan	zelena	12
raznostraničan	crvena	100

Svakom od atributa po kojem želimo razvrstati ispis možemo dodati ključnu riječ ASC ili DESC ako želimo dobiti uzlazni ili silazni poredak. Uzlazni poredak je predpostavljen, pa riječ ASC ne moramo pisati. Ako stavimo DESC, razvrstavanje će se po tom atributu obaviti silazno. Na primjer, želimo ispitati trokute uzlazno abecedno, a unutar toga silazno po opsegu. Imali bismo:

```
select * from Trokut
order by vrsta asc, opseg desc;
```

vrsta	boja	opseg
jednakokračan	plava	3
jednakostraničan	bijela	34
raznostraničan	crvena	100
raznostraničan	zelena	12

2.4. Aliasi i skalarne funkcije

Aliasi su zamjenska imena relacija i atributa koja su pogodna za uporabu u raznim selekcijama. Aliasi će u rezultirajućim tablicama biti prikazani umjesto originalnih imena relacija i atributa.

Pogledajmo najprije kako se aliasi definiraju kao zamjenska imena relacija.

AS (alias za relaciju)	SELECT ime_atributa FROM ime_relacije AS alias_relacije

```
select * from Trokut as Triangle;
```

Triangle		
vrsta	**boja**	**opseg**
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

Nadalje, pogledajmo kako se definiraju aliasi za pojedina imena atributa kako bi se u rezultatima vidjela prilagođena imena stupaca.

AS (alias za atribut)	SELECT ime_atributa AS alias_atributa FROM ime_relacije

```
select
  Trokut.boja as boja,
  Trokut.opseg as opseg_trokuta,
  Krug.površina as površina_kruga
from
  Trokut, Krug
where
  Trokut.boja = Krug.boja;
```

boja	opseg_trokuta	površina_kruga
plava	3	16.45
crvena	100	15.30

Pomoću aliasa možemo pri selekciji definirati nove stupce čije su vrijednosti izračunate iz postojećih. Izračunavanje novih vrijednosti obavlja se pomoću ugrađenih operatora, odnosno funkcija koje nazivamo **skalarnim** funkcijama. Naziv im dolazi otuda što operiraju nad pojedinačnim vrijednostima atributa (za razliku od takozvanih **agregatnih** funkcija koje operiraju nad cijelim skupovima vrijednosti atributa).

```
select
  vrsta, boja, opseg,
  opseg * 2 as opseg_puta_2
from Trokut;
```

vrsta	boja	opseg	opseg_puta_2
jednakokračan	plava	3	6
raznostraničan	crvena	100	200
jednakostraničan	bijela	34	68
raznostraničan	zelena	12	24

Osim matematičkih operatora, koji se primjenjuju na numeričkim atributima, na raspolaganju su nam i operatori koji se primjenjuju na tekstovnim atributima, na primjer operator konkatenacije (+).

```
select
  vrsta + ' ' + boja as vrsta_boja
from Trokut;
```

vrsta_boja
jednakokračan plava
raznostraničan crvena
jednakostraničan bijela
raznostraničan zelena

Želimo li raditi s atributima različitih tipova, moramo se pobrinuti da izvršimo pravilnu konverziju kojom ćemo ih svesti na isti tip.

Napomena. Funkcije za konverziju, kao i ostale ugrađene funkcije, mogu se razlikovati od sustava do sustava.

```
select
  vrsta + ' ' +
  boja + ' ' +
  convert(varchar, opseg) as vrsta_boja_opseg
from Trokut;
```

vrsta_boja_opseg
jednakokračan plava 3
raznostraničan crvena 100
jednakostraničan bijela 34
raznostraničan zelena 12

2.5. Agregatne funkcije

Agregatne funkcije operiraju nad cijelim skupom vrijednosti u stupcu koji pripada danom atributu (za razliku od skalarnih, koje operiraju nad pojedinačnim vrijednostima).

Skup agregatnih funkcija jako se razlikuje od implementacije do implementacije, ali standardno svi sustavi imaju funkcije SUM, AVG, COUNT, MAX, MIN. Neki sustavi dopuštaju funkcije FIRST, LAST i druge.

Promotrimo opet relaciju **Trokut**(**vrsta**, **boja**, **opseg**) i razne funkcije koje možemo definirati nad njenim atributima.

```
select * from Trokut;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

SUM(ime_atributa) vraća zbroj vrijednosti u stupcu koji pripada danom atributu.

AVG(ime_atributa) vraća prosjek vrijednosti u stupcu koji pripada danom atributu

```
select sum(opseg) from Trokut;
```

sum(opseg)
149

```
select avg(opseg) from Trokut;
```

avg(opseg)
37

Treba primijetiti da je prosjek izračunat kao cijeli broj 37, umjesto kao realni broj 37.25, jer je atribut **opseg** definiran kao cijeli broj.

COUNT(ime_atributa) vraća broj redaka u relaciji za koje je vrijednost danog atributa različita od NULL.

COUNT(*) vraća broj redaka u relaciji.

```
select count(*) from Trokut;
```

count(*)
4

MAX(ime_atributa) vraća najveću vrijednost u stupcu koji pripada danom atributu.

MIN(ime_atributa) vraća najmanju vrijednost u stupcu koji pripada danom atributu.

```
select max(opseg) from Trokut;
```

max(opseg)
100

```
select min(opseg) from Trokut;
```

min(opseg)
3

FIRST(ime_atributa) vraća prvu vrijednost u stupcu koji pripada danom atributu.

LAST(ime_atributa) vraća posljednju vrijednost u stupcu koji pripada danom atributu.

Napomena. Funkcije FIRST i LAST nisu dostupne u svim sustavima.

```
select first(vrsta) from Trokut;
```

first(vrsta)
jednakokračan

```
select last(vrsta) from Trokut;
```

last(vrsta)
raznostraničan

2.6. Klazula GROUP BY

Klauzula GROUP BY služi kako bi agregatne funkcije poput SUM() ili AVERAGE() mogle djelovati i na pojedinim dijelovima stupca tablice, a ne uvijek na cijeloj tablici.

GROUP BY	SELECT ime_atributa1,SUM(ime_atributa2) FROM ime_relacije GROUP BY ime_atributa1

Na primjer, ako želimo doznati sume opsega za svaku pojedinu vrstu trokuta, uporabit ćemo upravo ovu klauzulu.

```
select vrsta, sum(opseg)
from Trokut
group by vrsta;
```

vrsta	sum(opseg)
jednakokračan	3
raznostraničan	112
jednakostraničan	34

Ovim upitom trokuti su grupirani po vrsti, a mjere njihovih opsega zbrojene za svaku pojedinu vrstu. Za dva raznostranična trokuta dobili smo zbroj opsega 112, jer je jedan od njih imao opseg 100, a drugi 12. Za ostale vrste (jednakokračan i jednakostraničan) relacija sadrži samo po jedan primjerak, pa su zbrojevi njihovih opsega jednaki pojedinačnim opsezima.

Primijetimo da bez klauzule GROUP BY upit ovog oblika ne bi dao ispravno rješenje, jer bi funkcija **sum**(**opseg**) djelovala preko cijelog stupca opseg. Imali bismo u svakom retku **sum**(**opseg**) jednako 149, što je suma opsega svih trokuta u relaciji.

```
select vrsta, sum(vrsta)
from Trokut
```

vrsta	sum(opseg)
jednakokračan	149
raznostraničan	149
jednakostraničan	149
raznostraničan	149

Napomena. Neki sustavi ne dopuštaju izvršavanje ovako napisane agregatne funkcije bez klauzule GROUP BY.

2.7. Klauzula HAVING

U klauzuli WHERE ne mogu se pojavljivati agregatne funkcije. Zbog toga je jeziku SQL dodana klauzula HAVING. Pogledajmo njezinu sintaksu, pa onda primjer.

HAVING	SELECT ime_atributa1,SUM(ime_atributa2) FROM ime_relacije GROUP BY ime_atributa1 HAVING SUM(ime_atributa2) uvjet vrijednost

Na primjer, želimo dobiti sve vrste trokuta za koje je suma opsega pojedinačnih trokuta u relaciji veća od 30.

```
select vrsta, sum(opseg)
from Trokut
group by vrsta
having sum(opseg) > 30;
```

vrsta	sum(opseg)
raznostraničan	112
jednakostraničan	34

2.8. Klauzula COMPUTE

Uloga je klauzula COMPUTE i COMPUTE...BY izračunavanje agregatnih funkcija na kraju uobičajenog ispisa odabranih vrijednosti. Najčešće se koriste za računanje sume vrijednosti u ispisanom stupcu te međusuma po skupinama vrijednosti.

Napomena. COMPUTE i COMPUTE...BY nisu dio standardnog jezika SQL. Većina sustava ih ipak implementira, ali uz male razlike.

Klauzula COMPUTE na kraju pojedinačnih ispisa računa zadanu agregatnu funkciju.

```
select vrsta, boja, opseg
from Trokut
order by vrsta
compute sum(opseg);
```

vrsta	boja	opseg
jednakokračan	plava	3
jednakostraničan	bijela	34
raznostraničan	crvena	100
raznostraničan	zelena	12
sum(opseg) = 149		

Klauzula COMPUTE...BY računa međurezultate agregatne funkcije.

```
select vrsta, boja, opseg
from Trokut
order by vrsta
compute sum(opseg) by vrsta
compute sum(opseg);
```

vrsta	boja	opseg
jednakokračan	plava	3
sum(opseg) = 3		
jednakostraničan	bijela	34
sum(opseg) = 34		
raznostraničan	crvena	100
raznostraničan	zelena	12
sum(opseg) = 112		
sum(opseg) = 149		

2.9. Pitanja za ponavljanje

1 Navedite osnovne logičke operatore.
2 Navedite osnovne kondicionalne operatore.
3 Što se postiže uvođenjem klauzule ORDER BY?
4 Čemu služi klauzula GROUP BY?
5 Zašto je u sintaksu jezika SQL uvedena klauzula HAVING?

3. Operacije nad relacijama

Sažetak

Tehnike selekcije podataka koje smo do sada naučili operirale su nad jednom relacijom. Da bismo mogli razumjeti upite nad više relacija, bit će nam potrebni neki pojmovi iz relacijske algebre. Naučit ćemo kako se u jeziku SQL obavljaju ove operacije nad relacijama:

- projekcija i restrikcija relacije po zadanim atributima i zadanim uvjetima
- unija, presjek i razlika relacija
- Descartesov produkt relacija
- različite vrste spajanja relacija po zajedničkom atributu

3.1. Projekcija i restrikcija

Relacije, odnosno tablice su skupovi pa se nad njima mogu obavljati sve skupovne operacije. No, relacije su također posebna vrsta skupova pa se nad njima mogu obavljati i neke dodatne operacije.

Osnovne operacije koje se mogu obaviti nad jednom relacijom su **projekcija** i **restrikcija**. Projekcija je izdvajanje odabranih stupaca tablice uz izbacivanje vrijednosti koje se ponavljaju, dok je restrikcija izdvajanje pojedinih redaka tablice po zadanom kriteriju. Uobičajene su oznake za projekciju i restrikciju grčka slova π i σ.

Primjer. Projekcija relacije **Trokut** po atributu **vrsta** označava se kao

$$\pi_{\mathbf{vrsta}}(\mathbf{Trokut})$$

i predstavlja skup svih vrijednosti stupca **vrsta** koje se mogu pronaći u relaciji **Trokut**. Strogo formalno, imali bismo

$$\pi_{\mathbf{vrsta}}(\mathbf{Trokut}) = \{x \in D_{vrsta} \mid \exists\ (x_{boja}, x_{opseg})\ [(x, x_{boja}, x_{opseg}) \in \mathbf{Trokut}]\ \}$$

U jeziku SQL ovaj bismo rezultat dobili pomoću upita SELECT DISTINCT koji daje popis jedinstvenih vrijednosti. Sintaksa je ovakvog upita:

SELECT DISTINCT	SELECT DISTINCT popis_atributa FROM ime_relacije

```
select distinct vrsta
from Trokut;
```

vrsta
jednakokračan
raznostraničan
jednakostraničan

Ako želimo dobiti popis svih redaka, pa i onih koji se ponavljaju, izostavit ćemo kvalifikaciju DISTINCT.

```
select vrsta
from Trokut;
```

vrsta
jednakokračan
raznostraničan
jednakostraničan
raznostraničan

Ovo je, dakle, samo popis redaka, ali ne i projekcija u smislu navedene stroge definicije.

Dodamo li naredbi SELECT opcionalnu klauzulu WHERE, odnosno uvjet koji ona sadrži, odredit ćemo koje će vrijednosti podataka ili koji retci biti odabrani ili prikazani kao rezultat izvođenja naredbe SELECT.

Primjer. Restrikcija relacije **Trokut** po uvjetu **opseg** ≥ 30, oznaka

$$\sigma_{\textbf{opseg} \geq 30}(\textbf{Trokut})$$

je skup svih redaka relacije **Trokut** za koje je vrijednost u stupcu **opseg** veća ili jednaka 2. Strogo formalno,

$$\sigma_{\textbf{opseg} \geq 30}(\textbf{Trokut}) = \{(x_{vrsta}, x_{boja}, x_{opseg}) \in \textbf{Trokut} \mid x_{opseg} \geq 30\}$$

U jeziku SQL ovaj bismo rezultat dobili pomoću sljedećeg upita:

```
select * from Trokut
where opseg >= 30;
```

vrsta	boja	opseg
raznostraničan	crvena	100
jednakostraničan	bijela	34

Definirajmo sada općenito operacije projekcije i restrikcije.

Definicija. Neka je $R = R(A_1, ..., A_n)$ relacija, a X neki podskup od $\{A_1, ..., A_n\}$. Možemo bez gubitka općenitosti pretpostaviti da je to prvih k atributa, odnosno da je $X = \{A_1, ...,A_k\}$, tako da je $0 < k \leq n$. Definirajmo relaciju R' ovako:

$$R' = \{(x_1, ..., k_k) \mid \exists (x_{k+1}, ..., k_n) [(x_1, ..., k_n) \in R] \}$$

Relaciju R' tada zovemo **projekcijom** relacije R po atributima iz X i pišemo $R' = \pi_X(R)$.

Definicija. Neka je $R = R(A_1, ..., A_n)$ relacija, a P predikat koji određuje neki podskup R' od R, tj.

$$R' = \{(x_1,...,x_n) \in R \mid P(x_1,...,x_n)\}$$

Tada relaciju R' zovemo **restrikcijom** relacije R po uvjetu P i pišemo

$$R' = \sigma_P(R).$$

3.2. Unija relacija

Relacije su skupovi pa se za njih mogu definirati uobičajene skupovne operacije. Pogledajmo najprije uniju relacija.

Unija dviju relacija definira se kao skupovna unija njihovih redaka, pri čemu se podrazumijeva da obje relacije imaju isti broj atributa i da su domene odgovarajućih atributa jednake. Svaki redak unije dviju relacija pojavljuje se, dakle, ili u prvoj ili u drugoj relaciji, a možda i u obje.

UNION	SELECT popis_atributa FROM popis_relacija WHERE uvjet UNION [ALL]

```
SELECT popis_atributa
FROM popis_relacija
WHERE uvjet
```

Napravimo ponovo kopiju relacije **Trokut** kao relaciju **Trokut1**.

```
select * into Trokut1 from Trokut;
```

Promijenimo sada zadnja dva retka relacije **Trokut1** i napravimo uniju te relacije s izvornom relacijom **Trokut**.

```
update Trokut1
set vrsta = 'raznostraničan', boja = 'ljubičasta', opseg = 300
where boja = 'bijela';

update Trokut1
set vrsta = 'jednakokračan', boja = 'narančasta', opseg = 400
where boja = 'zelena';

select * from Trokut
union
select * from Trokut1;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12
raznostraničan	ljubičasta	300
jednakokračan	narančasta	400

Primjećujemo da se retci koji su isti u relacijama **Trokut** i **Trokut1** pojavljuju u njihovoj uniji samo jednom. Ako bismo htjeli dobiti i te retke, odnosno imati tablicu u kojoj se pojedini retci ponavljaju, koristili bismo se operatorom union all.

```
select * from Trokut
union all
select * from Trokut1;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12
jednakokračan	plava	3
raznostraničan	crvena	100
raznostraničan	ljubičasta	300
jednakokračan	narančasta	400

Zasjenjeni su retci koji se ne pojavljuju u običnoj skupovnoj uniji.

Definirajmo na kraju i strogo formalno uniju dviju relacija.

Definicija. Neka su $P(A_1,...,A_n)$ i $Q(B_1,...,B_n)$ relacije i neka su domene $D(A_i) = D(B_i)$ za $1 \leq i \leq n$. Definirajmo relaciju R ovako:

$$R = \{(x_1,...,x_n) \mid (x_1,...,x_n) \in P \text{ ili } (x_1,...,x_n) \in Q \}$$

Tada relaciju R zovemo **unijom** relacija P i Q.

3.3. Presjek i razlika relacija

Presjek, razlika i dijeljenje također su uobičajene skupovne operacije, no nisu ugrađene u sve implementacije jezika SQL. Tamo gdje jesu, obično se zadaju na isti način kao unija relacija.

Pogledajmo kako bi se računali presjek i razlika za naše relacije **Trokut** i **Trokut1** u sustavima koji dopuštaju te operacije. Neformalno, presjek čine oni retci koji se nalaze u obje relacije. Razliku čine oni retci iz prve relacije koji se ne nalaze u drugoj relaciji.

```
select * from Trokut
intersect
select * from Trokut1;
```

vrsta	boja	opseg
jednakokračan	plava	3
raznostraničan	crvena	100

```
select * from Trokut
minus
select * from Trokut1;
```

vrsta	boja	opseg
jednakostraničan	bijela	34
raznostraničan	zelena	12

Dobiveni rezultati mogu se dobiti i na drugi način ako implementacija jezika SQL ne podržava ove klasične skupovne operacije. O tome će biti riječi u kasnijim poglavljima koja se bave povezivanjem relacija.

Definirajmo na kraju i strogo formalno presjek i razliku dviju relacija.

Definicija. Neka su $P(A_1,...,A_n)$ i $Q(B_1,...,B_n)$ relacije i neka su domene $D(A_i) = D(B_i)$ za $1 \leq i \leq n$. Definirajmo relacije R i S ovako:

$$R = \{(x_1,...,x_n) \mid (x_1,...,x_n) \in P \text{ i } (x_1,...,x_n) \in Q \}$$

$$S = \{(x_1,...,x_n) \mid (x_1,...,x_n) \in P \text{ i } (x_1,...,x_n) \text{ nije } \in Q \}$$

Tada relaciju R zovemo **presjekom**, a relaciju S **razlikom** relacija P i Q.

3.4. Descartesov produkt relacija

Operacije projekcije i restrikcije definirane su, kao što smo vidjeli, nad jednom relacijom. Upoznajmo sada dvije standardne operacije koje se mogu obavljati nad dvije ili više relacija. To su **Descartesov produkt** i **spoj** relacija. Descartesov produkt skupova upoznali smo već prilikom definiranja pojma relacije. Kako su i relacije skupovi, za njih se također može promatrati Descartesov produkt.

Neformalno, Descartesov produkt dviju relacija je povezivanje njihovih redaka, svaki sa svakim. Za naše relacije **Trokut(vrsta, boja, opseg)** i **Krug(boja, površina)** Descartesov produkt bila bi relacija **Trokut × Krug** s atributima **vrsta**, **boja** i **opseg** iz relacije **Trokut**, te **boja** i **površina** iz relacije **Krug**, pri čemu je prvi redak relacije **Trokut** uparen sa sva tri retka relacije **Krug**, zatim je drugi redak relacije **Trokut** uparen sa sva tri retka relacije **Krug** i tako dalje. Strogo formalno,

$$\begin{aligned} \text{Trokut} \times \text{Krug} = \{ & (x_{vrsta}, x_{boja}, x_{opseg}, y_{boja}, y_{površina}) \mid \\ & (x_{vrsta}, x_{boja}, x_{opseg}) \in \text{Trokut i} \\ & (y_{boja}, y_{površina}) \in \text{Krug} \} \end{aligned}$$

Relacija **Trokut** ima 4 retka, a relacija **Krug** 3, dok relacija **Trokut × Krug** ima 3 × 4 = 12 redaka. Da bismo razlikovali istoimene atribute iz različitih relacija pisat ćemo **Trokut.boja** za atribut **boja** iz relacije **Trokut**, a **Krug.boja** za atribut **boja** iz relacije **Krug** i tako dalje. Descartesov produkt naših relacija dobili bismo pomoću naredbe

```
select *
from Trokut, Krug;
```

Trokut × Krug				
Trokut.vrsta	**Trokut.boja**	**Trokut.opseg**	**Krug.boja**	**Krug.površina**
jednakokračan	plava	3	plava	16.45
jednakokračan	plava	3	crvena	15.30
jednakokračan	plava	3	narančasta	30.22
raznostraničan	crvena	100	plava	16.45
raznostraničan	crvena	100	crvena	15.30
raznostraničan	crvena	100	narančasta	30.22
jednakostraničan	bijela	34	plava	16.45
jednakostraničan	bijela	34	crvena	15.30
jednakostraničan	bijela	34	narančasta	30.22
raznostraničan	zelena	12	plava	16.45
raznostraničan	zelena	12	crvena	15.30
raznostraničan	zelena	12	narančasta	30.22

Općenito, imamo sljedeću definiciju Descartesovog produkta relacija.

Definicija. Neka su $P(A_1,...,A_n)$ i $Q(B_1,...,B_m)$ relacije. Definirajmo relaciju S ovako:

$$S = \{(x_1,...,x_n, y_1,...,y_m) \mid (x_1,...,x_n) \in P \text{ i } (y_1,...,y_m) \in Q \}$$

Relaciju S tada zovemo Descartesovim produktom relacija P i Q i pišemo $S = P \times Q$.

3.5. Spoj relacija po zajedničkom atributu

U tablici koja prikazuje Descartesov produkt istaknuli smo neke stupce i neke retke. Naime, istaknuli smo stupce koji prikazuju atribut **boja**, zajednički za obje relacije. Taj atribut nema samo isto ime u obje relacije, nego i istu domenu, odnosno tip vrijednosti. Zato ga možemo nazvati **zajedničkim atributom** promatranih relacija.

Nadalje, istaknuli smo i one retke za koje je vrijednost zajedničkog atributa **boja** jednaka. Radi se o retcima koji predstavljaju plavi trokut i plavi krug te crveni trokut i crveni krug. Ta dva retka formiraju novu relaciju koju nazivamo **spoj** relacija **Trokut** i **Krug** po zajedničkom atributu **boja** i označavamo je sa

$$\text{Trokut} \blacktriangleright\blacktriangleleft_{\text{boja}} \text{Krug}$$

U jeziku SQL spoj relacija **Trokut** i **Krug** po zajedničkom atributu **boja** dobili bismo naredbom:

```
select *
from Trokut, Krug
where Trokut.boja = Krug.boja;
```

Primijetimo da smo u klauzuli WHERE dodavanjem imena odgovarajućih relacija ispred imena atributa **boja** izbjegli dvoznačnost koja bi nastala zbog istog imena atributa u obje relacije. Imenima atributa i inače možemo, ako želimo, dodati i ime relacije kojoj pripadaju.

Trokut.vrsta	Trokut.boja	Trokut.opseg	Krug.boja	Krug.površina
jednakokračan	plava	3	plava	16.45
raznostraničan	crvena	100	crvena	15.30

Formalnim jezikom rečeno, imali bismo

$$\text{Trokut} \blacktriangleright\blacktriangleleft_{\text{boja}} \text{Krug} = \{ (x_{\text{vrsta}},x_{\text{boja}},x_{\text{opseg}},y_{\text{boja}},y_{\text{površina}}) \mid (x_{\text{vrsta}},x_{\text{boja}},x_{\text{opseg}}) \in \text{Trokut i } (y_{\text{boja}},y_{\text{površina}}) \in \text{Krug i } x_{\text{boja}} = y_{\text{boja}} \}$$

Općenito, imali bismo ovu definiciju spoja:

Definicija. Neka su $P(A_1,...,A_n)$ i $Q(B_1,...,B_n)$ relacije i neka je X neki zajednički podskup od $\{ A_1,...,A_n \}$ i $\{ B_1,...,B_n \}$. Možemo bez gubitka općenitosti pretpostaviti da su elementi od X prvih k atributa u svakoj od relacija. Definirajmo relaciju S ovako:

$$S = \{(x_1,...,x_k, x_{k+1},...,x_n, y_1,...,y_k, y_{k+1},...,y_m) \mid (x_1,...,x_k, x_{k+1},...,x_n) \in P \text{ i } (y_1,...,y_k, y_{k+1},...,y_m) \in Q \text{ i } x_1.=.y_1, ..., x_k.=.y_k \}$$

Relaciju S tada zovemo **spojem** relacija P i Q po zajedničkim atributima iz X i pišemo

$$S = P \blacktriangleright\blacktriangleleft_X Q.$$

Ovo, međutim, nije jedini način na koji možemo definirati spoj relacija po zajedničkom atributu. Postoje još tri načina na koje se spoj može definirati. Zbog praktične važnosti različitih operacija spajanja relacija uvedeni su i sintaktički oblici za svaki od njih. Spajanje koje smo upravo opisali nazivamo također i unutrašnji spoj i za njega je uvedena posebna konstrukcija JOIN...ON. Prethodni upit možemo napisati ovako:

```
select *
from Trokut join Krug
on Trokut.boja = Krug.boja;
```

Postoje i dvije vrste vanjskih spojeva (outer join). Nazivamo ih lijevi i desni vanjski spoj – **left join** i **right join**. Osim njih postoji i tzv. puni spoj – **full join**. Svi su oni različiti podskupovi Descartesovog produkta relacija kojima je dodan i redak s nedefiniranim vrijednostima svih atributa.

Lijevi spoj relacija po zajedničkom atributu dat će sve retke iz relacije s lijeve strane operatora spajanja, čak i one za koje ne postoji podudarna vrijednost zajedničkog atributa u relaciji s desne strane. Pogledajmo primjer.

```
select *
from Trokut left join Krug
on Trokut.boja = Krug.boja;
```

Trokut.vrsta	Trokut.boja	Trokut.opseg	Krug.boja	Krug.površina
jednakokračan	plava	3	plava	16.45
raznostraničan	crvena	100	crvena	15.30
jednakostraničan	bijela	34	*NULL*	*NULL*
raznostraničan	zelena	12	*NULL*	*NULL*

Slično tome, desni spoj relacija po zajedničkom atributu dat će sve retke iz relacije s desne strane operatora spajanja, čak i one za koje ne postoji podudarna vrijednost zajedničkog operatora u relaciji s lijeve strane. Pogledajmo opet primjer:

```
select *
from Trokut right join Krug
on Trokut.boja = Krug.boja;
```

Trokut.vrsta	Trokut.boja	Trokut.opseg	Krug.boja	Krug.površina
jednakokračan	plava	3	plava	16.45
raznostraničan	crvena	100	crvena	15.30
NULL	NULL	NULL	narančasta	30.22

Konačno, pun spoj relacija po zajedničkom atributu dat će sve retke iz obje relacije, neovisno o tome podudaraju li se vrijednosti zajedničkog atributa ili ne. Tamo gdje se ne podudaraju, vrijednosti će biti nedefinirane. Ovu operaciju ne podržavaju sve implementacije jezika SQL. One koje je podržavaju, najčešće dopuštaju sintaktički izraz FULL JOIN.

```
select *
from Trokut full join Krug
on Trokut.boja = Krug.boja;
```

Trokut.vrsta	Trokut.boja	Trokut.opseg	Krug.boja	Krug.površina
jednakokračan	plava	3	plava	16.45
raznostraničan	crvena	100	crvena	15.30
jednakostraničan	bijela	34	NULL	NULL
raznostraničan	zelena	12	NULL	NULL
NULL	NULL	NULL	narančasta	30.22

3.6. Definiranje pogleda

Relacije se mogu zadavati i algoritmom, kao virtualne tablice. To je osobito pogodno ako želimo sačuvati često postavljane upite.

CREATE VIEW	CREATE VIEW ime_pogleda AS SELECT popis_atributa FROM popis_relacija WHERE uvjet

```
create view Likovi_iste_boje as
select
Trokut.boja as boja,
Trokut.opseg as opseg_trokuta,
Krug.površina as površina_kruga
from Trokut, Krug
where Trokut.boja = Krug.boja;
```

Sada se popis jednako obojenih likova može dobiti jednostavnom selekcijom:

```
select * from Likovi_iste_boje;
```

boja	opseg_trokuta	površina_kruga
plava	3	16.45
crvena	100	15.30

Napomena. U sustavu *Microsoft SQL Server* upit CREATE VIEW se tretira kao interna procedura i treba je pisati unutar ključnih riječi GO ... GO.

3.7. Spajanje projekcija i problem gubljenja informacija

Atributi i njihovi međusobni odnosi unutar jedne ili više relacija katkad mogu uzrokovati razne poteškoće i nepravilnosti prilikom dodavanja, promjene ili brisanja redaka.

U loše oblikovanim bazama podataka vrijednost nekog atributa nepotrebno će se ponavljati na mnogo mjesta, što, ako se radi o velikim relacijama, stvara razne praktične probleme prilikom promjene te vrijednosti. S druge strane, brisanjem nekih n-torki može se izgubiti informacija koja je trebala ostati sačuvana. Obratno, neke se n-torke možda ne mogu dodati bez unošenja nekih nepotrebnih informacija.

Zato je pri oblikovanju relacijskih baza podataka važno znati kako i koliko možemo prestrukturirati relacije da bismo izbjegli ovakve nepravilnosti, a da pri tom ne izgubimo informacije koje su sadržane u relacijama. Postupak takvog prestrukturiranja relacija nazivamo **normalizacijom**.

Prvo je značajno pitanje možemo li, i pod kojim uvjetima, relaciju rastaviti na neke njezine projekcije, a zatim spajanjem tih projekcija po nekom zajedničkom atributu (ili skupu atributa) obnoviti izvornu relaciju.

Pogledajmo tri takve projekcije naše relacije **Trokut**(**vrsta**, **boja**, **opseg**).

Trokut		
vrsta	**boja**	**opseg**
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

Promatrat ćemo projekcije ove relacije po atributima **vrsta** i **boja**, zatim **boja** i **opseg** te, konačno, po atributima **vrsta** i **opseg**. Dat ćemo im imena koja odražavaju projekcije, pa ćemo tako imati relacije **VrstaBoja**, **BojaOpseg** i **VrstaOpseg**.

Preciznije,

$$\textbf{VrstaBoja} = \pi_{\textbf{vrsta}, \textbf{boja}}(\textbf{Trokut})$$

$$\textbf{BojaOpseg} = \pi_{\textbf{boja}, \textbf{opseg}}(\textbf{Trokut})$$

$$\textbf{VrstaOpseg} = \pi_{\textbf{vrsta}, \textbf{opseg}}(\textbf{Trokut})$$

U jeziku SQL poslužit ćemo se upitima oblika SELECT INTO kako bismo napravili ove relacije, polazeći od izvorne relacije **Trokut**.

```
select distinct vrsta, boja into VrstaBoja
from Trokut;
select * from VrstaBoja;
```

VrstaBoja	
vrsta	**boja**
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela
raznostraničan	zelena

```
select distinct boja, opseg into BojaOpseg
from Trokut;
select * from BojaOpseg;
```

BojaOpseg	
boja	**opseg**
plava	3
crvena	100
bijela	34
zelena	12

```
select distinct vrsta, opseg into VrstaOpseg
from Trokut;
select * from VrstaOpseg;
```

VrstaOpseg	
vrsta	**opseg**
jednakokračan	3
raznostraničan	100
jednakostraničan	34
raznostraničan	12

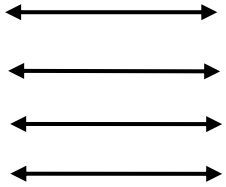
Relacije **VrstaBoja** i **BojaOpseg** imaju zajednički atribut **boja**, dok relacije **VrstaBoja** i **VrstaOpseg** imaju zajednički atribut **vrsta**.

Pogledat ćemo najprije spoj relacija **VrstaBoja** i **BojaOpseg** po zajedničkom atributu **boja** i utvrditi hoće li ta operacija obnoviti početnu relaciju **Trokut**. Pitamo se, dakle, je li

$$\textbf{Trokut} = \textbf{VrstaBoja} \blacktriangleright\blacktriangleleft_{\textbf{boja}} \textbf{BojaOpseg}$$

Iz dijagrama je lako vidjeti da će obnavljanje izvorne relacije uspjeti:

vrsta	boja
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela
raznostraničan	zelena



boja	opseg
plava	3
crvena	100
bijela	34
zelena	12

U jeziku SQL spoj ćemo ostvariti na uobičajeni način:

```
select VrstaBoja.vrsta, VrstaBoja.boja, BojaOpseg.opseg
from VrstaBoja, BojaOpseg
where VrstaBoja.boja = BojaOpseg.boja;
```

VrstaBoja.vrsta	VrstaBoja.boja	BojaOpseg.opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	zelena	12

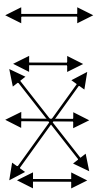
Možemo ustanoviti da smo u ovom slučaju spajanjem projekcija uspjeli obnoviti izvornu relaciju. To, međutim, nije uvijek moguće. Pogledajmo spoj relacija **VrstaBoja** i **VrstaOpseg** po zajedničkom atributu **vrsta**.

Ustanovit ćemo da

$$\textbf{Trokut} \neq \textbf{VrstaBoja} \blacktriangleright\blacktriangleleft_{\textbf{vrsta}} \textbf{VrstaOpseg}$$

Iz dijagrama se može vidjeti zašto obnavljanje izvorne relacije ovog puta neće uspjeti:

vrsta	boja
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela
raznostraničan	zelena



vrsta	opseg
jednakokračan	3
raznostraničan	100
jednakostraničan	34
raznostraničan	12

Pokažimo to i pomoću upita u jeziku SQL.

```
select VrstaBoja.vrsta, VrstaBoja.boja, VrstaOpseg.opseg
from VrstaBoja, VrstaOpseg
where VrstaBoja.vrsta = VrstaOpseg.vrsta;
```

VrstaBoja.vrsta	VrstaBoja.boja	VrstaOpseg.opseg
jednakokračan	plava	3
raznostraničan	crvena	100
raznostraničan	crvena	12
jednakostraničan	bijela	34
raznostraničan	zelena	12
raznostraničan	zelena	100

Dva tamnija retka, odnosno n-torke (raznostraničan, crvena, 12) i (raznostraničan, zelena, 100) pojavile su se u spoju, a ne pripadaju izvornoj relaciji. Pojavio se tzv. **problem gubljenja informacija**.

Proučimo li pobliže zašto spoj projekcija po atributu **vrsta** nije dao izvornu relaciju, vidjet ćemo da je problem zapravo u tome što u relaciji **VrstaBoja** za pojedinu vrijednost **vrste** postoji više od jedne vrijednosti **boje**. Zaista, imamo raznostraničan trokut crvene i zelene boje.

Također, u relaciji **VrstaOpseg** za jednu vrijednost **vrste** postoji više od jedne vrijednosti **opsega**. Opet, imamo raznostraničan trokut opsega 12 i opsega 34.

Pokušajmo to izmijeniti. Obojimo li zeleni trokut u crveni, imat ćemo u relaciji Trokut n-torku (raznostraničan, crvena, 12 umjesto n-torke (raznostraničan, zelena, 12). U jeziku SQL to bismo ostvarili ovim upitom.

```
update Trokut
set boja = 'crvena'
where boja = 'zelena';

select * from Trokut;
```

Trokut		
vrsta	**boja**	**opseg**
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	crvena	12

Projekcije **VrstaBoja**, **BojaOpseg** i **VrstaOpseg** sada će izgledati drugačije:

```
drop table VrstaBoja;
select distinct vrsta, boja into VrstaBoja
from Trokut;
select * from VrstaBoja;
```

VrstaBoja	
vrsta	**boja**
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela

Primijetimo da je nestala n-torka (raznostraničan, zelena).

```
drop table BojaOpseg;
select distinct boja, opseg into BojaOpseg
from Trokut;
select * from BojaOpseg;
```

BojaOpseg	
boja	**opseg**
plava	3
crvena	100
bijela	34
crvena	12

Naznačen je redak u kojem se crvena boja pojavljuje umjesto zelene. Primijetimo da sada, za razliku od prije, imamo dvije različite vrijednosti opsega (100 i 12) za istu vrijednost boje (crvena).

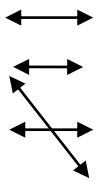
```
drop table VrstaOpseg;
select distinct vrsta, opseg into VrstaOpseg
from Trokut;
select * from VrstaOpseg;
```

VrstaOpseg	
vrsta	**opseg**
jednakokračan	3
raznostraničan	100
jednakostraničan	34
raznostraničan	12

U ovoj se tablici, kao što vidimo, ništa nije promijenilo.

Pokušamo li sad napraviti spoj relacija **VrstaBoja** i **VrstaOpseg** po zajedničkom atributu vrsta, ovog ćemo puta dobiti izvornu relaciju **Trokut** (to je ona u kojoj je nekadašnji zeleni trokut obojen u crveno).

vrsta	boja
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela



vrsta	opseg
jednakokračan	3
raznostraničan	100
jednakostraničan	34
raznostraničan	12

Provjerimo to.

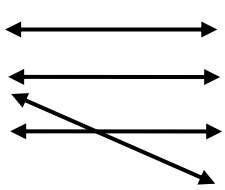
```
select VrstaBoja.vrsta, VrstaBoja.boja, VrstaOpseg.opseg
from VrstaBoja, VrstaOpseg
where VrstaBoja.vrsta = VrstaOpseg.vrsta;
```

VrstaBoja.vrsta	VrstaBoja.boja	BojaOpseg.opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	crvena	12

Napomena. Prilikom izvršavanja upita, redosljed redaka može biti i drugačiji, ali to je i dalje ista relacija.

Provjerimo još, nismo li promjenom zelene boje u crvenu narušili spoj projekcija **VrstaBoja** i **BojaOpseg** po atributu **boja**. Naime, u projekciji BojaOpseg imamo dvije vrijednosti opsega za jednu vrijednost boje (100 i 12 za crvenu). Iz dijagrama se vidi da nismo:

vrsta	boja
jednakokračan	plava
raznostraničan	crvena
jednakostraničan	bijela



boja	opseg
plava	3
crvena	100
bijela	34
crvena	12

Za obnavljanje izvorne relacije bilo je dovoljno da bar u projekciji **VrstaBoja** ni za jednu vrijednost **boje** nema više od jedne vrijednosti **vrste**.

Ponovo to možemo provjeriti upitom:

```
select VrstaBoja.vrsta, VrstaBoja.boja, BojaOpseg.opseg
from VrstaBoja, BojaOpseg
where VrstaBoja.boja = BojaOpseg.boja;
```

VrstaBoja.vrsta	VrstaBoja.boja	BojaOpseg.opseg
jednakokračan	plava	3
raznostraničan	crvena	100
jednakostraničan	bijela	34
raznostraničan	crvena	12

Ovo istraživanje uvjeta pod kojim se relacija može obnoviti spajanjem svojih projekcija bez gubitka informacija vodi nas do pojma **funkcionalne zavisnosti** o kojem će biti riječi u sljedećem poglavlju.

3.8. Pitanja za ponavljanje

1. Neformalno definirajte projekciju i restrikciju.
2. Neformalno definirajte uniju, presjek i razliku relacija.
3. Što je Descartesov produkt relacija (neformalno)?
4. Što je spoj relacija po zajedničkom atributu?
5. Što se podrazumijeva pod problemom gubljenja informacija?

4. Elementi teorije normalizacije

Sažetak

Vrijeme je da se kroz učenje jezika SQL upoznamo s međusobnim odnosima atributa unutar relacija i da naučimo kako oblikovati relacije na način da ih se može lako pretraživati, mijenjati i dopunjavati. Riječ je o tzv. teoriji normalizacije. Upoznat ćemo

- funkcionalne zavisnosti
- ponavljajuće skupine, pojam ključa i prvu normalnu formu
- nepotpune funckonalne zavisnosti i drugu normalnu formu
- tranzitivne zavisnosti i treću normalnu formu
- kako iskoristiti prednosti, a izbjeći probleme koje donosi normalizacija

4.1. Definicija funkcionalne zavisnosti

Izradimo relaciju **Lik(ime, naslov, autor)** koja će prikazivati likove iz poznatih književnih djela:

```
create table Lik
(
ime varchar(20) not null,
djelo varchar(30) not null,
autor varchar(30)
);

insert into Lik
(ime, djelo, autor)
values
('Wolland', 'Majstor i Margarita', 'Mihail Bulgakov'),
('Azazello', 'Majstor i Margarita', 'Mihail Bulgakov'),
('Adrian Leverkuhn', 'Doktor Faustus', 'Thomas Mann'),
('Mephisto', 'Mephisto', 'Klaus Mann'),
('Mephisto', 'Faust', 'Johann Wolfgang Goethe'),
('Wagner', 'Faust', 'Johann Wolfgang Goethe'),
('Frosch', 'Faust', 'Johann Wolfgang Goethe');

select * from Lik;
```

Lik		
ime	**djelo**	**autor**
Wolland	Majstor i Margarita	Mihail Bulgakov
Azazello	Majstor i Margarita	Mihail Bulgakov
Adrian Leverkuhn	Doktor Faustus	Thomas Mann
Mephisto	Mephisto	Klaus Mann
Mephisto	Faust	Johann Wolfgang Goethe
Wagner	Faust	Johann Wolfgang Goethe
Frosch	Faust	Johann Wolfgang Goethe

Napomena. Neki sustavi ne dozvoljavaju ovakav višestruki unos putem izravne SQL naredbe, ali za tu svrhu imaju razna zamjenska rješenja.

Definicija. Neka je $R(A_1, ..., A_n)$ relacija, a X i Y podskupovi njezinih atributa. Možemo bez gubitka općenitosti pretpostaviti da je $X = \{ A_1, ..., A_k \}$, a $Y = \{ A_{k+1}, ..., A_n \}$ za neki k, takav da je $1 \leq k < n$. Kažemo da je relacija R **funkcija** iz $\pi_X(R)$ u $\pi_Y(R)$ ako i samo ako za svaku vrijednost $(x_1,..., x_k) \in \pi_X(R)$ postoji jedna i samo jedna vrijednost $(x_{k+1},...,x_n) \in \pi_Y(R)$, takva da je $(x_1,..., x_n) \in R$. Skup X tada zovemo **ključem** relacije R, atribute iz X **ključnim atributima**, a atribute iz Y **sporednim atributima**.

Napomena. Relacija može imati i više od jednog ključa, a ponekad imamo potrebu napraviti i kompozitni ključ, sastavljen od više stupaca. Svaki od njih na jednoznačan način jedinstveno određuje retke relacije. Onaj koji se po pohranjenim podacima učini da je najprimjereniji, odaberemo da nam služi za identifikaciju redaka. Takav ključ nazivamo **primarnim ključem**.

Promotrimo odnose među vrijednostima pojedinih stupaca relacije **Lik**. Svako **djelo** ima u ovoj relaciji samo jednog **autora**. Preciznije, za svaku vrijednost iz projekcije $\pi_{\textbf{djelo}}(\textbf{Lik})$ postoji jedna i samo jedna vrijednost u projekciji $\pi_{\textbf{autor}}(\textbf{Lik})$, što po definiciji znači da je relacija $\pi_{\textbf{djelo,autor}}(\textbf{Lik})$ funkcija iz $\pi_{\textbf{djelo}}(\textbf{Lik})$ u $\pi_{\textbf{autor}}(\textbf{Lik})$.

Ovo će biti vidljivije ako imamo pred sobom navedenu projekciju $\pi_{\textbf{djelo,autor}}$.

```
select distinct djelo, autor
from Lik;
```

$\pi_{\textbf{djelo,autor}}(\textbf{Lik})$	
djelo	**autor**
Majstor i Margarita	Mihail Bulgakov
Doktor Faustus	Thomas Mann
Mephisto	Klaus Mann
Faust	Johann Wolfgang Goethe

Povezanost koju smo upravo otkrili među atributima **djelo** i **autor** u relaciji **Lik** simbolički ćemo zapisati ovako:

{**djelo**} ===> {**autor**}

i reći da postoji **funkcionalna zavisnost** između atributa **djelo** i atributa **autor**, odnosno da **djelo** funkcionalno određuje **autora**.

Možemo također utvrditi da grupa atributa **ime** i **djelo** funkcionalno određuje **autora**, tj.

{**ime**, **djelo**} ===> {**autor**}

jer za svaku kombinaciju **imena** i **djela** postoji jedna i samo jedna vrijednost pripadajućeg **autora**.

S druge strane, možemo primijetiti da se **ime** jednog lika pojavljuje u više **djela** različitih **autora**. Na primjer, Mephisto je lik iz istoimenog romana Klausa Manna, ali i iz tragedije „Faust“ Johanna Wolfganga Goethea. Zaključujemo, dakle, da ne postoji funkcionalna zavisnost između atributa **ime** i **djelo**, a ni između atributa **ime** i **autor**. Simbolički,

{**ime**} =/=> {**djelo**, **autor**}

Također, u jednom **djelu** pojavljuje se više različitih **imena** likova. Na primjer, u Goetheovu „Faustu“ pojavljuju se Mephisto, Wagner i Frosch, a u Bulgakovljevu „Majstoru i Margariti“ pojavljuju se Wolland i Azazello. Zaključujemo da

{**djelo**} =/=> {**ime**}

Funkcionalnu zavisnost možemo sada definirati i općenito.

Definicija. Neka je R relacija, a X i Y podskupovi njezinih atributa. Kažemo da između X i Y postoji **funkcionalna zavisnost** X ===> Y ako i samo ako je relacija $\pi_{XY}(R)$ u svakom trenutku funkcija iz $\pi_X(R)$ u $\pi_Y(R)$.

Uočimo i jedan ključ relacije **Lik**, dakle poseban skup atributa koji funkcionalno određuje sve preostale atribute u relaciji. Vidjeli smo već da **ime** i **djelo** u kombinaciji funkcionalno određuju **autora**, tj. preostali atribut relacije. Ova skupina atributa je, dakle, po definiciji ključ relacije **Lik**, odnosno **ime** i **djelo** su njezini ključni atributi. Ključne atribute obično označavamo podcrtano. Imamo, dakle

{**ime**, **djelo**} ===> {**autor**}

Vidljivo je da su funkcionalne zavisnosti vrlo važne u procesu oblikovanja baza podataka. O dobrom rasporedu funkcionalnih zavisnosti u relacijama ovisi učinkovitost unosa, mijenjanja i brisanja podataka u bazi. O tome govori **teorija normalizacije**.

4.2. Ponavljajuća skupina i prva normalna forma

Primjer. Zamislimo da su likovi iz prethodne relacije profesori stvarnih i izmišljenih predmeta na starim europskim sveučilištima i pogledajmo relaciju **Kolegij (profesor, predmet)** koja sadrži podatke o tome koje predmete predaju pojedini profesori[1].

Kolegij			
profesor	**predmet**		
Faust	alkemija	crna magija	
Wolland	alkemija	crna magija	metafizika
Kant	metafizika		

Odmah uočavamo da ova relacija po svojoj strukturi nije nalik na one s kojima smo do sada radili. Naime, vrijednosti atributa u relaciji **Kolegij** nisu nedjeljive, jer je atribut **predmet** zapravo zasebna relacija, prikazana tablicom unutar tablice. Takav atribut nazivamo **ponavljajućom skupinom**. Retci ovakve tablice nisu fiksne duljine pa ne može postojati ni ključ relacije na način kako smo definirali taj pojam. SQL ne predviđa rad s relacijama koje sadrže ponavljajuću skupinu.

Definicija: Relacija je u prvoj normalnoj formi (1NF) ako su vrijednosti njenih atributa nedjeljive. .

Drugim riječima, atributi relacije koja je u 1NF nisu i sami relacije. Takva relacija ne sadrži tablicu u tablici ili ponavljajuću skupinu atributa. Ona također uvijek sadrži ključ koji je, u najgorem slučaju, skup svih atributa relacije.

Želimo li iz tablice **Kolegij** izbaciti ponavljajuću skupinu, odnosno dobiti tablicu čiji će retci biti fiksne duljine, unijet ćemo za svaku vrijednost **predmeta** po jedan redak u kojem će se nalaziti ime pripadajućeg **profesora**.

[1] Imena u ovim primjerima likovi su iz književnih djela, ali i povijesne ličnosti. Faust, Wagner i Frosch likovi su iz Goetehove tragedije „Faust“. Wolland, Behemot i Azazello potječu iz Bulgakovljeva romana „Majstor i Margarita“. Immanuel Kant je stvarna osoba, njemački filozof iz 19. stoljeća.

```
create table Kolegij
(
profesor varchar(20) not null,
predmet varchar(20) not null
);

insert into Kolegij
(profesor, predmet)
values
('Faust', 'alkemija'),
('Faust', 'crna magija'),
('Wolland', 'alkemija'),
('Wolland', 'crna magija'),
('Wolland', 'metafizika'),
('Kant', 'metafizika');

select * from Kolegij;
```

Kolegij	
profesor	**predmet**
Faust	alkemija
Faust	crna magija
Wolland	alkemija
Wolland	crna magija
Wolland	metafizika
Kant	metafizika

Ovako prepravljena relacija K**olegij (profesor, predmet)** nema ponavljajuću skupinu, a skup {**profesor**, **predmet**} je njezin ključ pa možemo zaključiti da je relacija u 1NF.

Drugih ključeva nema, jer ni za jednog profesora nema jednoznačno određenog predmeta, a također ni za jedan predmet nema jednoznačno određenog profesora koji ga predaje. Dakle,

{**profesor**} =/=> {**predmet**}

{**predmet**} =/=> {**profesor**}

4.3. Nepotpune funkcionalne zavisnosti i druga normalna forma

Definicija. Neka su X i Y skupovi atributa. Funkcionalnu zavisnost X ===> Y nazivamo **nepotpunom** ako postoji pravi podskup X' od X koji funkcionalno određuje Y, tj. za koji je X' ===> Y. Inače kažemo da je X ===> Y **potpuna**.

Primjer: Dodajmo sada relaciji **Kolegij** atribut **sveučilište** koji sadrži podatke o tome gdje profesor održava svoja predavanja. Pretpostavit ćemo da profesori održavaju predavanja samo na sveučilištu u svojem rodnom gradu. Dobit ćemo novu relaciju **Kolegij (profesor, predmet, sveučilište)**, koju ćemo nadopuniti ovako:

```
alter table Kolegij
add sveučilište varchar(20);

update Kolegij
set sveučilište = 'Heidelberg'
where profesor = 'Faust';

update Kolegij
set sveučilište = 'Konigsberg'
where profesor = 'Wolland' or profesor = 'Kant';
select * from Kolegij;
```

Kolegij		
profesor	**predmet**	**sveučilište**
Faust	alkemija	Heidelberg
Faust	crna magija	Heidelberg
Wolland	alkemija	Konigsberg
Wolland	crna magija	Konigsberg
Wolland	metafizika	Konigsberg
Kant	metafizika	Konigsberg

Pogledamo li retke ove relacije, vidjet ćemo da Faust predaje alkemiju u Heidelbergu i nigdje drugdje. Nadalje, Faust predaje crnu magiju u Heidelbergu i nigdje drugdje. Napravimo li istu provjeru za Wollanda i Kanta, zaključit ćemo da **profesor** i **predmet** jednoznačno određuju **sveučilište**. Kako u relaciji nema drugih atributa, ovaj skup je ujedno i ključ relacije pa je ona u 1NF.

U skladu s našom pretpostavkom da svaki profesor predaje samo u svojem rodnom gradu, vidimo da Faust predaje samo u Heidelbergu, a Wolland i Kant samo u Konigsbergu, pa zaključujemo da i sam **profesor** jednoznačno određuje **sveučilište**. Formalno zapisane, ove funkcionalne zavisnosti izgledaju ovako:

{**profesor**, **predmet**} ==> {**sveučilište**}

{**profesor**} ==> {**sveučilište**}

Pojavljuje se, dakle, nepotpuna funkcionalna zavisnost sporednog atributa od ključa.

Što se tiče ostalih mogućih funkcionalnih zavisnosti, **profesor** ne određuje **predmet** jer, na primjer, Wolland predaje i alkemiju i crnu magiju i metafiziku. Također, ni **predmet** ne određuje **profesora** jer, na primjer, metafiziku predaju i Wolland i Kant, ali ne određuje ni **sveučilište**,jer se, na primjer, alkemija predaje i u Heidelbergu i u Konigsbergu. Ni **sveučilište** ne određuje **profesora**, jer u Heidelberu predaju i Wolland i Kant. Nadalje, ni kombinacija **sveučilišta** i **predmeta** ne određuju **profesora**, jer, na primjer, metafiziku u Konigsbergu predaju i Kant i Wolland.

Definicija: Relacija **R** je u **drugoj normalnoj formi** (2NF) ako je u prvoj normalnoj formi i ako ne postoji nepotpuna funkcionalna zavisnost bilo kojeg sporednog atributa relacije **R** od bilo kojeg ključa te relacije.

Prema ovoj definiciji relacija **Kolegij** nije u 2NF.

Relacije koje nisu u 2NF pokazuju karakteristične anomalije unošenja, brisanja i ažuriranja podataka. Na primjer, ne možemo unijeti podatak da prof. Moriarti ima katedru na sveučilištu u Oxfordu ako on trenutačno ne predaje ni jedan predmet (anomalija unošenja). Ako prof. Kant otkaže sva svoja predavanja, gubimo informaciju da on živi u Konigsbergu (anomalija brisanja). Ako prof. Wolland preseli iz Konigsberga u Sankt Petersburg, taj podatak bi se trebao promijeniti za svaki predmet koji on predaje (anomalija ažuriranja).

Želimo li ovu relaciju normalizirati, izdvojit ćemo one atribute koji sudjeluju u nepotpunoj funkcionalnoj zavisnosti i od njih napraviti novu relaciju. Dakle, formirat ćemo relaciju **Katedra (<u>profesor</u>, sveučilište)** ovako:

```
create table Katedra
(
profesor varchar(20) not null,
sveučilište varchar(20)
);

insert into Katedra
(profesor, sveučilište)
values
('Faust', 'Heidelberg'),
('Wolland', 'Konigsberg'),
('Kant', 'Konigsberg');

select * from Katedra;
```

Katedra	
<u>profesor</u>	**sveučilište**
Faust	Heidelberg
Wolland	Konigsberg
Kant	Konigsberg

Napomena. Podatke za relaciju **Katedra** mogli smo dobiti i uporabom podupita (na onim sustavima gdje su podupiti dozvoljeni). Budući da traženi podaci već postoje u relaciji **Kolegij**, imali bismo:

```
truncate table Katedra;
insert into Katedra
  select distinct profesor, sveučilište
  from Kolegij;
```

Ovdje očito vrijedi {**profesor**} ===> {**sveučilište**} pa je relacija sigurno u 1NF, jer ima ključ, a također je u 2NF, jer nema nepotpunih funkcionalnih zavisnosti sporednih atributa od ključa.

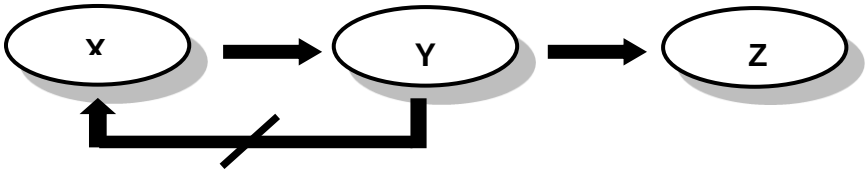
Preostali atributi formiraju već promatranu relaciju **Kolegij (profesor, predmet)** u kojoj također nema nepotpunih funkcionalnih zavisnosti pa je ona u 2NF.

```
alter table Kolegij
drop column sveučilište;

select * from Kolegij;
```

Kolegij	
profesor	**predmet**
Faust	alkemija
Faust	crna magija
Wolland	alkemija
Wolland	crna magija
Wolland	metafizika
Kant	metafizika

4.4. Tranzitivne zavisnosti i treća normalna forma

Definicija. Neka su X, Y i Z skupovi atributa. Funkcionalnu zavisnost X ===> Z nazivamo **tranzitivnom** ako se može prikazati kao kompozicija X ===> Y ===> Z, pri čemu Y =≠=> X.



Primjer. Zamislimo sada da studenti na ovim sveučilištima biraju svoje mentore, odnosno profesori imaju svoje klase studenata. Prikažimo to relacijom **Klasa (student, mentor, sveučilište)**. Neka studenti budu Wagner, Behemot, Azazello i Frosch i definirajmo tablicu mentorstva tako da Faust bude mentor Wagneru, Wolland Behemotu i Azazellu, a Kant Froschu. Pripadnost sveučilištima ista je kao i prije.

```
create table Klasa
(
student varchar(20) not null,
mentor varchar(20),
sveučilište varchar(20)
);

insert into Klasa
(student, mentor, sveučilište)
values
('Wagner', 'Faust', 'Heidelberg'),
('Behemot', 'Wolland', 'Konigsberg'),
('Azazello', 'Wolland', 'Konigsberg'),
('Frosch', 'Kant', 'Konigsberg');

select * from Klasa;
```

Klasa		
student	**mentor**	**sveučilište**
Wagner	Faust	Heidelberg
Behemot	Wolland	Konigsberg
Azazello	Wolland	Konigsberg
Frosch	Kant	Konigsberg

Napomena. Podatke za relaciju **Klasa** mogli smo unijeti kombinacijom ručnog unosa i podupita nad relacijom **Katedra**, na primjer ovako:

```
truncate table Klasa;

insert into Klasa
(student, mentor)
values
('Wagner', 'Faust'),
('Behemot', 'Wolland'),
('Azazello', 'Wolland'),
('Frosch', 'Kant');

update Klasa
set sveučilište =
  (select sveučilište from Katedra
   where Klasa.mentor = Katedra.profesor);
```

Lako uočavamo da je **student** ključ ove relacije, jer svaki student bira samo jednog mentora i studira na samo jednom sveučilištu. Dakle,

{**student**} ===> {**mentor**, **sveučilište**}

No, s druge strane, kako profesori predaju samo na sveučilištima u svojim rodnim gradovima, vidimo da i **mentor** određuje **sveučilište**, ali mentor ne određuje studenta. Simbolički,

{**student**} ===> {**mentor**} ===> {**sveučilište**}

{**mentor**} =/=> {**student**}

Imamo, dakle, tranzitivnu zavisnost sporednog atributa od ključa relacije.

Definicija. Kažemo da je relacija u **trećoj normalnoj formi** (3NF) ako je u 2NF i ako u njoj nema tranzitivne zavisnosti ni jednog sporednog atributa od nekog ključa te relacije.

Prema ovoj definiciji, relacija **Klasa** nije u 3NF.

Relacije koje nisu u 3NF također pokazuju anomalije unošenja, brisanja i ažuriranja. Na primjer, ponovo ne možemo unijeti podatak da prof. Moriarti predaje na sveučilištu u Oxfordu ako nema ni jednog studenta kojemu je on mentor. Kad Wagner završi studij, njegovim brisanjem gubimo podatke i o njegovu mentoru, prof. Faustu. Ako prof. Wolland preseli iz Konigsberga u Moskvu, taj podatak moramo ažurirati kod svih studenata kojima je on mentor.

Da bismo dobili relacije u 3NF, izdvojit ćemo problematične atribute i od njih napraviti novu relaciju. Izdvojimo li **mentor** i **sveučilište**, vidimo da smo tu relaciju već napravili pod imenom **Katedra**, pri čemu je **mentor** zapravo **profesor**. U relaciji **Klasa** ostaju, dakle, **student** i **mentor**, pri čemu svaki pojedini mentor treba odgovarati nekom profesoru u relaciji **Katedra**.

```
alter table Klasa
drop column sveučilište;

select * from Klasa;
```

Klasa	
student	**mentor**
Wagner	Faust
Behemot	Wolland
Azazello	Wolland
Frosch	Kant

```
select * from Kolegij;
```

Katedra	
profesor	**sveučilište**
Faust	Heidelberg
Wolland	Konigsberg,
Kant	Konigsberg

Ni u jednoj od ovih relacija nema više tranzitivnih zavisnosti pa su one, dakle, u 3NF.

4.5. Više normalne forme

Relacije se mogu normalizirati i dalje od 3NF. Postoje, naime, zavisnosti koje su složenije od funkcionalnih, a nazivamo ih **višeznačne zavisnosti**. Prisutnost višeznačnih zavisnosti može stvoriti situacije pri kojima se anomalije unošenja, brisanja i ažuriranja pojavljuju i kad su relacije dovedene do 3NF. Iz proučavanja takvih situacija nastale su **Boyce-Coddova normalna forma** (BCNF) te **četvrta** i **peta normalna forma** (4NF i 5NF). Ipak, ove više normalne forme primarno su od teoretskog značaja. Za praktične primjene u većini slučajeva posve je dovoljno svesti relacije na 3NF.

4.6. Nenormalizirani relacijski model

Postoje okolnosti u kojima je opravdano u određenoj mjeri odustati od normalizacije. Te okolnosti ovise o značajkama i primarnoj namjeni pojedine baze podataka.

Što je normalna forma relacije viša, to su upiti dodavanja, brisanja i promjene jednostavniji. Zato bi relacije u kojima se podaci često mijenjaju trebale bi biti u što višoj normalnoj formi. Međutim, selekcija podataka iz takvih relacija može pri velikoj količini podataka postati zamršena.

S druge strane, što je normalna forma relacije niža, to su upiti selekcije jednostavniji. Zato relacije u kojima su podaci razmjerno statični i koje služe pretežito za prikaz podataka ne bi trebale biti previše normalizirane. Međutim, promjena podataka u takvim relacijama može pri velikoj količini podataka biti kompliciranа.

Koliko daleko ići s normalizacijom relacija odluka je koja se donosi za svaku bazu podataka i ovisi o iskustvu projektanta. U praksi se pokazuje da je 3NF za većinu uobičajenih primjena dobro rješenje, ali ga ne treba primjenjivati previše kruto. S druge strane, odustajanje je od normalizacije nešto što treba primjenjivati s mjerom i razumnom dozom opreza.

U našem primjeru, kad bismo htjeli sve informacije iz relacija **Kolegij**, **Katedra** i **Klasa** prikazati u velikoj relaciji **Studij**, koja bi bila samo u 1NF, ona bi imala oblik **Studij** (**student**, **profesor**, **predmet**, **sveučilište**) i izgledala bi ovako:

Studij			
student	**profesor**	**predmet**	**sveučilište**
Wagner	Faust	alkemija	Heidelberg
Wagner	Faust	crna magija	Heidelberg
Behemot	Wolland	alkemija	Konigsberg
Behemot	Wolland	crna magija	Konigsberg
Behemot	Wolland	metafizika	Konigsberg
Azazello	Wolland	alkemija	Konigsberg
Azazello	Wolland	crna magija	Konigsberg
Azazello	Wolland	metafizika	Konigsberg
Frosch	Kant	metafizika	Konigsberg

Vidimo da postoji velika količina ponavljanja podataka, čak i za ovako malu tablicu. Upiti selekcije na ovakvoj bi tablici bili jednostavni, ali upiti dodavanja, brisanja i ažuriranja podataka postali bi komplilcirani, a treba uzeti u obzir i pojavu svih ranije navedenih nepravilnosti koje se javljaju u nenormaliziranim modelima.

Napomena. Pomoću podataka iz dosadašnjih relacija, relacija se **Studij** može rekonstruirati na ovaj način:

```
create table Studij (
  student varchar(20),
  profesor varchar(20),
  predmet varchar(20),
  sveučilište varchar(20)
);

insert into Studij
(student, profesor, predmet, sveučilište)
  select
    Klasa.student,
    Kolegij.profesor,
    Kolegij.predmet,
    Katedra.sveučilište
  from
    Klasa, Kolegij, Katedra
  where
    Klasa.mentor = Kolegij.profesor and
    Kolegij.profesor = Katedra.profesor;
```

4.7. Problem samogovorećih ključeva

Normalizacija neke relacije podrazumijeva, kao što smo vidjeli, njezino rastavljanje na projekcije iz kojih se kasnije izvorna relacija može rekonstruirati. Primijetimo da smo pri tome uvijek pazili da zajednički atribut po kojem će se projekcije spajati bude ključ u bar jednoj od projekcija. Postupajući tako, osiguravamo da bude ispunjen uvjet obnavljanja relacije spajanjem njenih projekcija bez gubitka informacija.

Na primjer, izvornu relaciju **Klasa** (**student**, **mentor**, **sveučilište**) rastavili smo na relacije **Klasa** (**student**, **mentor**) i **Katedra** (**profesor**, **sveučilište**), pri čemu su **mentor** i **profesor** zapravo jedan te isti atribut, a on je ključ u relaciji **Katedra**.

S druge strane, taj se atribut u novoj relaciji **Klasa** pojavljuje kao sporedni atribut **mentor** čije vrijednosti moraju odgovarati vrijednostima primarnog ključa **profesor** u relaciji **Katedra**. Takav atribut nazivamo **stranim ključem** u promatranoj relaciji.

Svi atributi s kojima smo se do sada susreli imali su neku semantičku vrijednost, odnosno značenje koje proizlazi iz toga da izražavaju određeno svojstvo objekta u realnom sustavu. To se odnosilo i na ključeve relacije. Takve ključeve zato nazivamo **samogovorećim ključevima**. Na primjer, **profesor** je primarni ključ relacije **Katedra** i njegove vrijednosti su stvarna imena nekih profesora. To ne predstavlja nikakvu poteškoću tako dugo dok se ne pokaže potreba za promjenom vrijednosti takvih atributa. Promotrimo naš dosadašnji sustav relacija:

Katedra(profesor, sveučilište),

Kolegij(profesor, predmet),

Klasa(student, mentor),

Upiti nad ovim sustavom relacija relativno su jednostavni, ali ako se promijeni neki od ključnih atributa, procedura ažuriranja je složenija. Želimo li, na primjer, za sve studente potražiti njihove mentore i sveučilišta na kojima su upisani, to se može izvesti razmjerno jednostavnim upitom: .

```
select Klasa.student, Klasa.mentor, Katedra.sveučilište
from Klasa, Katedra
where
Klasa.mentor = Katedra.profesor
```

Klasa.student	**Klasa.mentor**	**Katedra.sveučilište**
Wagner	Faust	Heidelberg
Behemot	Wolland	Konigsberg
Azazello	Wolland	Konigsberg
Frosch	Kant	Konigsberg

Pretpostavimo sada da Faust promijeni ime u Faustus. Riječ je, dakle, o promjeni vrijednosti ključnog atributa **profesor**.

Budući da u ovim relacijama ključevi imaju značenja, promjene njihovih vrijednosti kaskadno se odražavaju na više mjesta i ne mogu se izvesti jednostavnim ažuriranjem bez narušavanja integriteta podataka između različitih relacija.

Taj se problem rješava uvođenjem posebnog atributa u svaku relaciju, tzv. **identifikatora**, koji služi samo kao ključ, ali nema smislenog značenja. Najbolje je da to bude redni broj retka u relaciji. Takav **negovoreći ključ** nikada ne treba ažuirati.

Na primjer, uvedemo li nove relacije

Student (**id**, **ime**),

Profesor (**id**, **ime**),

Predmet (**id**, **naziv**) i

Sveučilište (**id**, **naziv**)

onda možemo imati:

Katedra (**id_profesora**, **id_sveučilišta**)

Kolegij (**id_profesora**, **id_predmeta**)

Klasa (**id_studenta**, **id_profesora**)

Relacije **Katedra**, **Kolegij** i **Klasa** osim primarnog ključa sadrže neke sporedne (ili ključne) atribute, koji se pojavljuju kao primarni ključevi u drugim relacijama.

Tako je, na primjer, **id_sveučilišta**, koji se u relaciji **Katedra** pojavljuje kao sporedni atribut, istodobno primarni ključ u relaciji **Sveučilište**. Atribut **id_profesora**, koji se u relacijama **Katedra** i **Kolegij** pojavljuje kao ključni atribut (dio ključa), istodobno je primarni ključ u relaciji **Profesor**. Atribut **id_studenta**, koji se u relaciji **Klasa** pojavljuje kao primarni ključ, također je primarni ključ u relaciji **Student**.

Ovakve atribute, koji se pojavljuju kao primarni ključevi u drugim relacijama, nazivamo **stranim ključevima**.

```
create table Profesor (
id int not null,
ime varchar(20)
);
insert into Profesor
(id, ime)
values
(1, 'Faust'),
(2, 'Wolland'),
(3, 'Kant');

create table Sveučilište (
id int not null,
naziv varchar(20)
);
insert into Sveučilište
(id, naziv)
values
(1, 'Heidelberg'),
(2, 'Konigsberg');

create table Katedra
(
id_profesora int not null,
id_sveučilišta int not null
);
insert into Katedra
(id_profesora, id_sveučilišta)
values
(1,1),
(2,2),
(3,2);

select Profesor.ime, Sveučilište.naziv
from Profesor, Sveučilište, Katedra
where
Katedra.id_profesora = Profesor.id AND
Katedra.id_sveučilišta = Sveučilište.id
```

Profesor.ime	**Sveučilište.naziv**
Faust	Heidelberg
Wolland	Konigsberg
Kant	Konigsberg

Dodajmo još relacije **Student** i **Klasa**:

```
create table Student (
id int not null,
ime varchar(20)
);
insert into Student
(id, ime)
values
(1, 'Wagner'),
(2, 'Behemot'),
(3, 'Azazello'),
(4, 'Frosch');
```

```
create table Klasa (
id_studenta int not null,
id_profesora int not null
);
insert into Klasa
(id_studenta, id_profesora)
values
(1,1),
(2,2),
(3,2),
(4,3);
```

Iz ovog sustava relacija možemo dobiti sve podatke iz reda predavanja, iako će upiti postati složeniji.

```
select Student.ime, Profesor.ime, Sveučilište.naziv
from Student, Profesor, Sveučilište, Klasa
where
Student.id = Klasa.id_studenta AND
Profesor.id = Klasa.id_profesora AND
Sveučilište.id = Profesor.id_sveučilišta
```

Ako se bilo koji podatak promijeni, morat će se se mijenjati samo na jednom mjestu. Na primjer, promjena imena profesora Fausta u profesor Faustus vrlo je jednostavna:

```
update Profesor
set ime = 'Faustus'
where ID = 1;
```

4.8. Pitanja za ponavljanje

1. Što je to ključ relacije?
2. Kada je relacija u prvoj normalnoj formi?
3. Kada je relacija u drugoj normalnoj formi?
4. Kada je relacija u trećoj normalnoj formi?
5. Zašto treba izbjegavati samogovoreće ključeve?

5. Oblikovanje relacijske baze podataka

Sažetak

U završnom poglavlju vraćamo se na početak, na oblikovanje relacijske baze podataka. Upoznat ćemo praktične i jednostavne tehnike modeliranja podataka te pravila prevođenja dobivenog modela u skup naredbi jezika SQL kojima ćemo oblikovati bazu podataka. Naučit ćemo konstruirati

- entitetne i atribute tipove
- tipove veza
- dijagram entiteta i veza
- skup upita u jeziku SQL koji će na osnovu danog modela izgraditi bazu podaka

5.1. Model entiteta i veza

Oblikovanje relacijske baze podataka normalizacijom na način koji smo opisali, u praksi se pokazuje dosta teškim i složenim, pogotovo kad je riječ o većem broju relacija i atributa. Postoje, međutim, razni postupci modeliranja podataka koji su razmjerno jednostavni i dobro primjenjivi u praksi, a zasnivaju se na opisivanju realnog sustava jezikom koji je mješavina prirodnog i formalnog jezika. Takav formalizirani opis realnog sustava zapravo je model više razine apstrakcije nego što je to relacijski model. Ovakav međumodel nazivamo **konceptualni model podataka**.

Postupak koji ćemo opisati sastoji se, dakle, od dva koraka. Umjesto da izravno pokušavamo napraviti relacijski model, izradimo najprije konceptualni model podataka, formalizirani opis realnog sustava, koji se sastoji od sljedećih elemenata: entitetnih tipova, atributnih tipova i tipova veza. Jezgru našeg konceptualnog modela podataka čini tzv. **dijagram entiteta i veza** (*entity relationship diagram*), **ER dijagram**. Ova vrsta konceptuaolnog modela podataka naziva se **model entiteta i veza** (*entity-relationship model*), **ER model**.

Prevođenje ER modela u relacijski model može se tada obaviti uz pomoć malog broja jednostavnih pravila, a postupak se može automatizirati. Može se dokazati da će relacije dobivene ovim postupkom biti u 3NF.

5.2. Entitetni tipovi

Entitetni tip je misaona konstrukcija koja predstavlja kolekciju objekata realnog svijeta, usporedivih po nekim obilježjima.

Praktično pravilo za prepoznavanje entitetnih tipova kaže da ako za neku kolekciju objekata uspijevamo pronaći definiciju koja se u promatranom kontekstu odnosi na svaki pojedinačni objekt iz te kolekcije, a može se izraziti u obliku

Svaki <*objekt-danog-tipa*> je <*definicija-objekta*>,

onda je takva kolekcija dobar kandidat za entitetni tip.

Primjer. Za entitetni tip **Profesor** iz naših prethodnih primjera imali bismo definiciju:

- Svaki **Profesor** je stvarna ili izmišljena osoba kojoj je zbog poznavanja pojedinih znanja ili vještina odobreno poučavanje studenata.

Entitetni tip **Profesor** prikazat ćemo pravokutnikom s odgovarajućim natpisom.

Slične ćemo definicije dati za ostale entitetne tipove koje prepoznajemo u našem sustavu: **Sveučilište**, **Predmet** i **Student**.

- Svako **Sveučilište** je ustanova na kojoj se obavlja nastava i istraživanje.

- Svaki **Predmet** je skup znanja i vještina iz nekog područja.

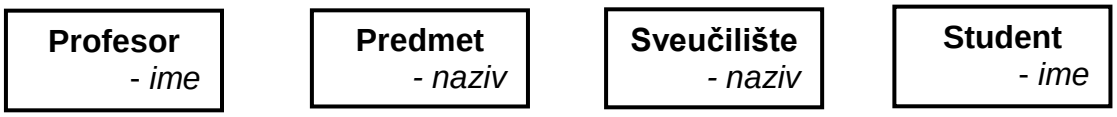
- Svaki **Student** je stvarna ili izmišljena osoba koja pohađa predavanja u okviru nekog studijskog programa.

Predstavit ćemo ih, kao i entitetni tip **Profesor**, odgovarajućim pravokutnicima:

5.3. Atributni tipovi

Atributni tip je misaona konstrukcija koja predstavlja vrstu obilježja po kojem se objekti istog entitetnog tipa mogu uspoređivati.

Primjer. Jedan je od atributnih tipova koje možemo dodijeliti entitetnom tipu **Profesor**, na primjer, **ime**. Možemo promatrati i druge odlike profesora, npr. **prezime**, **titula** ili **datum rođenja** ako je to od značaja za naš informacijski sustav. Važno je primijetiti je da tražimo samo one atributne tipove koji su izravno svojstveni entitetnom tipu. Za Profesora to, na primjer, ne bi bilo ime sveučilišta na kojem je zaposlen ili naziv predmeta koji predaje. Atributne tipove simbolički upisujemo u pravokutnik koji predstavlja pripadajući entitetni tip.



5.4. Tipovi veza

Tip veze je misaona konstrukcija koja predstavlja vrstu povezanosti između objekata dvaju entitetnih tipova.

Odabirom naziva za pojedini tip veze nastojimo što bolje odraziti prirodu povezanosti objekata realnog sustava. Zato se pri tome služimo izrazima koji su svojstveni samom realnom sustavu.

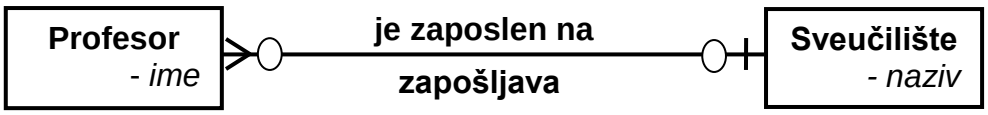
Kvantitativnu prirodu veze opisujemo izrazima 'nijedan ili jedan', 'jedan i samo jedan' te nijedan, jedan ili više'. Za svaki od ovih izraza rabimo određeni simbol, kao što ćemo vidjeti u primjerima.

Veze između objekata uvijek izražavamo u oba smjera, nazivom i kvantifikacijom.

Primjer. Definirajmo jednu od mogućih veza između **Profesora** i **Sveučilišta**:

- Svaki **Profesor** je zaposlen na nijednom ili jednom **Sveučilištu**. Obratno, svako **Sveučilište** zapošljava nijednog, jednog ili više **Profesora**.

Ove dvije tvrdnje simbolički se prikazuju ovako:

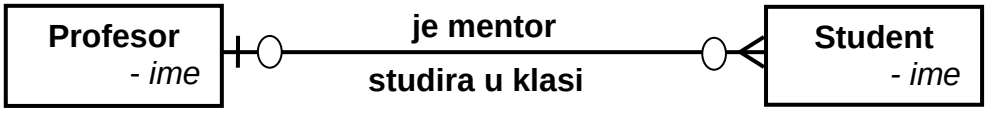


Simboli kružića, crtice i antene predstavljaju riječi 'nula', 'jedan' i 'više'. Riječi je zaposlen na i zapošljava opisuju stvarnu prirodu veze između **Profesora** i **Studenta**, dok izrazi 'jednom i samo jednom' te 'nula, jednog ili više' opisuju njezinu kvantifikaciju. Veza s ovakvom kvantifikacijom naziva se **veza tipa jedan prema više** ili **(1:m) veza**.

Slično tome, između **Profesora** i **Studenta** također imamo (1:m) vezu:

- Svaki **Profesor** je mentor nijednom, jednom ili više **Studenata**. Obratno, svaki **Student** studira u klasi jednog i samo jednog **Profesora**.

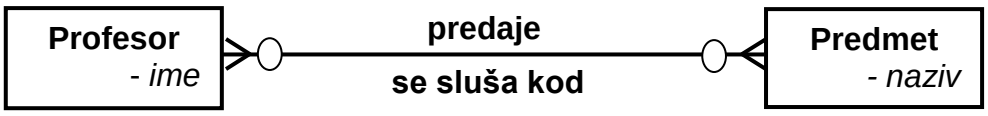
Simbolički prikazana, ta veza izgleda ovako:



Veza je između **Profesora** i **Predmeta** nešto drugačija.

- Svaki **Profesor** predaje nijedan, jedan ili više **Predmeta**. Obratno, svaki **Predmet** se sluša kod nijednog, jednog ili više **Profesora**.

Ove dvije tvrdnje simbolički se prikazuju ovako:



Ovakvu vezu nazivamo **vezom tipa više prema više** ili **(m:n) vezom**. Ona i sama može imati neke atributne tipove, odnosno, ponašati se kao entitetni tip. Nazovimo taj novi entitetni tip **Kolegij** i definirajmo ga ovako:

- Svaki **Kolegij** je skup predavanja iz nekog **Predmeta** koji neki **Profesor** održava u određenom vremenskom periodu.

Za ovakav entitetni tip, koji zapravo ostvaruje (m:n) vezu između drugih entitetnih tipova, reći ćemo da je **asocijativni entitetni tip**, za razliku od entitetnih tipova iza kojih stoje konkretni (stvarni ili izmišljeni) objekti, a koje zovemo **egzistencijalnim entitetnim tipovima**.

Asocijativni entitetni tip prikazat ćemo pravokutnikom u koji je upisan romb (ili njegov dio, ako za cijeli romb nema mjesta).

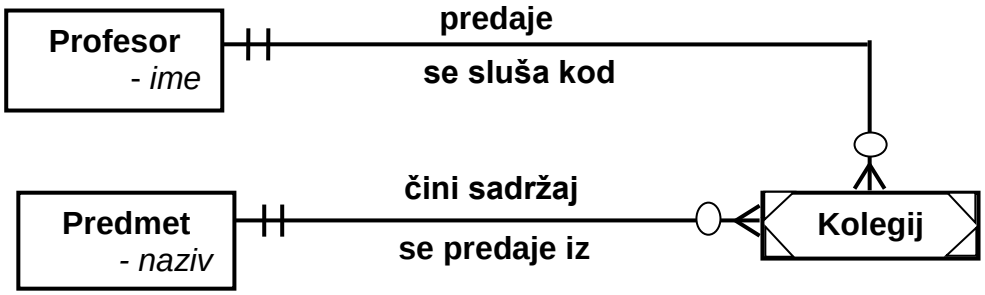
Definiranje asocijativnog entitetnog tipa **Kolegij** traži definiranje i dviju novih (1:m) veza prema entitetnim tipovima **Profesor** i **Predmet**:

- Svaki **Profesor** predaje nijedan, jedan ili više **Kolegija**. Obratno, svaki **Kolegij** se sluša kod jednog i samo jednog **Profesora**.

S druge strane, imamo:

- Svaki **Predmet** čini sadržaj nijednog, jednog ili više **Kolegija**. Obratno, svaki **Kolegij** se predaje iz jednog i samo jednog **Predmeta**.

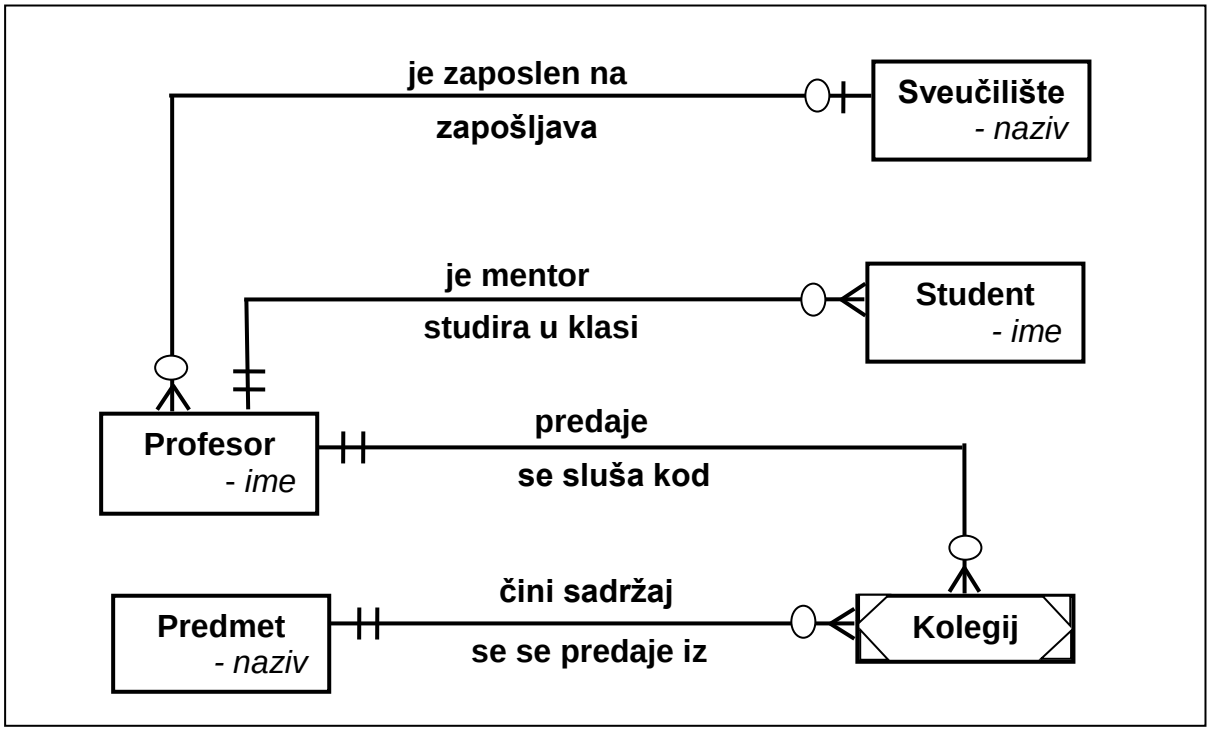
Time smo vezu (m:n) zamijenili entitetnim tipom i dvije (1;m) veze.



Entitetni tip **Kolegij** asocijativni je entitetni tip, dok su **Profesor**, **Predmet**, **Sveučilište** i **Student** egzistencijalni entitetni tipovi.

5.5. Dijagram entiteta i veza

Prikupimo li sve dosad definirane entitetne tipove zajedno s njihovim atributnim tipovima i tipovima veza te ih pregledno prikažemo na jednom dijagramu, dobit ćemo ranije spomenuti dijagram entiteta i veza, ER dijagram.



Kao što smo rekli, ER dijagram jezgra je našeg konceptualnog modela podataka, ER modela. Zaista, za rekonstrukciju formaliziranog prikaza realnog sustava dovoljno je promatrati i čitati simbole sadržane u ER dijagramu.

5.6. Prevođenje ER modela u relacijski model

Model entiteta i veza može se prevesti u relacijski model. Naime, ER dijagram lako je prevesti u skup relacija za koji se, što je od izuzetne praktične vrijednosti, može dokazati da su njegove relacije u 3NF. Postupak za to je izravan i može se svesti na nekoliko jednostavnih pravila.

Pravilo 1. Svaki entitetni tip u ER modelu postaje relacija (tablica) u relacijskom modelu.

Pravilo 2. Svaki atributni tip promatranog entitetnog tipa u ER modelu postaje atribut (stupac) odgovarajuće tablice u relacijskom modelu.

Pravilo 3. Svaka (m,1) veza od promatranog entitetnog tipa prema nekom drugom entitetnom tipu u ER modelu postaje strani ključ u promatranoj relaciji, a njegove vrijednsti odgovaraju vrijednostima primarnog ključa u relaciji koja odgovara drugom entitetnom tipu.

Pravilo 4. Svaka (m,n) veza između dva entitetna tipa u ER modelu postaje relacija sa stranim ključevima koji odgovaraju primarnim ključevima u relacijama koje odgovaraju promatranim entitetnim tipovima.

Pravilo 5. Svaki asocijativni entitetni tip u ER modelu postaje relacija sa stranim ključevima koji odgovaraju primarnim ključevima relacija odgovarajućih entitetnih tipova.

Za primarni ključ svake relacije najbolje je uzeti neki identifikator bez značenja (npr. redni broj retka u relaciji). Tom ključu obično dodijelimo naziv **id**.

Kako bismo ER model iz našeg primjera preveli u odgovarajući relacijski model, izradimo najprije dvije osnovne relacije: **Predmet(id, naziv)**, **Sveučilište(id, naziv)**.

```
create table Predmet
(
id int primary key,
naziv varchar(20)
);
insert into Predmet
(id, naziv)
values
(1, 'alkemija'),
(2, 'crna magija'),
(3, 'metafizika');

select * from Predmet;
```

Predmet	
id	**naziv**
1	alkemija
2	crna magija
3	metafizika

```
create table Sveučilište
(
id int primary key,
naziv varchar(20)
);
insert into Sveučilište
(id, naziv)
values
(1, 'Heidelberg'),
(2, 'Konigsberg');

select * from Sveučilište;
```

Sveučilište	
id	**naziv**
1	Heidelberg
2	Konigsberg

Dodajmo sada relaciju **Profesor** (**id**, **ime**, **id_sveučilišta**) u kojoj će veza **zapošljava – je zaposlen**, koja je tipa (1:m) biti realizirana pomoću stranog ključa **id_sveučilišta**.

```
create table Profesor
(
id int primary key,
ime varchar(20),
id_sveučilišta int foreign key references Sveučilište(id)
);
insert into Profesor
(id, ime, id_sveučilišta)
values
(1, 'Faust', 1),
(2, 'Wolland', 2),
(3, 'Kant', 2);

select * from Profesor;
```

id	**ime**	**id_sveučilišta**
1	Faust	1
2	Wolland	2
3	Kant	2

Dakle, Faust je zaposlen na sveučilištu sa šifrom 1, odnosno u Heidelbergu, a Wolland i Kant na sveučilištu broj 2, odnosno u Konigsbergu. Možemo se u to uvjeriti ovim upitom:

```
select Profesor.ime, Sveučilište.naziv
from Profesor, Sveučilište
where Profesor.id_sveučilišta = Sveučilište.id;
```

Izradimo sada i relaciju **Student**(**id**, **ime**, **id_profesora**) u kojoj strani ključ **id-profesora** realizira vezu tipa (1:m) **studira u klasi – je mentor** prema relaciji **Profesor**.

```
create table Student
(
id int primary key,
ime varchar(20),
id_profesora int foreign key references Profesor(id)
);
insert into Student
(id, ime, id_profesora)
values
(1, 'Wagner', 1),
(2, 'Behemot', 2),
(3, 'Azazello', 2),
(4, 'Frosch', 3);

select * from Student;
```

id	ime	id-profesora
1	Wagner	1
2	Behemot	2
3	Azazello	2
4	Frosch	3

Dakle, Wagner studira u klasi profesora sa šifrom 1, a to je Faust. Behemot i Azazello studiraju u klasi profesora sa šifrom 2, a to je Wolland i, konačno, Wagner studira u klasi profesora čija je šifra 3, a to je Kant. Možemo se u to uvjeriti ovim upitom:

```
select Student.ime, Profesor.ime
from Student, Profesor
where Student.id_profesora = Profesor.id;
```

Sve do sad definirane relacije **Predmet**, **Sveučilište**, **Profesor** i **Student** odgovarale su po jednom egzistencijalnom entitetnom tipu. Realizirajmo sada i preostali asocijativni entitetni tip

Kolegij pomoću istoimene relacije čiji će se ključ sastojati samo od dva strana ključa, **id-profesora** i **id-predmeta**.

```
drop table Kolegij;

create table Kolegij
(
id_profesora int foreign key references Profesor(id),
id_predmeta int foreign key references Predmet(id)
);

insert into Kolegij
(id_profesora, id_predmeta)
values
(1,1),
(1,2),
(2,1),
(2,2),
(2,3),
(3,3);

select * from Kolegij;
```

id-profesora	id-predmeta
1	1
1	2
2	1
2	2
2	3
3	3

Dakle, profesor sa šifrom 1, Faust, predaje predmete sa šiframa 1 i 2, alkemiju i crnu magiju. Profesor sa šifrom 2, Wolland, predaje predmete sa šiframa 1, 2 i 3, alkemiju, crnu magiju i metafiziku. Konačno, profesor sa šifrom 3, Kant, predaje predmet sa šifrom 3, metafiziku. Možemo se u to uvjeriti ovim upitom:

```
select Profesor.ime, Predmet.naziv
from Profesor, Predmet, Kolegij
where Kolegij.id_profesora = Profesor.id and
      Kolegij.id_predmeta  = Predmet.id;
```

Time su proces prevođenja ER-modela u relacijski model i njegova implementacija kao relacijske baze podataka završeni. Lako je provjeriti da su sve relacije koje smo dobili u 3NF i da se iz njih mogu rekonstruirati sve informacije o našem zamišljenom sustavu sveučilišta.

5.7. Definiranje indeksa

Indeksi su strukture koje se dodaju tablicama kako bi se ubrzalo pretraživanje. Može se definirati nad jednim ili više atributa, odnosno stupaca. Indeks je zapravo tablica razvrstanih vrijednosti promatranog stupca. Svaki redak indeksa sastoji se od neke vrijednosti promatranog stupca te ključa ili drugog pokazivača na redak indeksiranje relacije u kojem se ta vrijednost nalazi.

Prilikom pretraživanja baze podataka zapravo ne vidimo indekse, već samo ubrzanje pretraživanja u odnosu na situaciju kad indeksa nema. Pogledajmo sintaksu definiranja indeksa:

CREATE INDEX	CREATE INDEX ime_indeksa ON ime_relacije (ime_atributa)

Dobri su kandidati za definiranje indeksa strani ključevi relacije. Naime, ako njih indeksiramo, ubrzat ćemo pretraživanja kao što je, na primjer, pronalaženje svih studenata koji studiraju u klasi određenog profesora ili pronalaženje svih profesora koji su zaposleni na određenom sveučilištu i slično. Pogledajmo primjer:

```
create index index_id_profesora
on Student (id_profesora);
```

Prisutnost ovakvog indeksa bi, u slučaju da je riječ o relacijama s puno podataka, značajno ubrzalo upite kao što je 'pronađi sve studente profesora Wollanda', pri čemu onaj tko postavlja upit uopće ne mora biti svjestan prisutnosti indeksa.

```
select * from Student
where Student.id_profesora = Profesor.id and
Profesor.ime = 'Wolland';
```

Želimo li osigurati da se prilikom unosa redaka vrijednosti u indeksiranom stupcu ne ponavljaju, možemo postaviti tzv. jedinstveni indeks na taj stupac. Postavljanje jedinstvenog indeksa zapravo znači da će atribut nad kojim je on postavljen biti jedan od ključeva relacije. Opcija PRIMARY KEY navedena prilikom izrade relacije, automatski postavlja jedinstveni indeks nad stupac na koji se odnosi.

CREATE UNIQUE INDEX	CREATE UNIQUE INDEX ime_indeksa ON ime_relacije (ime_atributa)

Postavimo, na primjer, jedinstveni indeks nad skup atributa id-profesora, id-predmeta u relaciji **Kolegij**. Time ćemo osigurati da se ovaj skup atributa zaista ponaša kao ključ relacije, tj. neće biti moguće upisati dvije jednake kombinacije vrijednosti ovih atributa.

```
create unique index index_profesor_predmet
on Kolegij(id_profesora, id_predmeta);
```

Želimo li izbrisati indeks, poslužit ćemo se naredbom ovog oblika:

DROP INDEX	DROP INDEX ime_relacije.ime_indeksa

Na primjer,

```
drop index Student.index_id_profesora;
drop index Kolegij.index_profesor_predmet;
```

Brisanjem indeksa izvorne relacije ostaju nedirnute, ali se određene vrste pretraživanja izvršavaju usporeno ako se pretražuju velike količine podataka.

5.8. Sustavi za upravljanje relacijskim bazama podataka

Kao što smo rekli na početku, informacijski sustav je uvijek neka vrsta slike realnog sustava. Pri tome baza podataka, kao jezgra čitavog informacijskog sustava, odražava stanje realnog sustava, a programi nad bazom podataka odražavaju procese u realnom sustavu.

Ovi programi, međutim, ne operiraju izravno nad bazom podataka. Između njih i baze podataka postoji još jedan aplilkacijski sloj koji nazivamo **sustav za upravljanje bazom podataka**, **SUBP**. (*Data Base Management System* - DBMS).

Sustav za upravljanje bazom podataka sastoji se od komponenti koje osiguravaju da podaci budu organizirani na način koji sami odredimo, dakle, u skladu s našim modelom podataka, kao i da se u podacima ne naruše pravila integriteta koja smo odredili.

Sustav također sadrži upitni jezik kao što je SQL za neproceduralno izvršavanje operacija nad podacima te razne komponente koji omogućavaju programiranje u proceduralnim jezicima, vodeći računa o konzistentnosti podataka.

Nadalje, neki sustavi sadrže razne generatore izvještaja kao i generatore ekranskih formi za unos podataka. U jeziku SQL stvaranje početnih uvjeta za pohranjivanje podataka unutar neke baze, npr. mjesta za pohranjivanje podataka o samim relacijama, atributima i drugim objektima u bazi, tzv. **rječnik podataka**, obavlja se naredbom za izradu baze podataka, koja ima ovu sintaksu:

CREATE DATABASE	CREATE DATABASE ime_baze

Njoj suprotna naredba, koja briše sve strukture potrebne za funkcioniranje baze zajedno sa njenim podacima, ima ovakvu sintaksu:

DROP DATABASE	DROP DATABASE ime_baze

Rječnik podataka najčešće se također sastoji od relacija čiji atributi opisuju ostale objekte u bazi podataka. Različiti sustavi u pravilu imaju različita rješenja za rječnik podataka. Na primjer, u sustavu MS SQL, popis relacija dobili bismo ovakvim upitom:

```
select * from Sysobjects
where xtype = 'U';
```

Popis relacija s pripadajućim atributima dobili bismo ovakvim upitom:

```
select
  Sysobjects.name as relacija,
  Syscolumns.name as atribut
from
  Sysobjects,
  Syscolumns
where
  Sysobjects.xtype = 'U' and
  Sysobjects.id = Syscolumns.id;
```

Popis indeksa dobili bismo ovako:

```
select name from Sysindexes;
```

U drugim sustavima ovi bi upiti izgledali, naravno, drugačije.

U različitim sustavima za upravljanje bazama podataka koriste se i različiti upitni jezici, no i sustavi koji se koriste SQL-om imaju svoje vlastite inačice ovog jezika, ponešto različite od standardnog. Ipak, standardni SQL osnova je svake od ovih inačica i njegovo poznavanje omogućava lagano prilagođavanje bilo kojem danas uobičajenom sustavu za upravljanje relacijskim bazama podataka.

5.9. Pitanja za ponavljanje

1. Što su entitetni, a što atributni tipovi?
2. Što su tipovi veze?
3. U čemu je razlika između egzistencijalnih i asocijativnih entitetnih tipova?
4. Koja su pravila za prevođenje ER modela u relacijski model?
5. Što je sustav za upravljanje relacijskom bazom podataka?

Dodatak 1. Sintaktički elementi jezika SQL

Element	Sintaksa
AND / OR	SELECT popis_atributa FROM popis_relacija WHERE uvjet AND|OR uvjet
ALTER TABLE (dodavanje atributa)	ALTER TABLE ime_relacije ADD ime_atributa tip
ALTER TABLE (brisanje atributa)	ALTER TABLE ime_relacije DROP COLUMN ime_atributa
ALTER TABLE (promjena atributa)	ALTER TABLE ime_relacije ALTER COLUMN ime_atributa tip
AS (alias za atribut)	SELECT ime_atributa AS alias_atributa FROM ime_relacije
AS (alias za relaciju)	SELECT ime_atributa FROM ime_relacije AS alias_relacije
BETWEEN	SELECT popis_atributa FROM popis_relacija WHERE ime_atributa BETWEEN vrijednost1 AND vrijednost2
CREATE DATABASE	CREATE DATABASE ime_baze
CREATE INDEX	CREATE INDEX ime_indeksa ON ime_relacije (ime_atributa)
CREATE TABLE	CREATE TABLE ime_relacije (ime_atributa1 tip1, ime_atributa2 tip2, ...)
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX ime_indeksa ON ime_relacije (ime_atributa)
CREATE VIEW	CREATE VIEW ime_pogleda AS SELECT popis_atributa FROM popis_relacija WHERE uvjet
DELETE FROM	DELETE FROM ime_relacije (**Napomena:** brišu se svi retci) *ili* DELETE FROM ime_relacije WHERE uvjet
DROP DATABASE	DROP DATABASE ime_baze
DROP INDEX	DROP INDEX ime_relacije.ime_indeksa
DROP TABLE	DROP TABLE ime_relacije

GROUP BY	SELECT ime_atributa1,SUM(ime_atributa2) FROM ime_relacije GROUP BY ime_atributa1
HAVING	SELECT ime_atributa1,SUM(ime_atributa2) FROM ime_relacije GROUP BY ime_atributa1 HAVING SUM(ime_atributa2) uvjet vrijednost
IN	SELECT popis_atributa FROM popis_relacija WHERE ime_atributa IN (vrijednost1,vrijednost2,..)
INSERT INTO	INSERT INTO ime_relacije VALUES (vrijednost1, vrijednost2,....) *ili* INSERT INTO ime_relacije (ime_atributa1, ime_atributa2,...) VALUES (vrijednost1, vrijednost2,....)
LIKE	SELECT popis_atributa FROM ime_relacije WHERE ime_atributa LIKE uzorak
ORDER BY	SELECT popis_atributa FROM ime_relacije ORDER BY ime_atributa [ASC|DESC]
SELECT	SELECT popis_atributa FROM ime_relacije
SELECT *	SELECT * FROM ime_relacije
SELECT DISTINCT	SELECT DISTINCT popis_atributa FROM ime_relacije
SELECT INTO (koristi se za stvaranja kopija relacija)	SELECT * INTO ime_nove_relacije FROM ime_izvorne_relacije *ili* SELECT popis_atributa INTO ime_nove_relacije FROM ime_izvorne_relacije
TRUNCATE TABLE (briše se samo sadržaj relacije)	TRUNCATE TABLE ime_relacije
UPDATE	UPDATE ime_relacije SET ime_atributa1=nova_vrijednost1 [, ime_atributa2=nova_vrijednost2,

	`.....]` `WHERE ime_atributa=neka_vrijednost`

UNION	`SELECT popis_atributa`
	`FROM popis_relacija`
	`WHERE uvjet`
	
	`UNION [ALL]`
	
	`SELECT popis_atributa`
	`FROM popis_relacija`
	`WHERE uvjet`

WHERE	`SELECT popis_atributa`
	`FROM popis_relacija`
	`WHERE uvjet`

Dodatak 2. Formiranje tablica iz primjera

```
create table Trokut
(
vrsta varchar(20) not null,
boja varchar(20),
opseg int
);

insert into Trokut
(vrsta, boja, opseg)
values
('jednakokračan', 'plava', 3)
;

insert into Trokut
(vrsta, boja)
values
('raznostraničan', 'crvena')
;

insert into Trokut
(vrsta)
values
('jednakostraničan')
;

insert into Trokut
(vrsta, boja, opseg)
values
('raznostraničan', 'zelena', 2)
;

update Trokut
set opseg = 100
where boja = 'crvena';

update Trokut
set boja = 'bijela', opseg=34
where vrsta = 'jednakostraničan';

update Trokut
set opseg = opseg+10
where vrsta='raznostraničan' and boja='zelena';

select * into Trokut1 from Trokut;
select * from Trokut1;

alter table Trokut1
add površina decimal(4,2);
```

```
alter table Trokut1
drop površina;

alter table Krug1
alter column površina int;

delete from Trokut1
where vrsta = 'raznostraničan';

delete from Trokut1
where vrsta = 'jednakokračan' or boja = 'bijela';

truncate Krug1;

drop table Trokut1, Krug1;

select * into Trokut1 from Trokut;

update Trokut1
set vrsta = 'raznostraničan', boja = 'ljubičasta', opseg = 300
where boja = 'bijela';

update Trokut1
set vrsta = 'jednakokračan', boja = 'narančasta', opseg = 400
where boja = 'zelena';

go
create view Likovi_iste_boje as
select
Trokut.boja as boja,
Trokut.opseg as opseg_trokuta,
Krug.površina as površina_kruga
from Trokut, Krug
where Trokut.boja = Krug.boja;
go

select distinct vrsta, boja into VrstaBoja
from Trokut;
select * from VrstaBoja;

select distinct boja, opseg into BojaOpseg
from Trokut;
select * from BojaOpseg;

select distinct vrsta, opseg into VrstaOpseg
from Trokut;
select * from VrstaOpseg;

update Trokut
set boja = 'crvena'
where boja = 'zelena';
```

```
create table Lik
(
ime varchar(20) not null,
djelo varchar(30) not null,
autor varchar(30)
);

insert into Lik
(ime, djelo, autor)
values
('Wolland', 'Majstor i Margarita', 'Mihail Bulgakov'),
('Azazello', 'Majstor i Margarita', 'Mihail Bulgakov'),
('Adrian Leverkuhn', 'Doktor Faustus', 'Thomas Mann'),
('Mephisto', 'Mephisto', 'Klaus Mann'),
('Mephisto', 'Faust', 'Johann Wolfgang Goethe'),
('Wagner', 'Faust', 'Johann Wolfgang Goethe'),
('Frosch', 'Faust', 'Johann Wolfgang Goethe');

create table Kolegij
(
profesor varchar(20) not null,
predmet varchar(20) not null
);

insert into Kolegij
(profesor, predmet)
values
('Faust', 'alkemija'),
('Faust', 'crna magija'),
('Wolland', 'alkemija'),
('Wolland', 'crna magija'),
('Wolland', 'metafizika'),
('Kant', 'metafizika');

alter table Kolegij
add sveučilište varchar(20);

update Kolegij
set sveučilište = 'Heidelberg'
where profesor = 'Faust';

update Kolegij
set sveučilište = 'Konigsberg'
where profesor = 'Wolland' or profesor = 'Kant';

create table Katedra
(
profesor varchar(20) not null,
sveučilište varchar(20)
```

```
);

insert into Katedra
(profesor, sveučilište)
values
('Faust', 'Heidelberg'),
('Wolland', 'Konigsberg'),
('Kant', 'Konigsberg');

alter table Kolegij
drop sveučilište;

create table Klasa
(
student varchar(20) not null,
mentor varchar(20),
sveučilište varchar(20)
);

insert into Klasa
(student, mentor, sveučilište)
values
('Wagner', 'Faust', 'Heidelberg'),
('Behemot', 'Wolland', 'Konigsberg'),
('Azazello', 'Wolland', 'Konigsberg'),
('Frosch', 'Kant', 'Konigsberg');

alter table Klasa
drop sveučilište;

create table Predmet
(
id int(3) primary key,
naziv varchar(20)
);
insert into Predmet
(id, naziv)
values
(1, 'alkemija'),
(2, 'crna magija'),
(3, 'metafizika');

create table Sveučilište
(
id int(3) primary key,
naziv varchar(20)
),
insert into Sveučilište
(id, naziv)
values
(1, 'Heidelberg'),
```

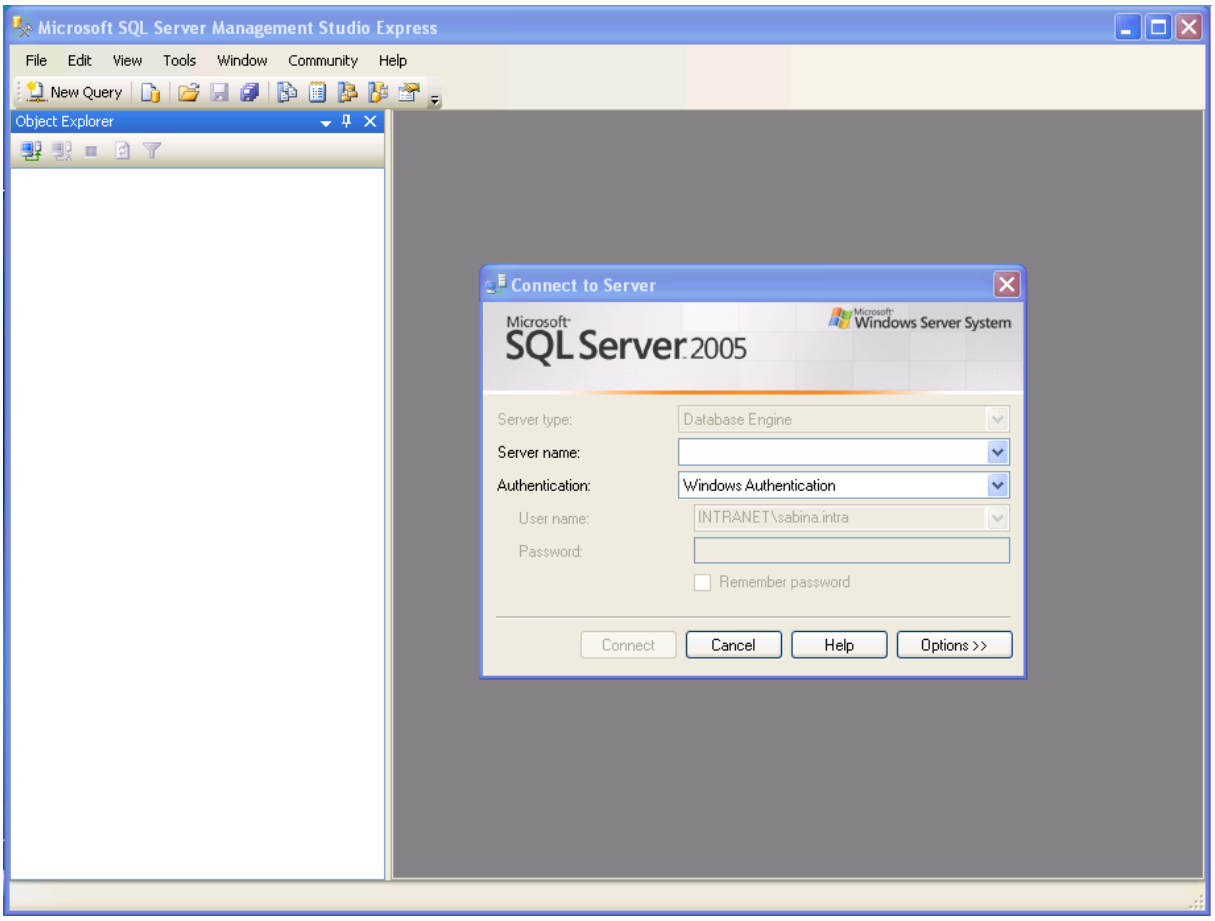
```
(2, 'Konigsberg');

create table Profesor
(
id int(3) primary key,
ime varchar(20),
id_sveučilišta int(3)
);
insert into Profesor
(id, ime, id_sveučilišta)
values
(1, 'Faust', 1),
(2, 'Wolland', 2),
(3, 'Kant', 2);

create table Student
(
id int(3) primary key,
ime varchar(20),
id_profesora int(3)
);
insert into Student
(id, ime)
values
(1, 'Wagner', 1),
(2, 'Behemot', 2),
(3, 'Azazello', 2),
(4, 'Frosch', 3);

create table Kolegij
(
id_profesora int(3),
id_predmeta int(3)
);

insert into Kolegij
(id_profesora, id_predmeta)
values
(1,1),
(1,2),
(2,1),
(2,2),
(2,3),
(3,3);

create index index_id_profesora
on Student (id_profesora);
```

Dodatak 3. MS SQL Server Management Studio Express

Pod operacijskim sustavom **MS Windows**, u sklopu sustava za upravljanje bazama podataka **MS SQL Server 2005 Express Edition**, nalazi se i korisničko sučelje za izvođenje izravnih SQL upita, **MS SQL Server Management Studio Express**. Pokreće se nizom naredbi:

Start → All programs → Microsoft SQL Server 2005 → SQL Server Management Studio Express

Pojavit će se prozor **Microsoft SQL Server Management Studio Express** i unutar njega prozor za autentikaciju, **Connect to Server**:

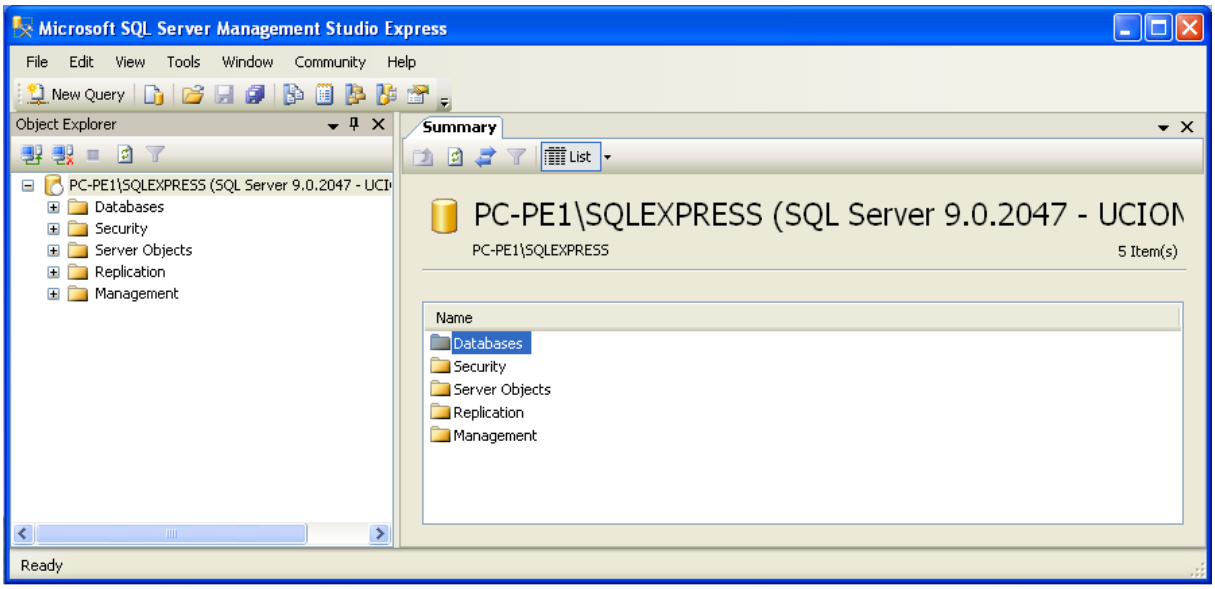


U polju ***Server name:*** odaberemo ime SQL poslužitelja preko kojega želimo pristupiti bazi podataka. Odabirom opcije *<Browse for more…>* iz padajućeg izbornika možemo dobiti popis SQL poslužitelja kojima trenutačno možemo pristupiti.

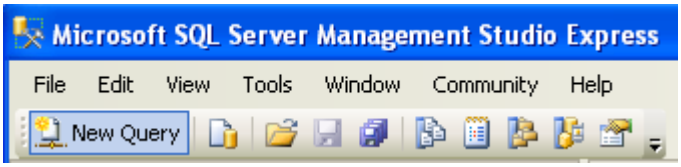
Iz padajućeg izbornika u polju ***Authentication:*** možemo odabrati način autentikacije prilikom povezivanja, *Windows authentication* ili *SQL Server Authentication*. U ovom drugom slučaju morat ćemo upisati korisničku oznaku i lozinku na razini samog SQL poslužitelja.

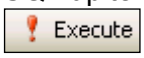
Proces povezivanja i autentikacije pokrećemo pritiskom na dugme Connect.

Ako povezivanje uspije, pokazat će nam se prozor **Microsoft SQL Server Management Studio Express**.



Za izradu upita potrebno je odabrati opciju ***New Query***.



Otvorit će se prozor za upite koji se sastoji od dva dijela: dio u koji unosimo SQL upite i dio u kojem se prikazuju rezultati. Upisani upiti izvršavaju se pritiskom na dugme 

.

Ako je označen neki upit, izvršit će se samo on.

Osim ovih osnovnih, dostupne su i razne druge naredbe i mogućnosti koje nam ovaj alat stavlja na raspolaganje.

Literatura

Alagić, Suad: Relacione baze podataka, Svjetlost, Sarajevo, 1984.

Tkalac, Slavko, Relacijski model podataka, Informator, Zagreb, 1988.

Linkovi

An Introduction to Database Normalisation,
http://dev.mysql.com/tech-resources/articles/intro-to-normalization.html

An Introduction to Structured Query Language,
http://www.techbookreport.com/sql-tut1.html

Database eLearning, http://db.grussell.org/

Intormation Technology – database language SQL,
http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

Introductory Transact SQL, http://www.developer.com/db/print.php/10920_2202631_1

SQL Course, http://www.sqlcourse.com/

SQL Course 2, http://www.sqlcourse2.com/

SQL Primer, http://www.sqlprimer.com/

SQL Reference and Example Site, http://www.fluffycat.com/sql/index.html

SQL Reference and Tutorial, http://www.ilook.fsnet.co.uk/ora_sql/sqlmain.htm

SQL Reference for SQL Server CE,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlce/htm/_lce_sql_reference_for_sql_server_ce.asp

SQL – Structured Query Language,
http://www.cs.unibo.it/%7Eciaccia/COURSES/RESOURCES/SQLTutorial/sqlcont.htm

SQL Tutorial, http://www.1keydata.com/sql/sql.html

SQL Tutorial – Learn SQL, http://www.sql-tutorial.net/

SQL Tutorial – SQL92, http://www.firstsql.com/tutor.htm

SQL Tutorial – W3C, http://www.w3schools.com/sql/

Sumarizing Data with SQL – Structured Query Language,
http://www.paragoncorporation.com/ArticleDetail.aspx?ArticleID=6

Bilješke:

Bilješke: