AKADEMIJA TEHNIČKO-UMETNIČKIH STRUKOVNIH STUDIJA BEOGRAD

ODSEK VISOKA ŠKOLA ELEKTROTEHNIKE I RAČUNARSTVA

Dr Gabrijela Dimić, dipl. ing.
Mr Miloš Pejanović

RELACIONE BAZE PODATAKA

Prvo izdanje

Beograd, 2024.

Autori:	dr Gabrijela Dimić, dipl. inž. mr Miloš Pejanović
Naziv publikacije:	RELACIONE BAZE PODATAKA
Recenzenti:	dr Danijela Milošević, redovni profesor, Fakultet tehničkih nauka u Čačku dr Perica Štrbac, profesor strukovnih studija, ATUSS, Odsek Visoka škola elektrotehnike i računarstva
Izdavač:	Akademija tehničko-umetničkih strukovnih studija Beograd, Starine Novaka 24, Beograd
Za izdavača:	dr Ana Savić, predsednik Akademije
Redni broj izdanja:	1.
Tehnička obrada:	dr Gabrijela Dimić, dipl. inž. mr Miloš Pejanović
Dizajn korica:	struk. inž. Aleksandar Ivanović
Mesto i godina izdanja:	Beograd, 2024.
Naziv i sedište štamparije:	BIROGRAF COMP D.O.O. BEOGRAD
Tiraž:	25 primeraka
Format:	B5
ISBN:	978-86-6090-162-2

CIP - Каталогизација у публикацији Народна библиотека Србије, Београд
004.652.4(075.8)
ДИМИЋ, Габријела, 1971-
Relacione baze podataka / Gabrijela Dimić, Miloš Pejanović. - 1. izd. - Beograd
: Akademija tehničko-umetničkih strukovnih studija, 2024 (Beograd : Birograf comp).
- 220 str. : ilustr. ; 25 cm
Na nasl. str.: Odsek Visoka škola elektrotehnike i računarstva. - Tiraž 25. - Bibliografija:
str. 217. - Registar.
ISBN 978-86-6090-162-2
1. Пејановић, Милош, 1963- [autor]
а) Релационе базе података

COBISS.SR-ID 161498633

PREDGOVOR

Knjiga Relacione baze podataka je namenjena studentima Akademije tehničko-umetničkih strukovnih studija u Beogradu, Odsek Visoke škole elektrotehnike i računarstva, za pripremu ispita iz predmeta Baze podataka. Takođe je korisna svima koji žele da savladaju koncepte i principe relacionih baza podataka kroz teoriju i praktične primere.

Tokom izlaganja materijala, autori su se trudili da obuhvate sve ključne koncepte relacionih baza podataka, koji čine osnovu za većinu poslovnih aplikacija. Za testiranje SQL koda korišćen je MySQL softverski sistem za upravljanje bazama podataka.

Prvo poglavlje obrađuje osnovne pojmove: podatak, informacija, baza podataka, obrada podataka, upravljanje podacima, sistem za upravljanje bazama podataka i model baze podataka. Analizira se proces evolucije i razvoja modela baza podataka od klasičnih sistema za rad sa podacima, koji su bazirani na programskim jezicima i datotekama, do savremenih baza podataka zasnovanih na softverskom sistemu za upravljanje i manipulaciju podacima.

U drugom poglavlju definišu se osnovni koncepti relacionog modela podataka. Pojmovi poput relacija, šeme relacije, šeme relacione baze podataka, atributa, domena, n-torki, instanci i null vrednosti detaljno su objašnjeni. Razmatraju se različiti tipovi ključeva, a posebna pažnja posvećuje se ograničenjima koja proizilaze iz relacionog modela podataka. Ova ograničenja uključuju integritet ključeva, integritet entiteta, referencijalni integritet i semantički integritet.

U trećem poglavlju detaljno se razmatraju matematičke operacije relacione algebre, koje su osnova upitnog jezika korišćenog za manipulaciju i rad sa relacionim bazama podataka. Definišu se matematički koncepti operacija selekcije, projekcije, preseka, unije, razlike i Dekartovog proizvoda. Takođe, definisane su operacije spajanja relacija, kojima se podaci iz dve relacije kombinuju na osnovu zajedničkih atributa ili određenih uslova.

Četvrto poglavlje detaljno obrađuje proces modelovanja i projektovanja baza podataka. Poseban naglasak stavljen je na ključne korake potrebne za kreiranje konceptualnog modela. Objašnjen je semantički model entiteta i veza na konceptualnom nivou. Razmatraju se koncepti entiteta, atributa, domena, veza između entiteta, ograničenja i tipovi veza. Definisane su i različite grafičke notacije semantičkog modela entiteta i veza.

Peto poglavlje se fokusira na objašnjenje procesa normalizacije relacionog modela podataka, s naglaskom na rešavanje problema anomalija koje nastaju zbog prisustva redundanse u podacima. Definišu se i objašnjavaju koncepti funkcionalnih zavisnosti, kao i formalne metode poznate kao normalne forme koje se primenjuju u procesu normalizacije. Detaljno su obrađene prva, druga, treća i Boyce-Coddova normalna

forma. Analizira se značaj normalizacije, kao i osnovne karakteristike denormalizacije relacionog modela podataka.

U šestom poglavlju detaljno su objašnjene SQL naredbe koje omogućavaju manipulaciju podacima iz jedne tabele relacione baze podataka. Definisani su tipovi podataka podržani u većini sistema za upravljanje bazama podataka (DBMS), kao i DDL (Data Definition Language) naredbe CREATE, ALTER i DROP, koje se koriste za kreiranje, izmenu i brisanje baza podataka i tabela. Detaljno je razrađena struktura SELECT naredbe, zajedno s ugrađenim funkcijama za rad s numeričkim, datumskim i znakovnim podacima. Posebna pažnja posvećena je agregatnim funkcijama i primeni GROUP BY i HAVING klauzula unutar njih.

U sedmom poglavlju detaljno je obrađen postupak prikaza podataka koji su izdvojeni iz dve ili više tabela. Razmatrane su karakteristike različitih metoda za povezivanje tabela. Detaljno je objašnjeno bezuslovno spajanje, spajanje po jednakosti, unutrašnje spajanje, prirodno spajanje, spoljašnje spajanje, spajanje tabele sa samom sobom, kao i različite kategorije podupita. Za svaku od ovih metoda analizirane su i istaknute specifičnosti i primene.

Osmo poglavlje je fokusirano na objašnjenje DML (Data Manipulation Language) naredbi koje omogućavaju osnovne operacije ažuriranja podataka u relacionim bazama podataka. Definisane su naredbe INSERT, UPDATE i DELETE, koje omogućavaju dodavanje, izmenu i brisanje podataka iz tabela. Razmatrani su principi implementacije akcionih SQL upita zasnovanih na prethodno navedenim naredbama, pri čemu se posebna pažnja posvećuje tome kako pravilno koristiti ove naredbe za efikasno upravljanje podacima unutar baze podataka.

U devetom poglavlju detaljno su objašnjeni ključni elementi relacione baze podataka koji su od suštinskog značaja za uspešnu implementaciju aplikacija u različitim programskim okruženjima. Definisani su koncepti pogleda, uskladištenih rutina i okidača. Obrađene su kategorije i podkategorije okidača, kao i njihova uloga u procesu modifikacije podataka. Ovo poglavlje pruža dublji uvid u napredne elemente baze podataka i omogućava čitaocima da bolje razumeju kako iskoristiti ove koncepte u razvoju svojih aplikacija.

Deseto poglavlje je fokusirano na objašnjenje strategije indeksiranja u relacionim bazama podataka. Definisani su ključni tipovi indeksa, poput klasterovanih, neklasterovanih i fulltext indeksa. Razmatrana je metodologija za definisanje različitih tipova indeksa kao i B-TREE i HASH mehanizmi za skladištenje indeksa.

U jedanaestom poglavlju detaljno je objašnjen koncept transakcije kao ključnog mehanizma za upravljanje podacima. Definisani su principi ACID modela (Atomicity, Consistency, Isolation, Durability) i istaknut značaj ovog modela u očuvanju integriteta podataka u relacionim bazama podataka. Razmatrani su problemi konkurencije transakcija i objašnjeni načini zaključavanja transakcija radi rešavanja istih. Takođe je razmotren pojam deadlock-a i strategije za njegovo sprečavanje.

Dalje, objašnjen je koncept različitih tipova izolacionih nivoa koji određuju nivo vidljivosti promena izvršenih unutar transakcije drugim transakcijama.

Autori zahvaljuju recenzentima dr Danijeli Milošević i dr Perici Štrbcu na korisnim predlozima i sugestijama koji su doprineli poboljšanju kvaliteta ovog udžbenika.

Autori će biti zahvalni svim čitaocima koji ukažu na eventualne slučajne greške, kao i na potencijalne predloge i sugestije.

U Beogradu, april 2024. Autori

SADRŽAJ

1 RAZVOJ BAZA PODATAKA

Cilj ovog poglavlja je da pruži osnovne koncepte koji su neophodni za razumevanje modela baza podataka. Detaljno su objašnjeni pojmovi kao što su podatak, informacija, baza podataka, obrada podataka, upravljanje podacima, sistem za upravljanje bazama podataka i model baza podataka. Takođe je razmotren proces evolucije i razvoja modela baza podataka. Opisan je proces poboljšanja u dizajnu modela, od sistema datoteka do objektno-relacionog modela baze podataka.

1.1 UVOD

Baze podataka i sistemi baza podataka se mogu smatrati važnim komponentama savremenog društva. Svakodnevno se realizuje znatan broj aktivnosti koje uključuju interakciju sa bazom podataka. Bankarske transakcije, online kupovina, poručivanje raznih proizvoda, rezervacija hotelskog smeštaja i turističkih aranžmana, online testiranje, anketiranje, e- učenje su samo neki od primera funkcionalnosti moderenih sistema koje su zasnovani na bazama podataka.

Ubrzan razvoj informaciono - komunikacionih tehnologija značajno utiče na generisanje velike količine podataka, a samim tim i promenu načina skladištenja podataka. Novi tipovi sistema baza podataka, često nazvani NoSQL (engl. *Not Only SQL*) sistemi, pojavili su se kao odgovor na zahteve koji su nastali pojavom velikih količina podataka kreiranih prvenstveno od strane aplikacija društvenih mreža. Ove baze podataka su dizajnirane da obezbede fleksibilne modele podataka i skalabilnost, što je ključno za upravljanje velikim količinama nestruktuiranih ili polustrukturiranih podataka. Aplikacije društvenih mreža, kao što su *Facebook*, *Twitter*, *Pinterst*, *Instagram* su zasnovane na velikim bazama podataka u kojima se čuvaju postovi, tvitovi, slike, video zapisi. *Youtube* predstavlja popularnu bazu za skladištenje i pronalaženje multimedijalnih audiovizuelnih podataka.

Evidentna je i implementacija baza podataka za prikaz informacija na veb sajtovima kao što su *Google*, *Amazon.com*, *Booking.com*, *Airbnb*, *Vikipedija*, *AccuWeathe*r. Prikupljanje i čuvanje podataka iz oblasti astronomije, genetskog inženjeringa, biohemije i mnogih drugih naučnih oblasti u predstavlja polazanu tačku mnogih naučnih istraživanja. Geografski podaci koji služi za analizu mapa, meteoroloških podataka, satelistkih snimaka skladište se u GIS sistemima (engl. *Geographic information system*) odnosno u prostornim bazama podataka. Zbog svoje sposobnosti efikasnog upravljanja i skladištenja, baze podataka imaju široku primenu u različitim oblastima.

Neke od ključnih područja primene su:

- Poslovni sistemi: Baze podataka se često koriste za upravljanje podacima o proizvodima, prodaji, korisnicima, transakcijama što predstavlja osnovu za efikasno poslovanje.
- Upravljanje ljudskim resursima (engl. Human Resources – HR): Baze podataka se koriste za čuvanje i upravljanje podacima o zaposlenima (ični podaci, radno iskustvo), platama, beneficijama i drugim relevantnim podacima za HR funkcije.
- Finansijski sektor: Ovde se baze podataka koriste za praćenje finansijskih transakcija, izveštavanje, upravljanje budžetima i slično.
- Zdravstvo: U zdravstvenom sistemu, u bazama podataka se čuvaju podaci o pacijentima, dijagnozama, lekarima, lekovima, itd.
- Obrazovanje: Univerziteti i škole koriste baze podataka za upravljanje podacima o studentima, kursevima, ocenama, predmetima, profesorima.
- Naučna istraživanja: Astronomi, genetičari, biohemičari i drugi naučnici koriste baze podataka za skladištenje, upravljanje i analizu podataka koji proizlaze iz njihovih istraživanja.
- Telekomunikacije: Baze podataka se koriste za čuvanje informacija o korisnicima, obavljenim pozivima, obračunima, servisima korisnika i sl.
- Veb aplikacije: Online kupovina proizvoda, usluga i slično podrazumeva prikaz podataka iz baze podataka. Poručivanje nekog proizvoda i ubacivanje u korpu, predstavlja dodavanje zapisa podataka u bazu podataka.
- E-trgovina: Veliki e-trgovinski sajtovi poput Amazona, eBay-a i drugih oslanjaju se na baze podataka za upravljanje inventarom, korisničkim nalozima, transakcijama i recenzijama proizvoda.
- Vladine aplikacije: Vlade koriste baze podataka za različite svrhe, uključujući evidenciju građana, poreza, pravnih dokumenata, itd.

Da bi se razumeli koncepti i tehnologije baza podataka, neophodno je definisati osnovne pojmove.

1.2 OSNOVNI POJMOVI

1.2.1 BAZA PODATAKA

Opšta definicija pojma **baza podataka** se odnosi na kolekciju povezanih podataka. Može biti gomila papira (telefonski imenik, rečnik, papirna arhiva zdrastvenih kartona pacijenata), ali je u savremenom svetu prisutna u računarskim sistemima. Prema prethodno navedenoj definiciji, skup reči koje čine jednu stranicu teksta kao povezane podatke mogu se posmatrati kao bazu podataka, što se ipak ne može smatrati ispravnim jer pojam baza podataka podrazumeva ispunjavanje određenih strožijih uslova.

U nastavku su navedene neke od definicija koje ukazuju na karaketristike baza podataka:

Baza podataka predstavljaja organizovanu kolekciju međusobno povezanih podataka, koji predstavljaju aspekte, koncepte, podskupove realnog sveta, logički su povezani na osnovu svojstvenog značenja, a koriste ih grupe korisnika i aplikacija.

Baza podataka je integrisani skup podataka o nekom sistemu i skup postupaka za njihovo održavanje i korišćenje, organizovan prema potrebama korisnika.

Baza podataka je integrisani skup podataka o nekom sistemu i skup postupaka za njihovo održavanje i korišćenje, organizovan prema potrebama korisnika.

Baza podataka je dobro struktuirana kolekcija podataka, koja postoji jedno određeno vreme, održava se i koju koristi više korisnika ili programa.

Baza podataka je kolekcija međusobno povezanih podataka, uskladištenih sa minimumom redundanse, organizovanih tako da u najkraćem vremenu daju informacije.

Baza podataka predstavlja neki aspekt stvarnog sveta, koji se ponekad naziva i minisvet. Promene u minisvetu se odražavaju u bazi podataka.

Baza podataka je dizajnirana, kreirana i popunjena podacima za određenu primenu. Ima određenu grupu korisnika i unapred osmišljene aplikacije koje su od interesa za ove korisnike.

1.2.2 PODATAK

Podatak je činjenica od značaja o nekom pojmu, objektu, pojavi, događaju. Može se zabeležiti, sačuvati, arhivirati u papirnom ili elektronskom obliku. Podatak je jednostavna neobrađena činjenica koja sama po sebi nema značenje. Nematerijalne prirode, podatak jednostavno postoji u i nema značenje unutar ili izvan svog postojanja pa se pridružuje značenju kojim se opisuju svojstva nekog objekata. Postoji u bilo kojem obliku, bio upotrebljiv ili ne. Oblici podataka su zvučni, slikovni, brojčani i tekstualni.

Po svojoj strukturiranosti podaci mogu biti nestrukturirani, kvazistrukturirani, polustrukturirani i strukturirani. Većina podataka su nestrukturirani što otežava proces njihove obrade jer zahteva mnogo znanja i vremena.

Strukturirani podaci imaju jasno definisanu strukturu i najčešće se nalaze u relacionim bazama podataka ili skladištima podataka (engl. *Data Warehouse - DW*). Predstavljaju kategoriju podataka čija je obrada, analiza i interpretacija precizno definisana i relativno jednostavna. Numerički, datumski i znakovni podaci se smatraju struktuiranim. Mogu se upotrebljavati više puta i pogodni su za automatsku obradu i procesiranje. Izvori strukturiranih podataka mogu biti mašinski (generišu ih sami uređaji) ili ljudski generisani (koji nastaju u interakciji čoveka i uređaja).

Polustrukturirani podaci se mogu definisati kao strukturirani podaci koji se ne uklapaju u formalnu strukturu modela podataka. Ne sadrže oznake koje razdvajaju semantičke elemente, nemaju zajedničku strukturu, ali imaju mogućnost prikaza hijerarhijske organizacije unutar podataka i predstvaljanja iste vrste podataka na više načina. Kvazistrukturirani podaci predstavljaju tekstualne podatke koji su dati u nestandardnom formatu tako da se mogu formatirati, ali je za to potrebno dosta vremena, alata i znanja. Nestrukturirani podaci u osnovi nemaju unapred definisani model (slike, tekst, video, zvuk i sl.).

Na primer, baza podataka turističkih aranžmanima pored struktuiranih podataka o nazivu aranžmana, ceni, datumu realizacije može da sadrže i druge vrste podataka kao što su PDF dokumente ugovora, skenirane stranice pasoša, mape, fotografije gradova, video zapis vezan za promociju samog aranžamana. U ovom primeru je prikazana mogućnost kombinovanja struktuiranih i nestruktuiranih podataka u okviru jedne relacione baze podataka sa ciljem njene implementacije na serverskoj strani online veb aplikacije.

1.2.3 INFORMACIJA

U uobičajnom govoru, termin podatak i informacija se često koriste u iste svrhe, što je pogrešno i neispravno.Treba napomenuti da ovi termini nisu sinonimi. Termin podaci (engl. *data*) obično označava skup vrednosti, odnosno sirove podatke (engl. *row data*). Nakon sažimanja, obrade i smeštanja u odgovarajući kontekst, podatak postaje informacija (engl. *information*). Kako je već unapred rečeno, podatak samo po sebi nema nikakvo značenje i predstavlja činjenicu o nekoj pojavi, objektu, događaju.

Informacija predstavlja obrađen podatak. Nastaje obradom, prevođenjem, sumiranjem ili nekom drugom vrstom procesiranja podataka. Može se posmatrati kao izdvojeno korisno, znanje iz podataka. Upotrebom odgovarajućih informacija, korisnici istih značajno uvećavaju svoje znanje i koriste u funkciji sistema za odlučivanje i donošenju poslovnih odluka.

Informacija je rezultat analize i organizacije podataka na način da daje novo znanje primaocu informacije. Postaje znanje kad je interpretirana i stavljena u kontekst ili kad joj je dodato značenje. Informaciju čine organizovani podaci kojima je dato značenje putem logičkih veza, uređeni za bolje shvatanje i razumevanje.

1.2.4 OBRADA PODATAKA

Obrada podataka obuhvata implementaciju postupaka i tehnika koje se koriste za manipulaciju, analizu i ekstrakciju sa ciljem dobijanja korisnih informacija. Omogućava efikasno upravljanje i upotrebu velike količine prikupljenih podataka. Predstavlja ključni aspekt za donošenje brzih i značajnih odluka u različitim sektorima, uključujući poslovanje, naučna istraživanja, medicinu, finansije i mnoge

druge. Obradom podataka se generišu izvještaji koji prikazuju relevantne informacije na jasan i pregledan način. Izvještaji mogu biti u formi tabela, grafova, dijagrama, ili drugih vizualnih prikaza.

1.2.5 UPRAVLJANJE PODACIMA

Upravljanje podacima obuhvata proces prikupljanja, organizacije, čuvanja, analiziranja i upotrebe podataka sa ciljem dobijanja značajnih informacija.

Kjučni aspekati upravljanja podacima u bazama podataka su:

- **Dizajn baze podataka.** Ovaj korak uključuje planiranje strukture baze podataka realizacijom postupka modelovanja, a sa ciljem generisanja modela baze podataka koji će da obezbedi integritet, konzistentnost i pouzdanost podataka.
- **Prikupljanje, unos i ažuriranje podataka.** Upravljanje procesima za unos novih podataka, izmenu postojećih te prikupljanje podataka na način koji osigurava njihovu tačnost i doslednost.
- **Integritet podataka.** Ovaj korak uključuje postavljanje ograničenja kako bi se onemogućilo unošenje netačnih podataka u bazu. Na primer, provera da li je broj unešen u polje koje je numeričkog tipa, a ne tekstualnog.
- **Sigurnosne mere.** Upravljanje pravima pristupa podacima kako bi se omogućilo da samo određeni tipovi korisnika imaju mogućnost rada sa određenim podacima u bazi.
- **Kreiranje kopija i oporavak podataka.** Redovno sigurnosno kopiranje podataka, zaštita od gubitka podataka zbog kvarova, prirodnih katastrofa ili ljudskih grešaka, kreiran plan oporavka podataka.
- **Održavanje performansi baze podataka.** Praćenje performansi baze podataka, identifikacija i rešavanje problema koji mogu usporiti pristup podacima, te optimizacija upita i struktura baze za bolje performanse.
- **Skaliranje baze podataka**. Upravljanje rastom i povećanjem opterećenja baze podataka. To može uključivati vertikalno skaliranje (poboljšanje hardverskih resursa) ili horizontalno skaliranje (dodavanje više servera).
- **Implementacija sigurnosnih procedura.** Uspostavljanje procedura kojim se obezbeđuje sigurnost podataka, koje uključujući enkripciju podataka, kontrolu pristupa i praćenje aktivnosti korisnika.
- **Automatizacija i planiranje zadatka.** Korišćenje automatizacije za rutinske zadatke kao što su sigurnosna kopiranja, ažuriranja i optimizacija baze podataka.
- **Nadzor nad sistemom.** Redovno praćenje performansi, sigurnosti i dostupnosti baze podataka kako bi se identifikovali i rešili svi potencijalni problemi.
- **Upravljanje verzijama podataka.** Praćenje i upravljanje promenama u strukturi baze podataka, uključujući nadogradnju i izmene.

Ovi elementi zajedno čine ključne komponente u upravljanju podacima u bazama podataka i obezbeđuju da organizacija može koristiti dobijene relevantne informacije u funkciji sistema za donošenje poslovnih odluka.

1.3 SISTEM ZA UPRAVLJANJE BAZAMA PODATAKA

U uobičajenom govoru, pojam baze podataka se odnosi na kolekciju podataka koji se skladište jedno određeno vreme, a njihovo manipulisanje i upravljanje se realizuje upotrebom specijalizovanih sistema za upravljanje bazom podataka.

Sistem za upravljanje bazama podataka (engl. *Database Management System - DBMS*) je softverski alat koji omogućava upravljanje bazom podataka. Olakšava proces definisanja, kreiranja, manipulacije i deljenja baza podataka među različitim korisnicima i aplikacijama. Ove aktivnosti mogu obuhvatiti jednostavno pretraživanje podataka do definisanja šema i strukture baze podataka. Osnovni cilj upotrebe *DBMS*-a u podrazumeva obezbeđivanje najefikasnijeg način za smeštanje, obradu podataka i izdvajanje informacija od značaja. Osim toga, *DBMS* omogućava korisnicima sigurno i paralelno upravljajanje bazom podataka, bez mešanja sa radom drugih korisnika i uz očuvanje integriteta podataka.

Kao osnovne funkcije *DMBS*- a mogu se izdvojiti procesi definisanja, kreiranja i manipulacije.

- Definisanje obuhvata postupak specifikacije tipova, struktura i ograničenja podatke koji se skladište u bazi podataka.
- Kreiranje podrazumeva postupke kojima se obezbeđuje čuvanje podataka na nekom medijumu koji kontroliše DBMS.
- Manipulisanje podrazumeva postavljanje upita za upis novih podataka, izdvajanje i prikaz postojećih podataka, kao i za ažuriranje i brisanje skladištenih podataka.

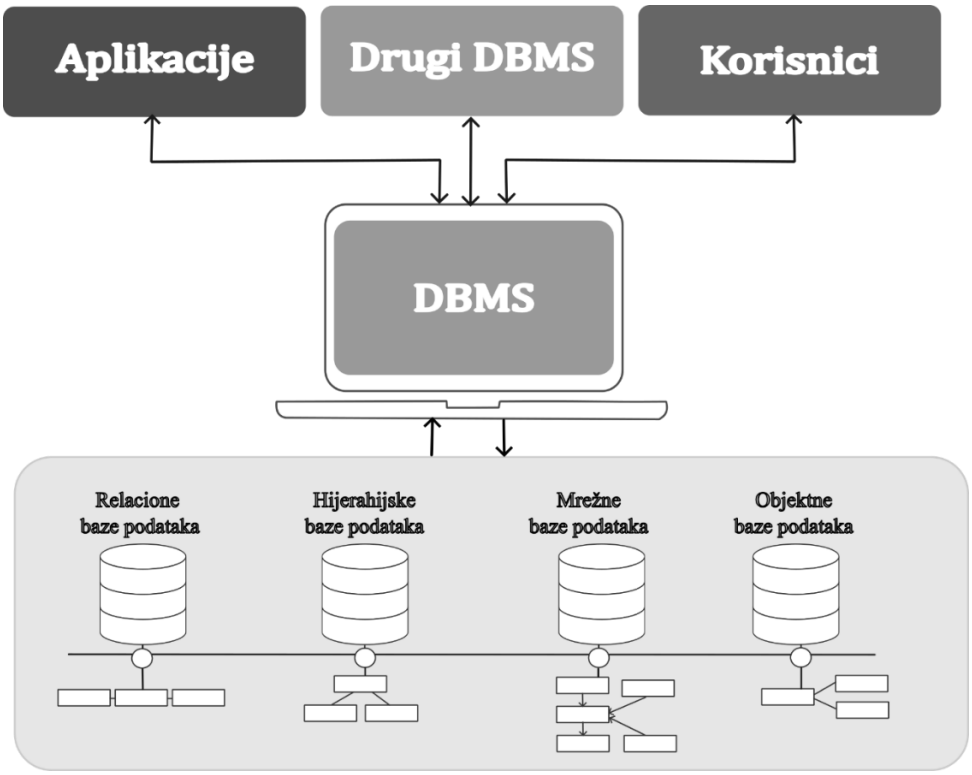
Tipični administratorski zadaci baze podataka koji se mogu izvršavati koristeći *DBMS* uključuju:

- **Konfigurisanje autentifikacije i autorizacije**. Konfigurisanje korisničkih naloga, definisanje politike pristupa, postavljanje i modifikovanje ograničenja i prava pristupa. Ove operacije omogućavaju administratorima da ograniče pristup osnovnim podacima, kontrolišu korisničke akcije i upravljaju korisnicima u bazama podataka.
- **Kreiranje rezervnih kopija i snimaka podataka.** DBMS može pojednostaviti proces pravljenja rezervnih kopija baza podataka pružajući jednostavan i direktniji interfejs za upravljanje rezervnim kopijama i snimcima. Mogu premestiti rezervne kopije na lokacije trećih strana poput skladišta u obalaku radi poboljšanja sigurnosti.

- **Optimizacija performansi**. DBMS može pratiti performanse baza podataka koristeći integrisane alatke i omogućiti korisnicima da podešavaju baze podataka kreiranjem optimizovanih indeksa. To smanjuje korišćenje I/O resursa radi optimizacije SQL upita, omogućavajući najbolje performanse bazi podataka.
- **Oporavak podataka.** U operaciji oporavka, DBMS pruža platformu za obnovu sa neophodnim alatkama za potpunu ili parcijalnu obnovu baza podataka u njihovo prethodno stanje.

Realizacija navedenih administrativnih zadataka olakšana je kroz efikasan i smislen upravljački interfejs. Većina modernih *DBMS* podržava upravljanje višestrukim opterećenjima baza podataka iz centralizovanog *DBMS* softvera, čak i u scenariju distribuiranih baza podataka. Osim toga, omogućavaju organizacijama organizovan pregled svih podataka, korisnika, grupa, lokacija.

Na slici 1.3.1 je prikazana šema sistema za upravljanje bazom podataka (DBMS).



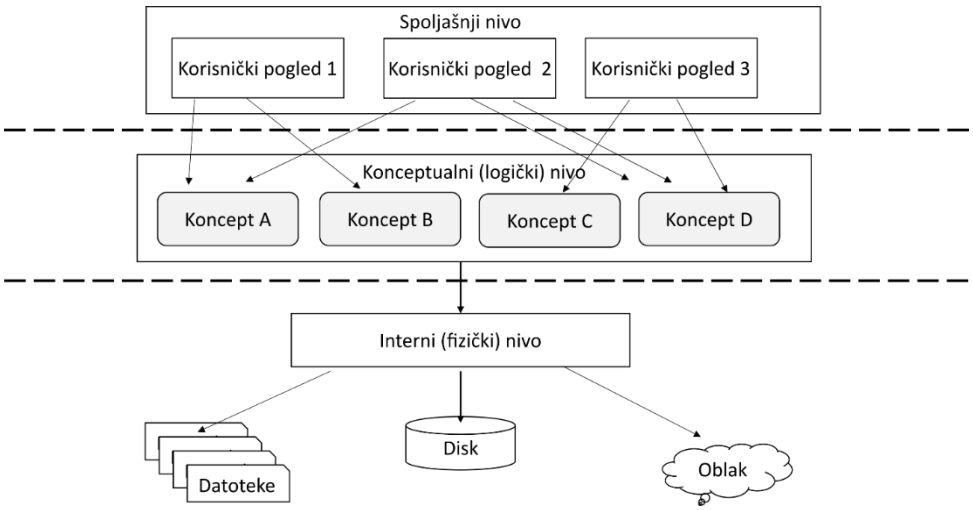
Slika 1.3.1. Šema sistema za upravljanje bazom podataka (DBMS)

Godine 1975. – te je predložena arhitektura *ANSI-SPARC* (engl. *American National Standards Institute, Standards Planning And Requirements Committee*) kao apstraktan standard dizajna za sistem upravljanja bazama podataka. Ovim standardom *DBMS* je zamišljen kao višeslojni sistem sa troslojnom arhitekturom

apstrakcije. Glavni cilj je da se korisnicima pruži apstraktan pogled na podatke kako bi se sakrili detalji o tome kako se podaci skladište i održavaju. *ANSI-SPARC* arhitektura se fokusira na razdvajanje podataka na različitim nivoima apstrakcije kako bi se omogućila efikasna upotreba i održavanje baza podataka.

Osnovni nivoi apstrakcije u ovoj arhitekturi uključuju spoljašnji (korisnički), konceptualni (logički) i unutrašnji (fizički) nivo. Svaki od nivoa ima različite zadatke. Promene na jednom novou imaju minimalan uticaj na ostale. Nezavisnost između nivoa omogućava jednostavniji razvoj aplikacija i promenu platforme.

Na slici 1.3.2 je prikazana *ANSI –SPARC* troslojna arhitektura apstrakcije podataka.



Slika 1.3.2. ANSI –SPARC troslojna arhitektura apstrakcije podataka.

Spoljašnji nivo (External Level): Ovaj nivo se odnosi na perspektivu korisnika ili aplikacije koja interaguje sa bazom podataka. Formira se na osnovu konceptualnog nivoa i definiše kako će korisnici videti podatke. Svaki spoljašni nivo predstavlja deo baze podataka relevantan za određenu grupu korisnika ili aplikacija. Omogućava korisnicima da rade sa podacima na način koji je specifičan za njihove potrebe, bez potrebe da znaju detalje o internoj strukturi baze podataka. Može se smatrati da se spoljašni (korisnički) nivo sastoji od skupa više korisničkih pogleda. Korisnički pogled predstavlja pogled na jedan deo baze podataka koji je od interesa određenom korisniku. Definisanje korisničkih pogleda je neophodno radi određivanja prava pristupa delovima baze podataka.

Konceptualni nivo (Conceptual Level): Ovaj nivo predstavlja globalni pogled na celokupnu strukturu baze podataka. Prikazuje logički model odnosno definiše kako su podaci organizovani i međusobno povezani unutar cele baze podataka. Ne zavisi od specifičnosti aplikacija i korisnika, već predstavlja objektivan pogled na podatke.

Interni nivo (Internal Level): Ovaj nivo se odnosi na fizičku organizaciju skladištenih podataka na različitim medijima. Predstavlja sadržaj cele baze podataka odnosno definiše se kako se podaci čuvaju, pristupaju i indeksiraju na niskom nivou. Ovde se razmatraju detalji oko zapisa i organizacije podataka na disku i u memoriji, kao i detalji vezani za performanse, upravljanje prostorom i sigurnost podataka.

Ova tri nivoa čine *ANSI-SPARC* arhitekturu apstrakcije. Razdvajanje na ove nivoe omogućava efikasnu organizaciju i upravljanje bazama podataka tako da se omogući jednostavan pristup podacima na različitim nivoima kompleksnosti, bez potrebe za poznavanjem unutrašnjih detalja baze podataka.

1.3.1 KOMPONENTE SISTEMA ZA UPRAVLJANJE BAZOM PODATAKA

Svi *DBMS*-ovi dolaze sa različitim integrisanim komponentama i alatima neophodnim za obavljanje zadataka upravljanja bazom podataka. Neki *DBMS* softveri pružaju mogućnost proširenja osnovne funkcionalnosti integrisanjem sa alatima i uslugama trećih strana, direktno ili putem dodataka.

Osnovne komponente *DBMS* softvera su:

- **Komponenta za skladištenje** (engl. *Storage engine*). Predstavlja osnovnu komponenta sistema za upravljanje bazom podataka (*DBMS*). Komunicira sa sistemom datoteka na nivou operativnog sistema kako bi skladištila podatke. Svi *SQL* upiti koji komuniciraju sa osnovnim podacima prolaze kroz komponentu za skladištenje.
- **Upitni jezik** (engl. *Query language*). Za interakciju sa bazom podataka neophodan je jezik za pristup podacima, počevši od kreiranja baza podataka pa do jednostavnog upisa ili prikaza podataka. *DBMS* mora podržavati jedan ili više jezika upita i dijalekata jezika. Funkcionalnost upitnog jezika se može dodatno klasifikovati prema specifičnim zadacima:
    - **Jezik za definisanje podataka** (engl. *Data Definition Language - DDL*). Obuhvata komande za definisanje šema baze podataka ili za modifikaciju strukture objekata u bazi podataka.
    - **Jezik za manipulaciju podataka** (engl. *Data Manipulation Language - DML*). Obuhvata komande za rad sa podacima u bazi podataka. Sve *CRUD* (engl. *Create, Read, Update, Delete*) operacije spadaju pod *DML*.
    - **Jezik za kontrolu podataka** (engl. *Data Control Language - DCL*). Podrazumeva komande rad sa dozvolama i drugim kontrolama pristupa bazi podataka.
    - **Jezik za kontrolu transakcija** (engl. *Transaction Control Language - TCL*). Predstavljaju komande koje se bave internim transakcijama baze podataka.

- **Procesor upita** (engl. *Query processor*). Smatar se posrednikom između upita korisnika i baze podataka. Procesor upita interpretira upite korisnika i pretvara ih u izvršne komande koje baza podataka može razumeti kako bi izvršila odgovarajuću funkcionalnost.
- **Optimizaciona komponenta** (engl. *Optimization engine*). Omogućava *DBMS*-u da pruži uvid u performanse baze podataka u smislu optimizacije same baze podataka i upita. Kada se koristi sa alatima za nadgledanje baze podataka, predstavlja moćan skup alatki za postizanje najboljih performansi baze podataka.
- **Rečnik podataka - katalog metapodataka** (engl. *Metadata catalog*). Predstavlja centralizovani katalog svih objekata u bazi podataka. Kada se objekat kreira, *DBMS* čuva zapis tog objekta sa određenim metapodacima (podaci o podacima) koristeći rečnik podataka. Zapis se može koristiti za proveru zahteva korisnika prema odgovarajućim objektima u bazi podataka i za pregled kompletne strukture baze podataka.
- **Upravljač dnevnikom** (engl. *Log manager*). Ova komponenta čuva sve zapise o pristupu bazi podataka. Ovi dnevnici uključuju prijave i aktivnosti korisnika, funkcije baze podataka, funkcije rezervnih kopija i obnove, itd. Upravljač dnevnikom osigurava da su svi ovi dnevnici pravilno zabeleženi i lako dostupni.
- **Alati za izveštavanje i nadgledanje** (engl. *Reporting and monitoring tools*). Predstavljaju još jednu standardnu komponentu koja dolazi sa *DBMS*-om. Alati za izveštavanje omogućavaju korisnicima generisanje izveštaja dok alati za praćenje omogućavaju nadgledanje baza podataka u pogledu potrošnje resursa, aktivnosti korisnika, itd.
- **Alati za manipulaciju podacima** (engl. *Data utilities*). Većina *DBMS* softvera dolazi sa dodatnim ugrađenim alatima za pružanje funkcionalnosti kao što su: provere integriteta podataka, kreiranje rezervne kopije, oporavak baze podataka, provere podataka.

1.3.2 TIPOVI SISTEMA ZA UPRAVLJANJE BAZOM PODATAKA

Postoji mnogo različitih tipova sistema za upravljanje bazom podataka (*DBMS*), ali najčešće korišćeni se mogu svrstati u tri osnovne kategorije.

Relacioni sistemi za upravljanje bazama podataka (*RDBMS*)

Ovo je najčešći tip *DBMS*-a. Koriste se za interakciju sa bazama podataka koje sadrže strukturirane podatke u tabelarnom formatu sa unapred definisanim relacijama. Osim toga, koriste struktuirani jezik upita (*SQL*) za interakciju sa bazama podataka.

Najpopularniji *DBMS*-ovi koji spadaju u ovu kategoriju su:

- *MySQL* je besplatan i otvorenog koda. Dobro podržava *SQL* jezik i pogodan je za manje do srednje baze podataka.

- *PostgreSQL* je besplatan i otvorenog koda. Ima napredne funkcionalnosti i proširenja. Odlična podrška za kompleksne upite.
- *Microsoft SQL Server* je dobro integrisan sa *Microsoft* ekosistemom. Poseduje moćne funkcionalnosti za upravljanje podacima i analitiku. Neophodna je komercijalna licenca, ali postoji besplatna verzija (*SQL Server Express*).
- *Oracle Database* je izuzetno skalabilan i moćan *DBMS*. Visoki nivo sigurnosti i integriteta podatak uslovlja izuzetno veću cenu komercijalne licence.

Sistemi za upravljanje dokument bazama podataka (*DoDBMS*)

DoDBMS sistemi se koriste za upravljanje bazama podataka koje sadrže podatke smeštene u strukturama sličnim JSON-u, sa ograničenom ili bez relacijske strukture. Pokreću se upitnim jezicima poput jezika za upite *MongoDB* (*MQL*) za operacije nad bazama podataka. *MongoDB, Azure Cosmos DB* su neki od istaknutih primera *DoDBMS*-a.

Sistemi za upravljanje kolonski orjentisanim bazama podataka kolona

Kao što naziv sugeriše, ovaj tip *DBMS*-a se koristi za upravljanje bazama podataka koje skladište podatke u kolonama umesto redovima, naglašavajući visoke performanse. Neke baze podataka koje koriste ovaj format čuvanja podataka su *Apache Cassandra, Apache HBase,* itd.

1.3.3 PREDNOSTI SISTEMA ZA UPRAVLJANJE BAZAMA PODATAKA

DBMS je uveden kako bi rešio osnovne probleme koji su bili prisutni čuvanju, upravljanju, pristupanju, ažuriranju podataka u tradicionalnim sistemima datoteka.

Efikasan pristup podacima. *DBMS* poseduje optimizovane mehanizme za čuvanje, pretraživanje i ažuriranje podataka, što može značajno ubrzati generisanje informacijama.

Integritet podataka. DBMS uključuje mehanizme za održavanje integriteta podataka, sprečavajući pojavu neusaglašenosti ili nevažećih vrednosti u bazi.

Minimalna redudansa podataka. DBMS pomaže u održavanju konzistentnosti podataka tako što se izbegava čuvanje istih podataka na različitim mestima.

Apstrakcija i nezavisnost podataka. *DMBS* obezbeđuje nezavisnost između aplikacija i podataka. Omogućen je prenos podataka na druge računarske sisteme bez potrebe za izmenom programskog koda aplikacije. *DBMS* omogućava korisnicima da menjaju fizičku šemu baze podataka bez promene logičke šeme koja upravlja odnosima u bazi podataka. Kao rezultat, organizacije mogu proširiti osnovnu infrastrukturu baze podataka bez uticaja na operacije baze podataka. Osim toga, bilo kakva promena u logičkoj šemi može se sprovesti i bez uticaja na aplikacije koje pristupaju bazama podataka.

Bezbednost i kontrola podataka. *DBMS* pruža mogućnost kontrole korisnika i primenu politika za upravljanje sigurnošću i usklađenosti definisanjem različitih nivoa pristupa podacima.. Obezbeđuje mehanizme za zaštitu podataka od neovlašćenog pristupa i omogućava enkripciju podataka kako bi se osigurala privatnost.

Konkurentan pristup i deljenje podataka. *DBMS* omogućava korisnicima siguran pristup bazi podataka bez obzira na njihovu lokaciju. Na taj način, mogu brzo obavljati sve zadatke povezane sa bazom podataka bez potrebe za kompleksnim metodama pristupa ili brige o sigurnosti baze podataka. Osim toga, omogućava deljenje podataka između različitih aplikacija i korisnika. Upravlja konkurentnim pristupom podacima od strane više korisnika.

Integracija podataka. *DBMS* korisnicima omogućava centralizovan prikaz baza podataka rasprostranjenih na više lokacija i njihovo upravljanje putem jednog interfejsa umesto rada sa njima kao zasebnim entitetima.

Efikasan mehanizam za rezervne kopije i obnovu. Većina baza podataka ima ugrađene alate za rezervne kopije i obnovu. Ipak, *DBMS* nudi centralizovane alate koji olakšavaju rezervne kopije i obnovu, omogućavajući bolje korisničko iskustvo. Obezbeđivanje sigurnosti podataka je olakšano sa funkcionalnostima kao što su: automatski snimci, planiranje rezervnih kopija, provere rezervnih kopija, višestruki načini obnove.

Uniformno upravljanje i nadgledanje. *DBMS* pruža jedan interfejs za obavljanje svih zadataka upravljanja i nadgledanja, čime se pojednostavljuje rad administratora baza podataka. Ovi zadaci mogu obuhvatiti kreiranje baze podataka, modifikaciju šema i izveštavanje i reviziju.

Ove prednosti čine DBMS ključnim alatom za efikasno i pouzdano upravljanje podacima u organizacijama svih veličina.

1.3.4 NEDOSTACI SISTEMA ZA UPRAVLJANJE BAZAMA PODATAKA (DBMS)

Iako *DBMS* sistemi donose mnoge prednosti, postoje i određeni nedostaci koji se mogu javiti u određenim situacijama:

- **Troškovi**. Implementacija i održavanje *DBMS*-a može biti skupo, uključujući licenciranje softvera, nabavku hardverske opreme i obuku osoblja.
- **Performanse.** U nekim slučajevima, *DBMS* može imati nešto sporiji pristup podacima u odnosu na direktni rad sa datotekama, posebno u situacijama kada je brzina od presudnog značaja.
- **Složenost.** *DBMS* može biti složen za konfiguraciju i upravljanje, što zahteva određeno vreme za učenje i osposobljavanje osoblja.

- **Zavisnost od proizvođača.** Kada se koristi specifičan *DBMS*, organizacija može postati zavisna od proizvođača tog *DBMS*-a, što može ograničiti fleksibilnost i prelazak na druge tehnologije.
- **Kompatibilnost i interoperabilnost**. Mogu postojati izazovi u vezi sa kompatibilnošću i interoperabilnošću između različitih *DBMS*-a ili sa drugim sistemima i aplikacijama.

Sistem za upravljanje bazom podataka (*DBMS*) je ključna komponenta za svaku organizaciju kada je reč o upravljanju bazama podataka. Kompleksnost i skup funkcija *DBMS*-a zavise od specifičnog *DBMS*-a i zahteva organizacije.S obzirom na to da različiti *DBMS*-ovi pružaju različite skupove funkcija, od velike je važnosti da organizacije pažljivo ocene softver DBMS-a pre nego što se opredele za jedan sistem. Pravilno konfigurisan *DBMS* će značajno pojednostaviti upravljanje i održavanje baza podataka bilo koje veličine.

1.4 RAZVOJ MODELA PODATAKA

Postoji mnogo preciznih definicija modela podataka. Model podataka opisuje strukturu baze podataka, uključujući kako su podaci definisani i prikazani, odnose među podacima i ograničenja. Može se koristiti da opiše organizovan i uređen skup podataka smešten na računaru.Ovaj uređeni skup često se strukturira primenom postupka za modelovanje na način koji čini izdvajanje i izmene podataka što efikasnijim. Zavisno od specifičnosti realizovanog sistema, struktura baze podataka može biti prilagođena kako bi omogućila što efikasnije upravljanje i manipulisanje podacima. Različiti modeli podataka koji su prethodili relacionom modelu su bili delimična rešenja za konstantan problem u vezi efikasnog skladištenja podataka. Trenutno, i dalje, relacioni model podataka predstavlja najbolje rešenje kako za skladištenje, tako i za prikaz podataka.

1.4.1 SISTEMI DATOTEKA

Koncept baza podataka se počeo razvijati u ranim 1950. - im godinama. Prve baze podataka su bile statičke i skladištene na magnetnim trakama. Tokom 1960. te godine, dominantnu ulogu su imali sistemi zasnovani na datotekama. Početno rešenje gotovo da i nije bilo baza podataka: to su bile datoteke poznate kao „ravna datoteka“ (engl. *flat file)*. Mogu se posmatrati kao kontejner za čuvanje podataka. Uobičajno su smeštene na uređaju za skladištenje u računarskom sistemu. Upravljanje ovim datotekama podrazumeva sistem zasnovan na konceptu operativnog sistema i bez primene tehnika modelovanja. Podaci se mogu pronaći na osnovu pozicije u datoteci. Svako dodavanje, pretraživanje, izmena podataka mora biti eksplicitno programirana. Podaci mogu biti smešteni u pojedinačne ili u više datoteka. Slično kao i pretraživanje, povezivanje podataka i provere između različitih datoteka realizovano je primenom programskih jezika. U slučaju povećavanja broj datoteka i količine podataka, proces upravljanja podacima je značajno otežan.

Objašnjenje karakteristika sistema zasnovanog na datotekama je simulirano primenom *Python* programskog jezika za skladištenje, obradu i upravljanje podacima na primeru kompanije za izradu delova za dečije igračke. Standardne funkcionalnosti aplikacije treba da obuhvate unos, prikaz i izmenu podataka o dobavljačima, delovima i zalihama.

U prilogu 1.4.1 se može videti da je postupak unosa podataka izvršen zajedno sa definicijom strukture niza koja se koristi za skladištenje podataka, i to u samom programskom kodu aplikacije.

```
Dobavljac = [];
Deo=[];
Zalihe = [];
Dobavljac.append({"sid":0,"naziv":"ABB","drzava":"SR"});
Dobavljac.append({"sid":1,"naziv":"ACC","drzava":"CZ"});
Deo.append({"pid":0,"naziv":"A"});
Deo.append({"pid":1,"naziv":"B"});
Zalihe.append((0,0));
Zalihe.append((0,1));
Zalihe.append((1,0));
```

Prilog 1.4.1 Unos podataka primenom Python programski kod

Aplikacija treba da omogući prikaz podataka na osnovu postavljenog uslova. Primer je dat za prikaz izveštaja o dobavljačima delova A (Prilog 1.2) i za prikaz svih delova za dobavljača ABB (Prilog 1.3):

```
for (sid,pid) in Zalihe:
  if pid==0:
    print (Dobavljac[sid]);
```

Prilog 1.4.2 Prikaz podataka primenom Python programski kod

```
for dobavljaci in Dobavljac:
    if dobavljaci["naziv"] == "ABB":
        for (sid, pid) in Zalihe:
            if sid == dobavljaci["sid"]:
                print(Deo[pid])
```

Prilog 1.4.3 Prikaz podataka primenom Python programski kod

Postavljaju se sledeća pitanja:

- Šta se dešava sa podacima kada se računar sa programskim kodom isključi?
- Kako obezbediti obradu količine podataka koja je veća od memorijskog kapaciteta računara?
- Šta ako se u nekom trenutku promeni format čuvanja podataka?
- Da li kôd iz Priloga 1.3 obezbeđuje optimalne performanse aplikacije?
- Na koji način se ažuriraju podaci?
- Šta ako više korisnika istvoremeno ažurira iste podatke?
- Šta ako sistem „padne“ usred izvršavanja programa?

Prethodno navedena pitanja ukazuju na probleme i nedostatke tipične za sisteme zasnovane na datotekama i programskim jezicima.

Zavisnost podataka. Prvi nedostatak se odnosi na postojanje zavisnosti između programa i podataka. Podaci se čuvaju u određenom formatu ili strukturi u datoteci. Promena u strukturi ili formatu podataka uslovljava i niz promena u postojećim aplikacionim programima koji pristupaju datotekama sa ažuriranim podacima.

Pristup podacima. Datoteke same po sebi nemaju mehanizam za izdvajanje podataka. Pristup podacima je moguć programskim putem, primenom aplikacionih programa u okviru kojih bi trebalo predvideti različite načine prikaza i izdvajanja podataka.

Redudansa podataka. Redudansa podrazumeva pojavu istih podataka (duplikata) na različitim mestima. U primeru aplikacije kompanije duplikati mogu da se jave kod unosa delova za istog dobavljača u dve različite datoteke. Ukoliko korisnički scenario funkcionisanja sistema podrazumeva da različiti dobavljači dobavljaju iste delove to znači da se podaci o delovima mogu ponavljati na različitim mestima u različitim datotekama. Ovi primeri ukazuju na redundantnost koju je teško izbeći u sistemu datoteka. Pojava redudanase u podacima dovodi do zauzimanje više prostornih resursa, a svakako i do pojave neusaglašenosti u podacima.

Nedoslednost podataka. Nedoslednost se javlja u slučaju kada isti podaci koji se nalaze na različitim mestima nisu usaglašeni. Na primer, ako se podaci o dobavljačima nalaze u dve datoteke, promena adrese dobavljača podrazumeva promene u koloni "adresa" u obe datoteke. U slučaju da se promene ne izvrše u jednoj od datoteka, adresa dobavljača će biti različita na dva mesta u sistemu. Postojenje različite vrednosti za isti podatak u različitim delovima sistema može biti rezultat grešaka u unosu, ažuriranju ili sinhronizaciji podataka. Nedoslednost podataka može dovesti do problema u tačnosti i pouzdanosti sistema.

Izolacija podataka. S obzirom da uspostavljanje relacija i veza nije podržano u sistemu datoteka, neophodni su posebni programi kojim će se obezbediti mapiranje povezanosti podataka. U kompleksnom sistemu kada se datoteke generišu od strane različitih osoba u različito vreme, može se smatrati da se kreiraju izolovano i da mogu

biti različitih formata. Kao posledica ovoga, teško je napisati nove aplikacione programe za izdvajanje podataka iz različitih datoteka održavanih na različitim mestima, jer je potrebno razumeti osnovnu strukturu svake datoteke.

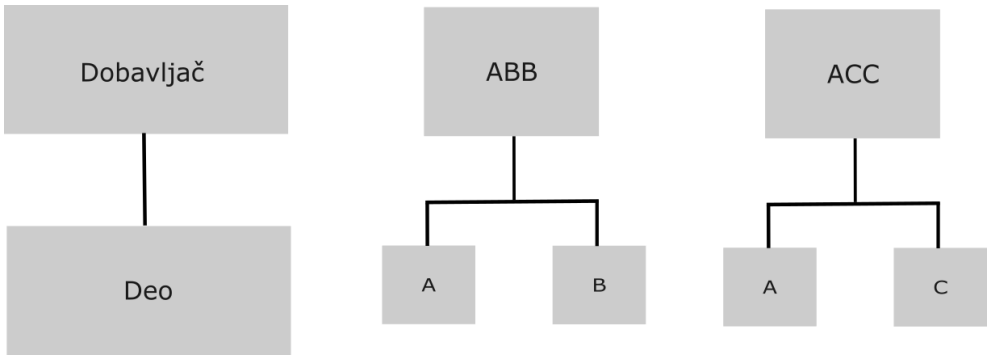
Deljenje podataka. Upotreba sistema datotekama podrazumeva realizaciju svake funkcionalnosti kao posebnog procesa koji obuhvata određene datoteke. Posledica prethodno navedenog se ogleda u nemogućnosti deljenja podataka između korisnika koji pristupaju sistemu kroz odvojene procese. Vrlo je teško sprovesti kontrolu pristupa u sistemu datoteka putem aplikacionih programa.

1.4.2 HIJERARHIJSKI MODEL

Hijerarhijski model podataka je bio popularan u prvim decenijama razvoja baza podataka. Razvijen je u ranim 1960.-ima od strane IBM-a za njihov sistem IMS (eng. *Information Management System*). Ključna karakteristika modela je hijerahijska struktura u obliku stabla. Podaci su organizovani u osnovne jedinice koji predstavljaju zapise (slogove). Veza između zapisa je zasnovana na relaciji roditelj - dete. Svaki dete zapis ima tačno jednog roditelja, osim korenog koji je na vrhu hijerarhije i nema roditelja. Zapisi koji predstavljaju roditelje mogu imati više dece zapisa. Povezivanje zapisa roditelj – dete realizovano je preko pokazivača koji ukazuju na memorijsku lokaciju u kojoj se nalazi određeni podatak.

Hijerahijska struktura modela predstavlja implementaciju veze jedan – prema – više. To znači da svaki zapis dete ima jednog roditelja, a svaki zapis roditelj može imati više zapisa dece. Zapisi dece su potpuno zavisni od nadređenih zapisa roditelja, odnosno zapis dete može postojati samo ako postoji njegov odgovarajući roditeljski zapis. Iz toga proizilazi da se podaci u zapisu dete mogu pojaviti ako postoji odgovarajući unosi u roditeljskom zapisu.

Na slici 1.4.2.1 je dat šematski prikaz hijerahijskog modela za primer kompanije za izradu delova za dečije igračke.



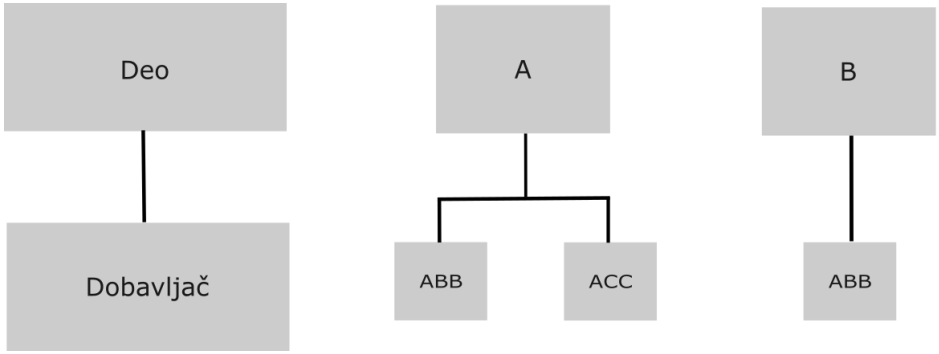
Slika 1.4.2.1. Prikaz hijerahijskog modela Dobavljač - Deo

Sa slike 1.1 se vidi da je pristup podacima zasnovan na pravilu odnosa roditelj – dete. U ovom primeru *Dobavljač* predstavlja roditeljski zapis, a *Deo* dete zapis. Za pristup

zapisu na najnižem nivou neophodno je započeti pretragu od početnog korenog čvora, odnosno korisnik mora da zna organizaciju strukture stabla da bi pronašao određeni podatak.

Za hijerahijski model na slici 1.4.2.2 lako se može dobiti odgovor na pitanje koje delove nabavlja određeni dobavljač.

Međutim, ako je potrebno odgovoriti na pitanje koji dobavljači nabavljaju određeni deo, neophodna je drugačija struktura modela. Na slici 1.2 je dat šematski prikaz hijerahijskog modela sa izmenjenom strukturom.



Slika 1.4.2.2. Prikaz hijerahijskog modela Deo - Dobavljač

Ovaj primer ukazuje na ograničenje hijerahijskog modela da omogući postojanje jedinstvene strukture modela koja obezbeđuje realizaciju i podršku za oba postavljena pitanja.

Dodavanje slogova u tabelu dete nije moguće izvršiti dok se prethodno ne izvrši dodavanje odgovarajućeg sloga u tabelu roditelja. Jedan od osnovnih nedostataka hijerarhijskog modela je pojava redudanse koja dovodi do anomalija u podacima prilikom ažuriranja.

Za prikazan primer hijerarhijskog modela može se pretpostaviti da jedan deo može da dobavlja više dobavljača, a da jedan dobavljač dobavlja više delova. Prikazivanje slučaja veze više – na – više podrazumeva pojavljivanje istog dela i dobavljača na više različitih mesta što dovodi do pojave redudanse i mogućnosti pojave greški prilikom ažuriranja podataka.

Generalno, nedostaci hijerahijskog modela podataka mogu se svrstati u tri kategorije i to:

- Anomalija dodavanja novih zapisa: Ne može se dodati podatak o nekom delu ako ne postoji dobavljač koji ga dobavlja.
- Anomalija brisanja zapisa: Ako se obriše podatak o dobavljaču koji je jedini nabavljao određeni deo, gube se i podaci o tom delu.

– Anomalija izmene zapisa: Promena podataka vezanih za neki deo podrazumeva ažuriranje istih podataka na više mesta, onoliko koliko ima različitih dobavljača tog dela.

Međutim i pored navedenih nedostaka, zbog svoje hijerhijske organizacije, danas postoje određeni komercijalni sistemi zasnovani na ovom modelu.

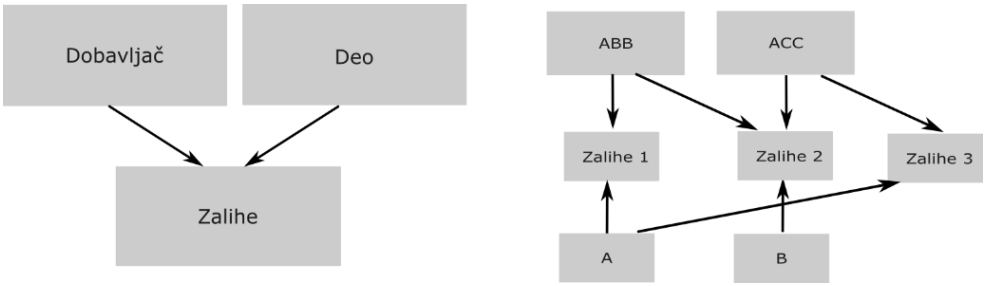
1.4.3 MREŽNI MODEL

Mrežni model podataka je još jedan od ranih modela organizacije baze podataka. Razvio se iz hijerarhijskog modela kako bi prevazišao neke od njegovih ograničenja i nedostataka. Standard mrežnog modela podataka je definisala CODASYL (engl. *Conference on Data Systems Languages*) organizacija u *CODASYL DBTG Report*-u.

Podaci su organizovani kao kolekcija zapisa koji mogu imati više roditelja, što omogućava veću fleksibilnost u definisanju veza između podataka. Mrežni model se može predstaviti orjentisanim grafovima opšteg tipa. Model koristi princip pokazivača ne samo na podatke nego i na druge tabele.

Tabele dete mogu da imaju više od jednog roditelja, čime se stvara struktura koja podseća na mrežu. Roditeljske tabele za svaku dete tabelu omogućavaju i odnose više - prema -više, pored odnosa jedan-prema-više.

U primeru mrežnog modela baze podataka prikazanom na slici 1.4.3.1, postoji odnos više – prema - više između dobavljača i delova.



Slika 1.4.3.1 Prikaz mrežnog modela Dobavljač – Zalihe - Deo

Jedan deo može biti nabavljen od više dobavljača, jedan dobavljač može nabavljati više delova. Najvažnije što treba primetiti sa Slike 1.3 je nova tabela *Zalihe* koja omogućava dodeljivanje delova dobavljačima i predstavlja direktnu posledicu mogućnosti višestrukog roditeljstva kod mrežnog modela podataka.

1.4.4 RELACIONI MODEL

Relacioni model podataka je dominantan model za organizaciju podataka u bazi podataka. Razvijen je 1970. godine od strane američkog naučnika Edgar F. Codda, a motiv za definisanje bili su nedostaci uočeni kod hijerarhijskog i mrežnog modela.

Predstavlja teorijsku osnovu za baze podataka relacionog tipa i značajan korak u organizaciji i manipulaciji podacima u računarskim sistemima.

S obzirom da je zasnovan na teoriji skupova i relacionoj algebri, relacija se smatra osnovnim konceptom, ali zbog strukturalne jednostavnosti može se reći da na realni svet gleda putem tabela.

Relacija je skup n-torki (kraće: torki) pri čemu svaka komponenta torke predstavlja vrednost iz jednog skupa koji predstavlja domena obeležja.

Relacija predstavlja specifičan tip tabele koja poseduje određene karakteristike.

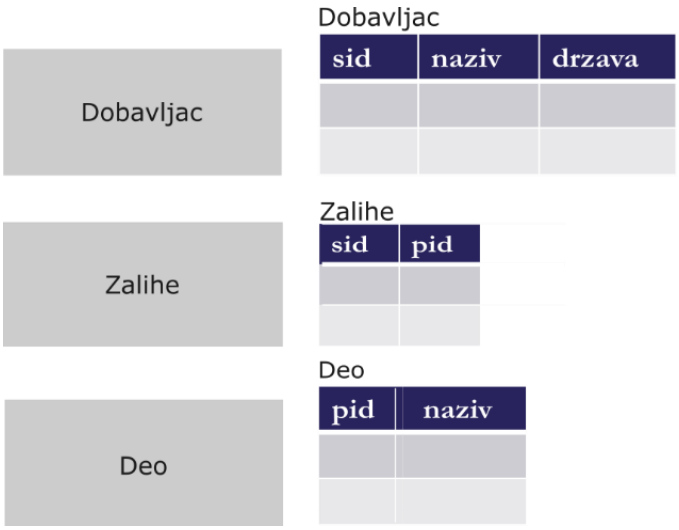
Osnovna ideja relacionog modela je organizacija podataka u tabelama (relacijama). Svaka tabela (relacija) mora imati jedinstveni naziv. Tabela može imati više kolona, pri čemu svako ime kolone treba da bude jedinstveno. Redovi tabele predstavljaju zapise podataka. Zapis, odnosno red u tabeli, ima svoje atribute. Atributi u tabeli predstavljaju se kolonama, naziv kolone je ujedno naziv atributa. Svaki zapis podataka u tabeli (relaciji) se sastoji od vrednosti pojedničnih atributa koje su smeštene u poljima (ćelijama) kolona.

Tabeli se može direktno pristupiti bez potrebe za pristupom roditeljskim objektima. Cilj je znati šta se traži, na primer ako je potrebno prikazati adresu određenog zaposlenog, neophodno je postaviti uslov koji sadrži ime i prezime zaposlenog ili jednostavno prikazati podatke o svim zaposlenim. U slučaju relacionog modela, nije potrebno pretražiti čitavu hijerarhiju, kako bi se pronašli podaci za jednog zaposlenog. Još jedna prednost relacionog modela podataka je ta da se tabele mogu povezati bez obzira na njihovu hijerarhijsku poziciju. Neophodno je da postoji logična veza između dve tabele, ali ne treba da postoji ograničenje hijerarhijskom strukturom.

Sve relacije se mogu smatrati tabelama, ali da bi tabela mogla da se definiše kao relacija mora da ispuni sledeće uslove:

- Identifikuje se jedinstvenim imenom.
- Redovi tabele predstavljaju zapise podataka.
- Kolone tabele predstavljaju atribute.
- Nazivi kolona (atributa) moraju da budu jedinstveni.
- Ćelija (polje) u koloni sadrži jednu vrednost koja predstavlja vrednost atributa.
- Sve vrednosti u jednoj koloni su istog tipa podataka.
- Ne mogu postojati dva identična reda.
- Redosled kolona nije od važnosti.
- Redosled redova nije od važnosti.

Na slici 1.4.4.1 je dat šematski prikaz relacionog modela za primer kompanije za izradu delova za dečije igračke.



Slika 1.4.4.1Prikaz relacionog modela Dobavljac – Zalihe - Deo

Model se sastoji od tabela *"Dobavljac"*, *"Zalihe"*, *"Deo"*. Kolona *"sid"* u tabeli *"Dobavljac"* jedinstveno identifikuje svakog dobavljača. Kolona *"pid"* u tabeli *"Deo"* jedinstveno identifikuje svaki deo. Odnos između tabela *"Dobavljac"* - *"Zalihe"* i *"Deo"* - *"Zalihe"* je odnos jedan-prema-više koristeći kolone *"sid"* i *"pid"*.

Odgovor na pitanje koji dobavljači su zaduženi za nabavku određenog dela može se dobiti na osnovu postavljenog upita prikazanog u prilogu 1.4.4.1

```
SELECT D.naziv, D.drzava
FROM Dobavljac D, Zalihe Z, Delovi De
WHERE D.sid = Z.sid  AND Z.pid = De.pid AND  De.naziv = "A";
```

Prilog 1.4.4.1 Upit za prikaz podataka o dobavljačima koji su zaduženi za nabavku dela „A“

Za prikaz izvešataja o delovima koje nabavlja određeni dobavljač nije potrebno izvršiti izmene u modelu i strukturi podataka. Traženi izveštaj se generiše na osnovu upita prikazanog u prilogu 1.5.

```
SELECT De.naziv
FROM Dobavljac D, Zalihe Z, Delovi De
WHERE D.sid = Z.sid  AND Z.pid = De.pid
AND  D.naziv = "ABB";
```

Prilog 1.4.4.2 Upit za prikaz podataka o delovima koje nabavlja dobavljač „ABB“

Relacioni model je rešio probleme i nedostatke hijerahijskog i mrežnog modela. Za rad sa podacima je definisan deklarativan upitni jezik. Obezbeđena je nezavisnost

podataka koji se čuvaju u bazi - aplikacija ne treba da zna kako su podaci skladišteni. Upit pretpostavlja da je relacija (tabela) uređen skup primera, bez znanja o načinu čuvanja i prikazivanja podataka. Upit se ne menja ako se promeni fizička reprezentacija podataka. Sistem za upravljanje relacionim modelom obezbeđuje efikasan pristup podacima i automatsku optimizaciju upita na osnovu generisanog plana izvršavanja (engl. *Query Excetuion Plan*).

1.4.5 OBJEKTNI MODEL PODATAKA

Objektni model podataka je način organizovanja podataka koji omogućava predstavljanje entiteta i njihovih međusobnih odnosa na način koji prikazuje njihovu prirodnu strukturu. Podaci se organizuju u klase koje definišu strukturu i ponašanje objekata. Svaki objekat pripada određenoj klasi. Za razliku od relacionog modela podataka koji se koristi za izdvajanje zapisa u dve dimenzije, objektni model je efikasan za pronalaženje jedinstvenih elemenata. Shodno tome, objektni model podataka se slabije ponaša kada je u pitanju izdvajanje više od jednog elementa, gde je relacioni model podataka bolji.

Model objektnih baza podataka rešava neke od kompleksnosti specifične za relacioni model podataka, kao što su ograničenja kod upotrebe tipova podataka i realizacije veze više-na-više. Još jedna prednost objektnog modela podataka je sposobnost upravljanja izuzetno kompleksnim aplikacijama i bazama podataka. Osnovni principa objektnih metoda, prema kojem se izuzetno složeni elementi mogu razložiti na svoje najosnovnije delove, omogućava eksplicitan pristup, kao i izvršenje unutar tih osnovnih delova.

Jedan od najvećih problema između aplikacija koje su programirane koristeći objektnu metodologiju i relacionih baza podataka predstavlja proces mapiranja između ova dva strukturna tipa: objektnog i relacionog. Objektna i relaciona struktura su potpuno različite. Stoga je bitno razumeti tehnike modelovanja objektnih baza podataka kako bi se omogućio efikasan razvoj upotrebe relacionih baza podataka od strane aplikacija zasnovanih na objektnoj metodologiji.

1.4.6 OBJEKTNO-RELACIONI MODEL PODATAKA

Objektno-relacioni model podataka je nastao kao odgovor na probleme između relacionih i objektnih modela podataka. U suštini, mogućnosti modelovanja objektnih baza podataka su uključene u relacione baze podataka, ali ne i obrnuto. Mnoge relacione baze podataka omogućavaju skladištenje binarnih objekata i ograničene mogućnosti kodiranja objektnih metoda, sa različitim stepenima uspeha.

Najveći problem sa skladištenjem binarnih objekata u relacionoj bazi podataka je što se potencijalno veliki objekti čuvaju u elementu malog obima odnosno kao u jednom polju reda zapisa u tabeli. Ovo nije uvek slučaj jer neke relacione baze podataka

omogućavaju skladištenje binarnih objekata u posebnom datotekama koje su izvan dvodimenzionalnih struktura zapisa tabele.

Evolucija modelovanja baza podataka je počela sa suštinski nepostojećim modelom baze podataka u slučaju baza podataka zasnovanih na sistemima datoteka, razvijajući se kroz hijerarhije radi strukturiranja, mreža radi omogućavanja specijalnih veza, prelazeći na relacioni model podataka koji omogućava pristup jedinstvenim pojedinačnim elementima bilo gde u bazi podataka.

Objektni model podataka ima specifičnu funkciju u upravljanju visokom brzinom primene malih podataka unutar velikih i visoko kompleksnih skupova podataka. Objektno-relacioni model pokušava uključiti odgovarajuće aspekte objektnog modela u strukturu realacionog modela podataka.

1.5 PITANJA I ZADACI

1. Objasniti pojam baze podataka.
2. Šta je podatak?
3. Šta je informacija?
4. Objasniti razliku između podatka i informacije.
5. Koji su ključni aspekati upravljanja podacima u bazama podataka?
6. Objasniti pojam sistema za upravljanje bazama podataka.
7. Koje su osnovne funkcije sistema za upravljanje bazama podataka?
8. Objasniti pojam ANSI –SPARC troslojne arhitekture apstrakcije podataka.
9. Koje su prednosti i nedostaci sistema za upravljanje bazama podataka?
10. Navesti razlike između hijerahijskog, mrežnog, relacionog, objektnog i objektno-relacionog modela.

2 RELACIONI MODEL PODATAKA

Cilj ovog poglavlja je usmeren na osnovne koncepte relacionog modela podataka. Objašnjeni su pojmovi relacija, šema relacije, šema relacione baze podataka, atribut, domen, *n*-torke, instance i *NULL* vrednosti. Izdvojene su ključne karakteristike relacija koje ih izdvajaju od običnih datoteka ili tabele podataka. Analizirani su i opisani različite vrste ključeva. Definisani su i objašnjena ograničenja koja proizilaze iz relacionog modela podataka, uključujući integritet ključeva, integritet entiteta, referencijalni integritet i semantički integritet.

2.1 UVOD

Definicija relacionog modela, postavljena od strane E.F.Codd-a, odnosi se na model podataka koji koristi relacije kao osnovnu strukturu podataka, uz formalni jezik zasnovan na teorijskim konceptima relacione algebre. Tokom vremena, razvoj i evolucija principa modelovanja podataka doveli su do toga da danas ne postoji jedinstvena standardizovana specifikacija relacionog modela. Umesto toga, postoje različite klase modela koje se oslanjaju na koncept relacija.

Relacioni model podataka omogućio je fleksibilniji i jednostavniji način manipulacije podacima. Upotreba različitih operatora za definisanje upita, ažuriranje podataka i specifikaciju ograničenja omogućava jasnu i konzistentnu organizaciju podataka, značajno olakšavajući proces razvoja baza podataka.

Ovaj model poslužio je kao osnova za istraživanje odnosa i ograničenja u podacima, razvoj različitih metodologija dizajna baza podataka, kao i za standardizovanje jezika za upravljanje i manipulaciju podacima, poznat kao struktuirani upitni jezik (SQL), koji se koristi u većini sistema za upravljanje bazama podataka.

Relacioni model razmatra strukturalne, integritetske i manipulativne komponente podataka. Obuhvata strukturu podataka (objekti, entiteti), zaštitu podataka (ograničenja) i manipulaciju podacima (ažuriranje).

Teorijska osnova relacionog modela omogućava sistematsko projektovanje i razvoj relacionih baza podataka kroz lako razumljive apstrakcije. Na taj način, kompleksnost procesa projektovanja se smanjuje, omogućavajući dizajnerima da se fokusiraju na suštinski problem.

2.2 OSNOVNI KONCEPTI RELACIONOG MODELA PODATAKA

Relacioni model podataka opisuje svet kao "skup međusobno povezanih relacija odnosno tabela". U terminologiji relacionog modela, tabela se naziva relacija, redovi relacije se nazivaju torke ili *n*-torke, a kolone atributi. Vrednosti koje mogu da se pojave u nekoj od kolona, odnosno vrednosti atributa, su određene domenom.

Relacija se može posmatrati kao dvodimenzionalna tabela koja poseduje specifične karakteristike. Sastoji se od redova i kolona. **Red**, ili **zapis**, predstavlja skup povezanih podataka koji se nalaze u različitim kolonama.

Relacije *STUDENT* je prikazana na slici 2.2.1. Svaki red predstavlja jednog studenta, dok svaka kolona predstavlja jednu osobinu ili karakteristiku studenta.

indeks	**ime**	**datum_r**
RT – 1/24	Lazar	11.09.2005.
NRT-2/21	Milica	10.01.2000.
IS – 12/23	Ana	01.01.2004.

Slika 2.2.1. Relacija STUDENT

U relacionom modelu podataka, matematička definicija relacije koristi se skupovima, funkcijama i predikatima.

Formalna definicije relacije

Neka je $A_1,A_2,...,A_n$ skup atributa (karakteristika) koji pripadaju relaciji R. Svaki atribut ima svoj domen, tj. skup mogućih vrednosti koje može uzeti. Domen atributa A_i označavamo sa D_i.

Relacija R u relacionom modelu podataka je podskup Kartezijevog proizvoda domena svojih atributa i definiše se kao:

$$R \subseteq D_1 \times D_2 \times ... \times D_n$$

Svaki element relacije R je uređena n-torka. $D_1 \times D_2 \times ... \times D_n$ predstavlja Kartezijev proizvod domena atributa, a R je podskup tog proizvoda. To znači da R sadrži samo one n-torke (v_1, v_2, ..., v_n) tako da v_1 pripada domenu D_1, v_2 pripada domenu D_2, ... v_n pripada domenu D_n.

Struktura relacije u relacionom modelu podataka precizno se opisuju sledećim karakteristikama:

– Zaglavlje relacije R predstavlja skup atributa $\{A_1, A_2,..., A_n\}$ koji je definišu. Svaki atribut ima svoje jedinstveno ime i pripada određenom domenu Di. Različiti atributi mogu imati isti domen vrednosti.

- Telo relacije *R* čine *n*-torke (v_1, v_2,..., v_n) koje označavaju stvarne podatke u relaciji. Svaka *n*-torka ima isti broj elemenata kao što ima atributa u zaglavlju, a svaki elemenat (član, komponenta) *n*-torke ima vrednost iz odgovarajućeg domena atributa *Ai* (v_i pripada D_i).
- Domen atributa D_i se definiše kao skup mogućih vrednosti koje određeni atribut *Ai* može imati.
- Stepen relacije *R* je jednak broju atributa u zaglavlju relacije.
- Kardinalnost relacije *R* je jednaka broju *n*-torki ili redova koji se nalaze u relaciji.

Kolone relacije su imenovane **atributima**. Svaka relacija može imati više kolona, pri čemu svako ime kolone - atribut treba da bude jedinstven. Atributi se pojavljuju na vrhu kolona i opisuju značenje podataka koji se unose u odgovarajuću kolonu. U tabeli 2.2.1 atributi su *indeks, ime, datum_r*. Kolona sa atributom *datum_r* sadrži datum rođenja studenta izražen u formatu dan.mesec.godina. Može se reći da atribut definiše zapis, a zapis sadrži skup atributa.

Šema relacije predstavlja opis relacije. Prikazuje se tako što se navede naziv relacije, skup atributa i evenatualno skup ograničenja. Uz svaki atribut se može navesti domen i ograničenje ključa.

Atributi u šemi relacije predstavljaju skup, a ne listu. S obzirom da su u pitanju relacije, neophodno je odrediti standardni redosled atributa. Kada se uvede šema relacije sa listom atributa, kao u prethodnom primeru, taj redosled se smatra standardnim kad god se prikazuje relacija ili neki od njenih redova. Može se smatrati da šema relacije predstavlja zaglavlje tabele koja predstavlja relaciju.

Formalna matematička definicija šeme relacije u relacionom modelu uključuje skup atributa {A_1, A_2, ..., A_n}, gde svaki atribut A_i ima svoje ime i pripada određenom domenu D_i.

Šema relacije *R* sa skupom atributa A_1, A_2, ..., A_n može se predstaviti kao:

- $R(A_1, A_2, ..., A_n)$
- $R(\underline{A_1}, A_2, ..., A_n)$
- $R(A_1{:}D_1, A_2{:}D_2, ..., A_n{:}D_n)$ – uključeno navođenje domena za atribute
- $R(\underline{A_1}{:}D_1, A_2{:}D_2, ..., A_n{:}D_n)$ – uključeno navođenje domena za atribute

A1 se odnosi na ograničenje tipa "primarni ključ relacije".

Primer: U nastavku su navedene različite varijante šeme relacije *STUDENT* koja je prikazana tabelom 2.2.1 :

- STUDENT (indeks, ime, datum_r)
- STUDENT (indeks:string, ime:string, datum_r:date)
- STUDENT (indeks, ime, datum_r)
- STUDENT (indeks:string, ime:string, datum_r:date)

U relacionom modelu, baza podataka se sastoji od jedne ili više relacija. Skup šema relacija u bazi podataka se naziva **relacijskom šemom baze podataka**, ili jednostavno, **šemom baze podataka**. Šema baze podataka se takođe naziva vizualnom ili logičkom arhitekturom jer ukazuje na način kako su podaci organizovani. Definiše način na koji su podaci organizovani i međusobno povezani. Uključuje informacije o tabelama, kolonama, tipovima podataka koje svaka kolona može sadržati, ograničenjima koja se primenjuju na podatke, kao i o odnosima između tabela. Šema baze podataka služi kao osnova za skladištenje i upravljanje podacima u bazi podataka.

Torke (*n*-torke) (engl. *tuples*) su redovi relacije, osim reda zaglavlja koji sadrži imena atributa. *N*-torka ima jedan elemenat (član, komponentu) za svaki atribut relacije. Na primer, prva od tri *n*-torke prikazana u tabeli 2.2.1 ima tri elementa: "RT-1/24", "Lazar", "11.09.2005." za atribute *indeks, ime, datum_r*.

Izdvojeno od relacije, prethodno navedena *n*-torka se prikazuje na sledeći način:

<RT-1/24", "Lazar", "11.09.2005.">

Kada se *n*-torka pojavi izolovano od relacije, atributi se ne navode. Zbog toga je neophodno obezbediti poštovanje definisanog rasporeda atributa u šemi relacije kako bi se znalo kojoj relaciji *n*-torka pripada. Uzimajući u obzir da su relacije skupovi *n*-torki, a ne liste, redosled navođenja *n*-torki u relaciji nije od značaja. Takođe, promena redosleda atributa u relaciji neće promeniti samu relaciju. Međutim, promena redosleda atributa u šemi relacije, podrazumeva promenu redosleda kolona. Kada se menja redosled atributa, menja se i redosled elemenata *n*-torki. Neophodno je da svaka *n*-torka ima elemente permutovane na isti način kao što su i atributi permutovani.

Na slici 2.2.2 prikazana je jedna od varijanti relacija nastala permutovanjem redova i atributa relacije *STUDENT* prikazane na slici 2.2.1. Ove dve relacije se smatraju "istom" pa se može reći i da prikazane tabele predstavljaju različite prikaze iste relacije.

datum_r	**ime**	**indeks**
01.01.2004.	Ana	IS – 12/23
10.01.2000.	Milica	NRT-2/21
11.09.2005.	Lazar	RT – 1/24

Slika 2.2.2. Jedna od varijanti relacije STUDENT

Domen predstavlja specifikaciju skupa mogućih vrednosti atributa. Koncept domena u relacionom modelu omogućava definiciju značenja i izbora vrednosti koje jedan atribut može imati. Definiše se tipom podataka, dužinom podataka i opsegom vrednosti. Atributi uzimaju vrednosti iz odgovarajućeg domena koji im je dodeljen, što znači da će vrednosti u tabeli za neku kolonu da budu onog tipa podataka koji je izabran za tu kolonu.

Relacioni model zahteva da svaka komponenta n-torke bude **atomarna**; to jest, mora biti nekog od osnovnih tipova podataka kao što su celobrojni, znakovni, datumski. Nije dozvoljeno da vrednost bude struktura zapisa, skup, lista, niz ili bilo koji drugi tip koji bi mogao imati svoje vrednosti razdvojene na manje komponente. Za svaki atributu relacije mora biti pridružen domen, odnosno određen osnovni tip podatka. Pojedinačne komponente bilo koje *n*-torke u relaciji moraju imati vrednost koja pripada domenu odgovarajuće kolone. Na primer, *n*-torke u relaciji *STUDENT* moraju za prvi i drugi elemenat imati definisan tip podataka *string*, a za treći tip podataka *datum*.

Relacija se menjaju tokom vremena. Na primer, u relaciji *STUDENT* mogu se dodati *n*-torke za nove studente. Moguće je izvršite određene izmene u postojećim *n*-torkama (na primer, ažuriranje broja indeksa jer je student promenio smer). Ukoliko se neki student ispisao, briše se *n*-torka sa komponentama koje sadrže podatke o istom. Manje je uobičajeno da se šema relacije menja. Postoje situacije u kojima je potrebno dodati nove ili obrisati postojeće atribute. Promene u šemi značajno utiču na performanse baze podataka. Pre dodavanja ili brisanja atributa neophodno je sačuvati trenutne podatke što podrazumeva kopiranje možda i više hiljada *n*-torki u posebnu relaciju. Dodavanje novih atributa može dovesti do pojave praznih odgovarajućih komponenti u postojećim *n*-torkama što se može smatrati posledicom nemogućnosti generisanja odgovarajućih vrednosti.

NULL vrednost u kontekstu baza podataka predstavlja specifičnu vrednost koja se koristi kako bi označila da određeni podatak nije poznat ili da ne postoji. U mnogim slučajevima, *null* se koristi kada vrednost određenog atributa nije dostupna ili nije primenjiva za određeni red u relaciji. Na primer, ako uzmemo u obzir atribut "e-mail adresa" u relaciji *STUDENT, null* vrednost bi se koristila ako student nije dao svoju e-mail adresu ili ako taj podatak nije dostupan. *Null* vrednost često ukazuje na nedostatak informacija i može se interpretirati kao "nepoznato" ili "neprimenjivo". Međutim, važno je napomenuti da null vrednost nije isto što i prazna ili nula vrednost. *Null* predstavlja odsustvo vrednosti ili nepoznanicu, dok prazna vrednost može imati svoj specifičan značaj u zavisnosti od tipa podataka. Iako *null* vrednosti omogućavaju fleksibilnost u radu sa podacima, one takođe mogu predstavljati izazov u upravljanju bazama podataka. *Null* vrednosti mogu uzrokovati poteškoće prilikom izvođenja upita, spajanja tabela ili ažuriranja podataka. Stoga, u praksi se teži minimiziranju korišćenja *null* vrednosti gde god je to moguće, a često se koriste alternative poput podrazumevanih vrednosti ili posebnih oznaka koje ukazuju na nedostatak podataka. Razumevanje kako null vrednosti funkcionišu i kako se nose sa njima od suštinskog je značaja za održavanje kvaliteta podataka u bazi podataka.

Instanca predstavlja skup *n*-torki relacije u nekom trenutku. Može se definisati i kao pojava relacije u jednom trenutku. U bazi podataka se čuva jedna verzija bilo koje relacije odnosno trenutni skup *n*-torki koji se nalaze u istoj. Relacija može da sadrži različite n-torke u različitim trenucima.

Primer: Za šemu relaciju *STUDENT* (indeks:*string*, ime:*string*, datum_r:*date*) relacija *STUDENT* u jednom trenutku može da sadrži sledeće instance:

<"RT-1/24", "Lazar", "11.09.2005.">

<"NRT-2/21", "Milica", "10.01.2000.">

<"IS-12/23", "Ana", "01.01.2004.">

<"NET-5/22", "Marko", "19.11.2001.">

<"RT-2/22", "Aca", "22.12.2003.">

indeks	ime	datum_r
RT – 1/24	Lazar	11.09.2005.
NRT-2/21	Milica	10.01.2000.
IS – 12/23	Ana	01.01.2004.
NET-5/22	Marko	19.11.2001.
RT-2/22	Aca	22.12.2003.

2.3 ZNAČAJNE OSOBINE RELACIJE I ŠEME RELACIJE

U relacionom modelu podataka, mogu se izdvojiti značajne osobine relacije i šeme relacije koje je čine različitom od datoteke podataka ili obične tabele.

- Svaki atribut u šemi relaciji ima jedinstveno ime. Šeme relacije ne može da sadrži dva atributa sa istim nazivom.
- Redosled atributa u šemi relaciji nije od važnosti.
- Svaka *n*-torka u relaciji mora biti jedinstveno identifikovana putem sadržaja svojih elemenata. Relacija ne može da sadrži dve identične *n*- torke.
- Redosled atributa u relaciji nije od važnosti.
- Redosled *n*-torki u relaciji nije od važnosti.
- Sve vrednosti podataka u atributu moraju biti iz istog domena (isti tip podataka).
- Svaka vrednost podataka povezana sa atributom mora biti atomarna (ne može se dalje podeliti na značajne delove).
- Nijedan atribut ne može imati više vrednosti podataka u jednoj *n*-torki.

2.3.1 KLJUČEVI

Relacioni model podrazumeva implementaciju različitih ograničenja koja se postavljaju prilikom definisanja šeme baza podataka. Fundamentalan tip ograničenja predstavlja ograničenje ključa. Ograničenje ključa ima značajnu ulogu u održavanju integriteta podataka, sprečavanja pojave nekonzistentnosti i obezbeđenja tačnosti i jedinstvenosti podataka u bazi podataka.

Šema relacije se karakteriše ključem, koji može biti pojedinačni atribut ili kombinacija atributa koja definiše kompozitni ključ. Vrednosti ključa jedinstveno identifikuju svaku instance relacije. Atribut ili skup atributa koji čini ključ obezbeđuje da dve n-torke relacije imaju različite vrednosti u svim atributima ključa. Sve relacije imaju najmanje jedan ključ. Ograničenje ključa se specificira prilikom definisanja baze podataka kroz različite vrste ključeva. U terminologiji relacionog modela podataka koriste se određeni pojmovi koji opisuju relacione ključeve.

Iz same definicije ključa proizilazi jedna od značajnih karakteristika koja se odnosi na svojstvo da ključ treba da predstavlja minimalni skup atributa kojim se jedinstveno identifikuje svaka *n*-torka relacije. Uklanjanje ili izmena jedne od komponenti ključa dovodi do gubljenja svojstva identifikacije.

Kandidat ključ

Relacija može imati jedan ili više atributa koji uzimaju jedinstvene vrednosti. Bilo koji od ovih atributa može se koristiti kako bi se jedinstveno identifikovale *n*-torke u relaciji. Takvi atributi se nazivaju **kandidatski ključevi**. Kandidat ključ predstavlja skup jednog ili više atributa čije kombinacije obezbeđuju jedinstveno identifikovanje svake n-torke u relaciji.

Primer: U šemi relacije *STUDENT* (indeks:string, ime:string, datum_r:date) atribut *indeks*, u svakoj *n*-torci, ima jedinstvenu vrednost. Dva studenta ne mogu imati isti broj indeksa. Zbog toga atribut *indeks* predstavlja kandidat ključ, jer može biti kandidati za primarni ključ. Uzimajući u obzir da dva studenta mogu imati isto ime i mogu biti rođeni istog dana, atributi *ime* i *datum_r* ne mogu se smatrati kandidatskim ključevima.

Primarni ključ

Primarni ključ se bira između jednog ili više kandidatskih ključeva tako da jedinstveno identifikuje svaku *n*-torku relacije. Preostali atributi u listi kandidatskih ključeva nazivaju se alternativni ključevi. Primarni ključ je poseban tip ključa koji jedinstveno identifikuje svaku instancu određene relacije.

Ključne karakteristike primarnog ključa su:

- Sve vrednosti primarnog ključa moraju biti jedinstvene za svaku *n*-torku u relaciji. Posledica prethodno navedenog je precizno i nedvosmisleno identifikovanje svake instance.
- Vrednosti koje čine primarni ključ ne smeju biti prazne vrednosti što osigurava da instanca relacije ima važeći primarni ključ.
- Vrednosti primarnog ključa ne mogu se menjati nakon što su dodeljene. Na ovaj način se obezbeđuje stabilnost identifikacije i integritet podataka.
- Primarni ključ obično čine jedan ili više atributa koji su direktno povezani sa suštinom podataka u relaciji. Često se koristi jedan atribut koje je jedinstven za svaku instancu relacije.
- U većini slučajeva, primarni ključ se definiše postavljanjem mehanizma koji obezbeđuje automatsko generisanje pripadajućih numeričkih vrednosti.
- Primarni ključ često igra ključnu ulogu u povezivanju podataka između različitih relacija.
- Pravilno definisan i implementiran primarni ključ ključan je za održavanje integriteta podataka i olakšava efikasne upite i modifikacije podataka unutar baze.

Primer: Šema relacije *STUDENT* (indeks:string, ime:string, datum_r:date) može se modifikovati dodavanjem atributa *id_student:integer* koji će da ima jedinstvenu numeričku vrednost koja se automatski generiše dodavanjem nove instance. Postavljanjem atributa *id_student* za primarni ključ, atribut *indeks* se može smatrati alternativnim ključem.

Nakon modifikcije, šema relacije *STUDENT* se može prikazati na sledeći način:

STUDENT (id_student:integer, indeks:string, ime:string, datum_r:date)

Kompozitni primarni ključ

Ako nijedan pojedinačni atribut u relaciji ne obezbeđuje jedinstveno identifikovanje n-torki, više od jednog atributa se definiše zajedno kao primarni ključ. Takav primarni ključ koji se sastoji od više od jednog atributa naziva se **kompozitni primarni ključ**.

Strani ključ

Strani ključ se koristi kako bi se predstavila veza između dve relacije. Strani ključ je atribut čija vrednost proizlazi iz primarnog ključa druge relacije. To znači da bilo koji atribut relacije (koji referencira), a koji se koristi za referenciranje sadržaja iz druge (referencirane) relacije, postaje strani ključ ako se odnosi na primarni ključ referencirane relacije. Relacija koja ima ovakvu referencu naziva se *strana relacija*. U nekim slučajevima, strani ključ može uzeti vrednost *NULL* (vrednost nije dodeljena) ako nije deo primarnog ključa u stranoj relaciji. Relacija u kojoj je definisan primarni ključ koji se referiše naziva se *primarna relacija* ili *glavna relacija*.

2.3.2 ŠEMA RELACIONE BAZE PODATAKA

Šema relacione baze podataka je precizno definisan skup šema relacija R_i *(i=1, 2,..., m)* i postavljenih ograničenja *IC* koja regulišu interakciju između tih relacija. Ova šema, označena kao $S(R_1,R_2,...,R_m;IC)$, obuhvata sve detalje o organizaciji podataka u bazi, uključujući tabele (relacije u tabelarnoj formi) i njihove međusobne odnose.

Instanca relacione baze podataka definisane šemom $S(R_1,R_2,...,R_m;IC)$ predstavlja konkretnu realizaciju šeme i sadrži stvarne podatke koji zadovoljavaju strukturu i ograničenja definisana šemom. Označava se sa *s*.

Svaka instanca (*s*) sastoji se od n-torki koje čine pojedinačne redove u svakoj od tabela unutar šeme. Osigurava da skup relacija u instanci zadovoljava sva međurelaciona ograničenja i integritet podataka, čime se obezbeđuje doslednost i validnost informacija unutar baze podataka.

Važno je napomenuti da šema predstavlja planiranu strukturu, dok instanca ukazuje na stvarne podatke koji se čuvaju u bazi podataka u skladu sa zadatim pravilima i ograničenjima.

Primer: U bazi podataka fakultet čuvaju se podaci o profesorima, studentima, predmetima, prisutnosti studenata na predavanjima i rezultatima polaganja testova. Za profesore se čuvaju podaci o imenu, zvanju i kabinetima. Podaci koji se čuvaju o studentima odnose se na ime, broj indeksa, studijski program, godinu upisa. Za svaki predmet se čuvaju podaci o nazivu predmeta, broju ESPB bodova, predmetnom profesoru. Predmet može biti preduslov nekom drugom predmetu. Beleži se prisutnost studenta na predavanjima za termin predavanja. Rešavanjem testova studenti ostvaruju određeni broj poena.

Potrebno je:

- Dizajnirati šemu relacione baze podataka.
- Dizajnirati šeme relacija.
- Prikazati instance baze podataka FAKULTET

Dizajn šeme relacione baze podataka obuhvata: naziv šeme baze podataka (FAKULTET), imena šema relacija (PROFESOR, STUDENT, PREDMET, PREDUSLOV, PRISUTNOST, REZULTATI_TESTOVA) i imena atributa u svakoj relaciji. Svaka šema relacije sadrži specifikaciju primarnog ključa (podvučen atribut u šemi relacije) i specifikaciju stranih ključeva (italic u šemi relacije).

FAKULTET({PROFESOR, STUDENT, PREDMET, PREDUSLOV, PRISUTNOST, REZULTATI_TESTOVA }; IC)

PROFESOR (ProfesorID, ImeP, Zvanje, Kabinet)

STUDENT (StudentID, ImeS, BrInd, SP, GodUpisa)

PREDMET (PredmetID, Naziv, ESPB, *ProfesorID*)

PREDUSLOV (*PredmetID1*, *PredmetID2*)

PRISUTNOST (*StudentID*, *PredmetID*, *Datum*)

REZULTAT(*StudentID*, *PredmetID*, *DatumT*, Poeni)

Jedna pojava baze podataka FAKULTET

PROFESOR

ProfesorID	ImeP	Zvanje	Kabinet
100	Milena	Prof.	223
101	Zoran	Pred.	124
102	Jasna	VPred.	231

STUDENT

StudentID	ImeS	BrInd	SP	GodUpisa
200	Lazar	13	NRT	2024
201	Jovan	19	RT	2023
202	Aleksej	26	IS	2025

PREDMET

PredmetID	Naziv	ESPB	*ProfesorID*
10	Baze podataka 1	8	102
20	Programiranje	8	100
30	Baze podataka 2	8	102
40	Matematika	6	101

PREDUSLOV

PredmetID1	*PredmetID2*
10	30
10	20
40	10

PRISUTNOST

StudentID	*PredmetID*	*Datum*
200	10	10.10.2024.
201	10	15.11.2023.
200	10	20.11.2024.

REZULTAT

StudentID	*PredmetID*	*DatumT*	Poeni
200	10	20.10.2024.	10
201	10	12.12.2023.	7
200	10	10.01.2025.	10

2.4 OGRANIČENJA U RELACIONOM MODELU

Ograničenja u relacionom modelu baza podataka predstavljaju ključne elemente za očuvanje integriteta podataka, sprečavanje anomalija i garantovanje doslednosti i tačnosti informacija. Definišu se prilikom formiranja baza podataka kako bi se održala konzistentnost sadržaja u skladu s identifikovanim odnosima između atributa stvarnog sistema. Ova ograničenja čine neizostavan deo definicije šeme baze podataka. Pravilno definisana ograničenja igraju ključnu ulogu u dizajniranju snažnih i efikasnih baza podataka.

Neka od ograničenja su već pomenuta u prethodnom delu ovog poglavlja, kao što je ograničenje domena ili domensko ograničenje, koje određuje skup dozvoljenih vrednosti atributa relacije na osnovu prethodno definisanog domena. Takođe, bitan koncept je pojam ključa šeme relacije koji garantuje da su sve *n*-torke unutar relacije jedinstvene. Obezbeđuje identifikaciju svakog reda na osnovu jedinstvene vrednosti ključa.

Ključevi igraju značajnu ulogu u očuvanju integriteta podataka i pomažu u izbegavanju dupliciranih ili konfliktnih informacija unutar baze podataka. U relacionom modelu podataka, postoji niz pravila integriteta koja ograničavaju ili zabranjuju pojavu određenih *n*-torki u relaciji. Pravila se definišu u vezi sa ključevima, entitetima, međusobnim referenciranjem i semantikom relacija.

Integritet ključa

Ovaj tip ograničenja definiše jedinstvenu vrednost ključa u šemi relacije. Na taj način se onemogućava dupliranje *n*-torki i obezbeđuje jedinstvena identifikacija svakog reda unutar relacije. Podrazumeva se da vrednosti ključeva kandidata moraju biti jedinstvene u okviru šeme relacije, definišući time ograničenje jedinstvenosti.

Razlog uvođenja ovog ograničenja proizlazi iz činjenice da je jedinstvenost vrednosti ključeva kandidata neophodna s obzirom na to da relacija predstavlja skup *n*-torki, onemogućavajući tako postojanje identičnih *n*-torki u relaciji.

Integritet entiteta

Ovo ograničenje podrazumeva da vrednost primarnog ključa, kao i bilo koja od njegovih komponenti, ne smeju imati nepoznatu (*NULL*) vrednost u okviru relacije. S obzirom na to da primarni ključ služi za jedinstveno identifikovanje pojedinačnih

n-torki unutar relacije, postojanje nepoznate vrednosti primarnog ključa onemogućilo bi identifikaciju određenih *n*- torki relacije.

Referencijalni integritet

Referencijalni integritet predstavlja ključno ograničenje u relacionom modelu. Osigurava održavanje veza između relacija, garantujući da vrednosti stranog ključa u jednoj relaciji referenciraju isključivo na postojeće vrednosti primarnog ključa u drugoj relaciji. Predstavlja pravilo ograničenja stranog ključa.

Ovo pravilo definiše da ako postoji strani ključ u nekoj relaciji, vrednost stranog ključa mora biti jednaka vrednosti ključa kandidata neke *n*-torke u referenciranoj relaciji, ili može imati *NULL* vrednost. Ograničenje se uspostavlja između dve relacije kako bi se očuvala konzistentnost među *n*-torkama povezanih relacija.

Referencijalni integritet garantuje da *n*-torka iz jedne relacije referencira samo postojeću *n*-torku u drugoj ili istoj relaciji. Veza obezbeđuje integritet podataka i sprečava situacije u kojima bi se referenca odnosila na nepostojeće podatke. Ograničenje stranog ključa je ključan mehanizam za implementaciju referencijalnog integriteta, a njegovo ispravno postavljanje igra ključnu ulogu u očuvanju integriteta baza podataka.

Semantički integritet

Ovo ograničenje definiše specifične uslove ili semantičke zahteve koji moraju biti ispunjeni kako bi se zadovoljile potrebe aplikacije ili poslovnog konteksta. Odnose se na opšta, aplikativno orijentisana ograničenja koja korisnici definišu kako bi specificirali ili ograničili određene aspekte realnog sistema.

Pravila semantičkog integriteta ukazuju na situacije koje ne mogu biti obuhvaćene ograničenjima na nivou šeme baze podataka zbog specifičnosti sistema. Na primer, ograničenja poput "Broj ESPB bodova za određeni predmet ne može biti manji od 2 i veći od 10" ili "Ocena na ispitu može se formirati samo ako je student ostvario više od 50 poena" ne mogu se precizno specificirati korišćenjem prethodno opisanih deklarativnih ograničenja na nivou šeme baze podataka. Umesto toga, ova ograničenja se mogu implementirati kroz procedure s ciljem osiguravanja da se poslovna pravila stvarnog sistema pravilno primenjuju.

Ovi aspekti ograničenja integriteta igraju ključnu ulogu u očuvanju doslednosti, tačnosti i pravilnog funkcionisanja relacionih baza podataka. Pravilno definisana ograničenja često proizlaze iz analize potreba sistema i detaljnog razumevanja podataka koji će biti smešteni u bazi, obezbeđujući stabilnu i efikasnu strukturu podataka.

Definisanje ograničenja

U strukturi relacije eksplicitno se definišu određena ograničenja kako bi se osigurala doslednost i konzistentnost podataka:

- Domeni atributa, koji precizno definišu prihvatljive vrednosti za svaki atribut.
- Primarni ključ, ključna vrednost koja jedinstveno identifikuje svaku n-torke u relaciji.
- Strani ključevi i referenca na primarni ključ, uspostavljanje veza između različitih relacija.
- Mogućnost atributa da sadrži *NULL* vrednost, što ukazuje na nepoznate ili nedostupne vrednosti.
- Obaveznost atributa da ima jedinstvenu vrednost, čime se zabranjuje dupliciranje vrednosti među n-torkama.
- Definisanje podrazumevane vrednosti za atribut, koja se primenjuje ako vrednost nije eksplicitno zadana.
- Semantička ograničenja koja se odnose na specifične karakteristike domena podataka.

Ograničenja integriteta jasno se specificiraju u šemi baze podataka kroz korišćenje jezika za definisanje podataka (*DDL*), dela *SQL*-a posvećenog definiciji strukture baze podataka. Ova ograničenja prate i održavaju sistemi za upravljanje bazama podataka (*DBMS*) kako bi garantovali doslednost podataka.

Semantička ograničenja, koja se odnose na specifične uslove ili zahteve relevantne za poslovnu logiku, takođe se mogu kontrolisati putem aplikativnih programa za ažuriranje baze podataka. Alternativno, mogu se direktno ugraditi u šemu baze podataka pomoću jezika za definisanje ograničenja. Većina komercijalnih *DBMS*-a podržava osnovna ograničenja ključa i entiteta, a manji broj pruža podršku za referencijalni i semantički integritet, što je važno imati na umu prilikom odabira odgovarajućeg sistema za upravljanje bazom podataka.

2.5 PITANJA I ZADACI

1. Objasniti koncept relacije u relacioniom modelu podataka.
2. Navesti osnovne karakteristike relacije.
3. Šta je šema relacije, a šta šema baze podataka?
4. Navesti tipove ključeva.
5. Koja je razlika između primarnog i stranog ključa?
6. Šta je osnovni cilj definisanja ograničenja u relacionom modelu baza podataka?
7. Objasniti pojam integriteta ključa.
8. Kojim integitetom se osigurava održavanje veza između relacija?
9. Koje ograničenje podrazumeva da vrednost primarnog ključa, kao i bilo koja od njegovih komponenti, ne smeju imati nepoznatu (NULL) vrednost?
10. Na šta ukazuju pravila semantičkog integriteta?

3 RELACIONA ALGEBRA

Cilj ovog poglavlja je definisanje i objašnjenje matematičkih operacija relacione algebre, koje su osnova upitnog jezika korišćenog za manipulaciju i rad sa relacionim bazama podataka. Objašnjen je razlika između osnovnih i izvedenih operacija, kao i između unarnih i binarnih operacija. Definisani su matematički koncepti operacija selekcije, projekcije, preseka, unije, razlike i Dekartovog proizvoda. Objašnjeni su koncepti prostih i složenih uslova, kao i značaj unijski kompatibilne relacije. Takođe, definisani su matematički koncepti različitih operacija spajanja relacija, kojima se kombinuju podaci iz dve relacije na osnovu zajedničkih atributa ili uslova.

3.1 OPERACIJE RELACIONE ALGEBRE

Relaciona algebra je osnova za upitne jezike koji se koriste u radu sa bazama podataka. Svaki algebarski izraz u relacionoj algebri predstavlja formalni upit proceduralnog karaktera, koji se koristi za opisivanje operacija u relacionom modelu podataka. Ova algebra se zasniva na matematičkoj teoriji skupova i obuhvata skup operatora za manipulaciju relacijama, pri čemu su rezultati operacija takođe relacije. Operacija u relacionoj algebri podrazumeva primenu određenog operatora na jednu ili više izvornih relacija kako bi se formirala nova relacija.

Na osnovu načina formiranja i korišćenja, operacije se mogu podeliti na **elementarne** i **izvedene**.

Elementarne operacije su osnovne operacije za manipulaciju podacima u relacionoj algebri. U ovu kategoriju spadaju:

- selekcija ili restrikcija
- projekcija
- unija
- razlika
- Dekartov proizvod.

Izvedene operacije se formiraju korišćenjem kombinacija elementarnih operacija i omogućavaju naprednije manipulacije podacima. U ovu kategoriju spadaju:

- spajanje
- intersekcija ili presek
- deljenje

Bitno je napomenuti da u relacionom modelu, kao i u samoj relacionoj algebri, relacije predstavljaju kako operande, tako i rezultat operacija. Drugim rečima, operandi u izrazima relacione algebre su same relacije, odnosno skupovi *n*-torki ili njihovi ekvivalenti, dok je rezultat primene izraza takođe relacija.

Operacije relacione algebre se takođe mogu klasifikovati i **prema broju operanada** koje koriste i to:

- **Unarne operacije (1 operand).** Ove operacije se izvršavaju nad jednom relacijom. Na primer, selekcija i projekcija su unarne elementarne operacije, gde selekcija bira određene *n*-torke iz jedne relacije na osnovu uslova, dok projekcija bira određene atribute iz jedne relacije.
- **Binarne operacije (2 operanda).** Ove operacije se izvršavaju nad dve relacije. Unija, razlika i Dekartov proizvod su elementarne binarne operacije. Unija kombinuje *n*-torke dve relacije, razlika pronalazi razlike između dve relacije, dok Dekartov proizvod kombinuje sve moguće parove elemenata iz dve relacije, bez obzira na uslove ili zajedničke atribute. Spajanje, presek i deljenje su izvedene binarne operacije. Spajanje kombinuje podatake iz različitih relacija na osnovu zajedničkih atributa, presek pronalazi n-torke koji postoje u obe relacije, a deljenje pronalazi skupove vrednosti koji zadovoljavaju određeni uslov.

3.1.1 SELEKCIJA (σ)

Selekcija (engl. *selection*) ili restrikcija predstavlja operaciju relacione algebre koja rezultira izdvajanjem određenih n-torki iz polazne relacije u skladu sa navedenim uslovom (izdvaja redove iz tabele). Uslov je definisan kao logički izraz koji se primenjuje na svaku *n*-torku u relaciji. Selekcija generiše rezultat u formi nove relacije sa istom strukturom kao i polazna. Rezultujuća relacija sadrži samo podskup *n*-torki koje zadovoljavaju specificiran uslov. Ukoliko ne postoje *n*-torke koje zadovoljavaju postavljeni uslov, rezultat operacije selekcije je prazan skup.

Operacija selekcije se može izraziti kao:

$\sigma_{P(X)}(r)$, *P(X)* uslov selekcije, a *r* naziv relacije.

Uslov selekcije može biti prost ili složen.

Prost uslov predstavlja logički izraz u jednoj od formi:

- A_i operator_poređenja A_j
- A_i operator_poređenja C,

gde *operator_poređenja* može da bude: $=, \neq, <, >, \leq, \geq$, A_i i A_j predstavljaju nazive atributa relacije nad kojom se selekcija izvodi, a C je konstanta koja uzima vrednosti iz domena vrednosti atributa A_i.

Složen uslov selekcije sadrži više prostih uslova koji su povezani logičkim operatorima: ∧ (*AND*), ∨(*OR*), ¬ (*NOT*).

Primer: Iz relacije *STUDENT* (StudentID, ImeS, SP, GodUpisa) izdvojiti sve studente koji ispunjavaju uslov:

a. SP = 'RT'

b. SP = 'RT' i GodUpisa = 2019

StudentID	**ImeS**	**SP**	**GodUpisa**
100	Milica	IS	2019
101	Lazar	RT	2024
102	Jovan	RT	2023
...	...		...
139	Aca	NRT	2018

Primena operacije selekcije:

a. $\sigma_{SP = 'RT'}(STUDENT)$
b. $\sigma_{SP = 'RT' \ AND \ GodUpisa = 2019}(STUDENT)$

Za slučaj pod *a.* operacija selekcije će izdvojiti sledeće *n*-torke:

StudentID	**ImeS**	**SP**	**GodUpisa**
101	Lazar	RT	2024
102	Jovan	RT	2023

Za slučaj pod *b.* operacija selekcije će izdvojiti prazan skup jer u relaciji ne postoje n-torke koje zadovaljavaju postavljen složen uslov da je SP = 'RT' i GodUpisa = 2019.

***Primer*:** Iz relacije *PREDMET* (PredmetID, Naziv, ESPB) izdvojiti predmete koji imaju 8 ESPB poena.

PredmetID	**Naziv**	**ESBP**
1000	Inženjerska matematika	7
1001	Elektrotehnika	7
1002	Engleski	4
1003	Baze podataka	8
...	...	...
1301	Nerelacione baze podataka	8
...	...	...
1503	Big data	9

Primena operacije selekcije:

$$\sigma_{ESPB\,=\,8}(PREDMET)$$

Operacija selekcije će izdvojiti sledeće *n*-torke:

PredmetID	Naziv	ESBP
1003	Baze podataka	8
1301	Nerelacione baze podataka	8

3.1.2 PROJEKCIJA (π)

Projekcija (engl. *project*) generiše relaciju koja sadrži samo određene atribute polazne relacije (izdvaja kolone iz tabele). Nova relacija koja se formira na osnovu projekcije se sastoji od *n*-torki koje sadrže samo izdvojene atribute. Redosled atributa u rezultujućoj relacije je definisan listom atributa projekcije. Skup atributa koji se izdvaja mora biti podskup skupa atributa polazne relacije. Vrednosti atributa u *n*-torkama nove relacije odgovaraju vrednostima tih atributa u polaznoj relaciji. Prilikom primene operacije projekcije, moguće je da više *n*-torki polazne relacije daje iste vrednosti za odabrane atribute. Međutim, pošto rezultat operacije projekcije mora biti relacija, u rezultujućoj relaciji uzima se samo jedna n-torka za svaku jedinstvenu kombinaciju vrednosti atributa.

Operacija projekcije se može prikazati na sledeći način:

$$\pi_{<A_k\,,\ldots,A_s>}(naziv\ relacije),$$

gde $< A_k, \ldots, A_s >$ predstavlja atribute nad kojima se vrši projekcija u relaciji za koju je specificiran naziv.

Primer: Primeniti operaciju projekcije nad relacijom *STUDENT* (StudentID, ImeS, SP, GodUpisa) po atributima ImeS i GodUpisa.

Operacija projekcije nad relacijom *STUDENT* po navedenim atributima

$$\pi_{ImeS,GodUpisa}\ (STUDENT\ )$$

kao rezultat daje relaciju koja sadrži samo podatke ImeS i GodUpisa.

ImeS	GodUpisa
Milica	2019
Lazar	2024
Jovan	2023
...	...
Aca	2018

Prethodno je navedeno da se u rezultujućoj relaciji prikazuju samo *n*-torke sa jedinstvenom kombinacijom atributa, što znači da se eliminišu duplikati. Ova osobina proizlazi iz toga što operacije relacione algebre tretiraju relacije kao skupove, te se pri projekciji vrednosti jednog ili više atributa prikazuju samo jedinstvene vrednosti. Međutim, važno je napomenuti da u slučaju *SQL* upitnog jezika ova osobina ne važi uvek.

U datom primeru, ako postoje studenti koji imaju isto ime i godinu upisa, projekcija samo po imenu studenta i godini upisa može dovesti do gubitka podataka.

Primer: Iz relacije *PREDMET* (PredmetID, Naziv, ESPB, ProfesorID) izdvojiti nazive predmeta i broj ESPB bodova koji su dodeljeni profesoru sa identifikacionim brojem 13.

Za rešavanje ovog primera neophodno je kombinovati primenu operacije selekcije i projekcije.

U ovom primeru, primena kombinovane operacije relacione algebre generiše rezultujuću operaciju *prof_13_pred*(Naziv,ESPB) u kojoj se nalaze podaci o nazivu predmeta i broju ESPB bodova koje predaje profesor sa identifikacionim brojem 13.

$\pi_{Naziv,ESPB}\left(\sigma_{ProfesorID\ =\ 13}(PREDMET)\right) \rightarrow$ prof_13_pred(Naziv,ESPB)

Primer: Iz relacije *PREDMET* (PredmetID, Naziv, ESPB, Status, ProfesorID) izdvojiti nazive predmeta, ESPB i status koji su dodeljeni profesoru sa identifikacionim brojem 13 i imaju više od 6 ESPB poena.

U ovom primeru, primena kombinovane operacije relacione algebre generiše rezultujuću operaciju *prof_13_espb* (Naziv,ESPB) u kojoj se nalaze podaci o nazivu predmeta i broju ESPB bodova za predmete koji imaju više od 6 ESPB bodova i predaje profesor sa identifikacionim brojem 13.

$\pi_{Naziv,ESPB}\left(\sigma_{ProfesorID\ =\ 13\ \wedge ESPB>6}(PREDMET)\right) \rightarrow$ prof_13_espb(Naziv,ESPB)

3.1.3 UNIJA (∪)

Unija, kao operacija relacione algebre, omogućava formiranje nove relacije spajanjem svih n-torki iz dve relacije. Rezultat unije relacija *r* i *s* je rezultujuća relacija koja sadrži n-torke koje se pojavljuju u relaciji *r* ili u relaciji *s* ili u obe relacije. Rezultujuća relacija ima nazive atributa prve relacije.

Operacija unije je moguća samo između unijski kompatibilnih relacija. **Unijski kompatibilne relacije** su dve relacije koje ispunjavaju uslove kojima se osiguravaju da se operacija unije može izvesti na logički ispravan način i odnose se na:

- **Isti broj atributa.** Prvi uslov je da obe relacije moraju imati isti broj atributa. To znači da se svaka *n*-torka u obe relacije može direktno uporediti po broju atributa.

- **Podudaranje atributa po značenju i tipu.** Iako atributi ne moraju imati iste nazive, moraju se podudarati po značenju i tipu.

Ovi uslovi osiguravaju da prilikom spajanja relacija unijom ne dolazi do gubitka podataka ili nepredvidivih rezultata.

Ako su *r* i *s* relacije nad šemom *R(X)* i *S(X)* i *X* predstavlja skup atributa koji su unijski kompatibilni u obe relacije, u nastavku je prikazana formalna definicija unije:

$r \cup s = \{x | \ x \in r \ \vee \ x \in s\}$.

Primer: Relacija *StudentiBaze* sadrži podatke o studentima koji su birali predmet Baze podataka, a relacija *StudentiProgramiranje* o studentima koji su birali predmet Programiranje. Neka je njihov trenutni sadržaj:

StudentiBaze

BrIndeks	**Prezime**	**Ime**
RT-1/22	Milović	Milica
RT-2/22	Petrović	Lazar
RT-3/22	Jovanović	Jovan

StudentiProgramiranje

BrInd	**PrezimeS**	**ImeS**
RT-2/22	Petrović	Lazar
RT-5/22	Marković	Marko
NRT-13/22	Janković	Janko
IS-1/22	Lazić	Ilija

Nakon primene operacije unije nad relacijima *StudentiBaze* i *StudentiProgramiranje* kreirana je rezultujuća relacija *StudentiBazeProgramiranje* tako da važi:

$$StudentiBaze \cup StudentiProgramiranje = StudentiBazeProgramiranje$$

Relacija *StudentiBazeProgramiranje* ima istu strukturu kao i polazne relacije. Sadrži podatke o studentima koji su birali ILI Baze podataka, ILI Programiranje ILI oba predmeta. Njen trenutni sadržaj je:

StudentiBazeProgramiranje

BrIndeks	**Prezime**	**Ime**
RT-1/22	Milović	Milica
RT-2/22	Petrović	Lazar
RT-3/22	Jovanović	Jovan
RT-5/22	Marković	Marko
NRT-13/22	Janković	Janko
IS-1/22	Lazić	Ilija

3.1.4 RAZLIKA(-)

Razlika relacija *r* i *s* je nova relacija koja sadrži sve n-torke koje se nalaze u prvoj relaciji, ali se ne pojavljuju u drugoj relaciji. Drugim rečima, iz prve relacije se izdvajaju sve n-torke koje su jedinstvene za tu relaciju u poređenju sa drugom relacijom. Razlika operacije se primenjuje nad dve polazne relacije koje moraju biti

unijski kompatibilne kako bi bila izvodljiva. Rezultujuća relacija ima nazive atributa prve relacije.

Ako su *r* i *s* relacije nad šemom *R(X)* i *S(X)* i *X* predstavlja skup atributa koji su unijski kompatibilni u obe relacije, u nastavku je prikazana formalna definicija razlike:

$r - s = \{x | \, x \in r \ \wedge \ x \notin s\}$.

Primer: Relacija *StudentiBaze* sadrži podatke o studentima koji su birali predmet Baze podataka, a relacija *StudentiProgramiranje* o studentima koji su birali predmet Programiranje. Neka je njihov trenutni sadržaj:

StudentiBaze

BrIndeks	**Prezime**	**Ime**
RT-1/22	Milović	Milica
RT-2/22	Petrović	Lazar
RT-3/22	Jovanović	Jovan

StudentiProgramiranje

BrInd	**PrezimeS**	**ImeS**
RT-2/22	Petrović	Lazar
RT-5/22	Marković	Marko
NRT-13/22	Janković	Janko
IS-1/22	Lazić	Ilija

Nakon primene operacije razlike nad relacijima *StudentiBaze* i *StudentiProgramiranje* kreirana je rezultujuća relacija *StudentiSamoBaze* tako da važi:

$$StudentiBaze - StudentiProgramiranje = \ StudentiSamoBaze$$

Relacija *StudentiSamoBaze* ima istu strukturu kao i polazne relacije. Sadrži podatke o studentima koji su birali Baze podataka ali nisu birali Programiranje. Njen trenutni sadržaj je:

StudentiSamoBaze

BrIndeks	**Prezime**	**Ime**
RT-1/22	Milović	Milica
RT-3/22	Jovanović	Jovan

3.1.5 PRESEK (∩)

Preseka relacija *r* i *s* je nova relacija koja sadrži samo one n-torke koje su zajedničke za obe relacije, odnosno koje se pojavljuju i u prvoj i u drugoj relaciji. Rezultat operacije preseka rezultira skupom n-torki koje se pojavljuju u obe relacije, a. rezultujuća relacija ima nazive atributa prve relacije. Ova operacija može se primeniti samo na unijski kompatibilne relacije.

Ako su *r* i *s* relacije nad šemom *R(X)* i *S(X)* i *X* predstavlja skup atributa koji su unijski kompatibilni u obe relacije, u nastavku je prikazana formalna definicija preseka:

$r \cap s = \{x | \, x \in r \ \wedge \ x \in s\}$.

Primer: Relacija *StudentiBaze* sadrži podatke o studentima koji su birali predmet Baze podataka, a relacija *StudentiProgramiranje* o studentima koji su birali predmet Programiranje.

Neka je njihov trenutni sadržaj:

StudentiBaze

BrIndeks	**Prezime**	**Ime**
RT-1/22	Milović	Milica
RT-2/22	Petrović	Lazar
RT-3/22	Jovanović	Jovan

StudentiProgramiranje

BrInd	**PrezimeS**	**ImeS**
RT-2/22	Petrović	Lazar
RT-5/22	Marković	Marko
NRT-13/22	Janković	Janko
IS-1/22	Lazić	Ilija

Nakon primene operacije preseka nad relacijima *StudentiBaze* i *StudentiProgramiranje* kreirana je rezultujuća relacija *StudentiBazeProg* tako da važi:

$$StudentiBaze \cap StudentiProgramiranje = StudentiBazeProg$$

Relacija *StudentiBazeProg* ima istu strukturu kao i polazne relacije. Sadrži podatke o studentima koji su birali i Baze podataka i Programiranje. Njen trenutni sadržaj je:

StudentiBazeProg

BrIndeks	**Prezime**	**Ime**
RT-2/22	Petrović	Lazar

3.1.6 DEKARTOV PROIZVOD (×)

Dekartov proizvod relacija r i s je nova relacija $r \times s$ koja obuhvata sve moguće kombinacije parova n-torki. Rezultujuća relacija predstavlja skup n-torki dobijenih kombinovanjem svake n-torke iz relacije r sa svim n-torkama iz relacije s. Šema rezultujuće relacije uključuje sve atribute iz oba polazne relacije.

Formalna definicija Dekartovog proizvoda relacija *r* i *s* nad šemama *R(X)* i *S(Y),* gde su *X* i *Y* su skupovi atributa u šemama relacija, izražava se kao:

$r \times s = \{xy | \, x \in r \ \wedge \ x \in s\}$,

xy predstavlja kombinaciju svake n-torke *x* iz relacije r sa svakom n-torkom *y* iz relacije *s.*

Primer: Iz relacija PROFESOR(ProfID, Prezime, Ime) i PREDMET(PredID, Naziv, ProfID) izdvojiti podatke o profesorima i predmetima koji su im dodeljeni.

Trenutni sadržaj relacija PROFESOR i PREDMET je prikazan u nastavku:

Profesor (S)

ProfID	**Prezime**	**Ime**
10	Milojković	Ana
11	Petrović	Ivan
12	Stamenković	Milica
13	Marković	Marko

Predmet (P)

PredID	**Naziv**	**ProfID**
1000	Inženjerska matematika	10
1003	Baze podataka	12
1301	Nerelacione baze podataka	12
1501	Programiranje	NULL

Primenom operacije Dekartovog proizvoda dobija se nova relacija $Profesor \times Predmet$ koja sadrži sledeće atribute: S.ProfID, Prezime, Ime, PredID, Naziv, P.ProfID. Trenutni sadržaj rezultujuće relacije je prikazan u nastavku:

$Profesor \times Predmet$

S.ProfID	**Prezime**	**Ime**	**PredID**	**Naziv**	**P.ProfID**
10	Milojković	Ana	1000	Inženjerska matematika	10
10	Milojković	Ana	1003	Baze podataka	12
10	Milojković	Ana	1301	Nerelacione baze podataka	12
10	Milojković	Ana	1501	Programiranje	NULL
11	Petrović	Ivan	1000	Inženjerska matematika	10
11	Petrović	Ivan	1003	Baze podataka	12
11	Petrović	Ivan	1301	Nerelacione baze podataka	12
11	Petrović	Ivan	1501	Programiranje	NULL
12	Stamenković	Milica	1000	Inženjerska matematika	10
12	Stamenković	Milica	1003	Baze podataka	12
12	Stamenković	Milica	1301	Nerelacione baze podataka	12
12	Stamenković	Milica	1501	Programiranje	NULL
13	Marković	Marko	1000	Inženjerska matematika	10
13	Marković	Marko	1003	Baze podataka	12
13	Marković	Marko	1301	Nerelacione baze podataka	12
13	Marković	Marko	1501	Programiranje	NULL

Iz prikaza rezultujuće relacije može se zaključiti da nakon primene Dekartovog proizvoda, samo neke od *n*-torki sadrže tražene podatke.

3.2 SPAJANJE

Spajanje je operacija relacione algebre koja kombinuje podatke iz dve polazne relacije na osnovu zajedničkih atributa ili uslova. Rezultat operacije spajanja je nova relacija koja sadrži kombinovane n-torke dobijene iz polaznih relacija. Spajanje se obično izvodi tako što se svaka n-torka jedne relacije spoji sa svakom n-torkom druge relacije, pri čemu se uzimaju u obzir kriterijumi spajanja, kao što su jednakost vrednosti određenih atributa ili zadovoljavanje određenih uslova. Nakon spajanja, rezultujuća relacija ima strukturu koja je kombinacija šema polaznih relacija, a sadrži samo one n-torke koje zadovoljavaju uslove spajanja.

3.2.1 Θ-SPOJ

Θ-spoj je operacija relacione algebre koja kombinuje dve relacije *r* i *s* nad njihovim šemama $R(A_1, A_2, \dots A_n)$ i $S(B_1, B_2, \dots B_m)$ kako bi formirala novu relaciju q nad šemom $Q(A_1, A_2, \dots A_n, B_1, B_2, \dots B_m)$. Rezultujuća relacija q ima po jednu *n*-torku za svaku kombinaciju iz *r* i *s*, kada god ta kombinacija *n*-torki zadovoljava uslov spoja.

Glavna razlika između θ-spoj i Dekartovog proizvoda je ta što *Θ-spoj* uzima u obzir samo one kombinacije *n*-torki koje zadovoljavaju određeni uslov, dok Dekartov proizvod kombinuje sve moguće kombinacije *n*- torki bez obzira na uslove.

Spoj se označava sa:

$$r \bowtie_{<uslov\ spoja>} s,$$

gde uslov spoja predstavlja logički izraz u formatu:

$$uslov_1 \wedge (\vee) uslov_2 \dots \wedge (\vee) uslov_k ,$$

a uslov je oblika $A_i \theta\ B_j$, $\theta \epsilon \{=, \neq, <, >, \leq, \geq\}$

Θ-spoj može da bude realizovan kao Ekvi spoj (*Equijoin*) ili kao prirodni spoj (*Natural join*).

3.2.2 EKVI SPOJ (EQUIJOIN)

Ekvi spoj predstavlja varijaciju *Θ-spoja* gde je uslov spoja izražen kao $A_i = B_j$, što znači da se spajanje vrši na osnovu jednakosti vrednosti odgovarajućih atributa. Ekvi spajanje može biti zasnovano na spajanju po jednom ili više atributa. Ekvi spoj se često koristi u upitima relacione algebre i čini osnovu za implementaciju spoja u *SQL* upitima.

Primer: Iz relacija *PROFESOR*(ProfID, Prezime, Ime) i *PREDMET*(PredID, Naziv, ProfID) izdvojiti podatke o profesorima i predmetima koji su im dodeljeni.

Rezultat spoja relacija *PROFESOR* (označićemo je sa S) i *PREDMET* (označićemo je sa P) zasnovan je na uslovu spoja koji definiše da su vrednosti atribut ProfID jednake u obe relacije.

$$S \bowtie_{S.ProfID=P.ProfID} P$$

U rezultujućoj relaciji redosled atributa zavisi od redosleda navođenja relacija u spoju. Prvo se navode atributi relacije koja je sa leve strane operacije (u ovom primeru to je relacija *PROFESOR*), a nakon toga iz relacije sa desne strane operacije (u ovom primeru to je *PREDMET*). Atribut ProfID se pojavljuje na dva mesta. Sadržaj relacija *PROFESOR* i *PREDMET* je prikazan u nastavku:

Profesor (S)

ProfID	**Prezime**	**Ime**
10	Milojković	Ana
11	Petrović	Ivan
12	Stamenković	Milica
13	Marković	Marko

Predmet (P)

PredID	**Naziv**	**ProfID**
1000	Inženjerska matematika	10
1003	Baze podataka	12
1301	Nerelacione baze podataka	12
1501	Programiranje	NULL

Primenom operacije Ekvi spoja dobija se nova relacija koja sadrži sledeće atribute: S.ProfID, Prezime, Ime, PredID, Naziv, P.ProfID. Sadržaj rezultujuće relacije je prikazan u nastavku:

$$S \bowtie_{S.ProfID=P.ProfID} P$$

S.ProfID	**Prezime**	**Ime**	**PredID**	**Naziv**	**P.ProfID**
10	Milojković	Ana	1000	Inženjerska matematika	10
12	Stamenković	Milica	1003	Baze podataka	12
12	Stamenković	Milica	1301	Nerelacione baze podataka	12

3.2.3 PRIRODNI SPOJ (NATURAL JOIN)

Prirodni spoj je vrsta spoja u kojoj se iz rezultata isključuje jedan od dva identična atributa ($A_i ili\ B_j$). Ovaj spoj se često označava sa "*", ili se koristi simbol za spoj bez eksplicitnog navođenja uslova. Ključno je da bi prirodni spoj bio moguć, oba atributa u uslovu spoja moraju imati ista imena. Ukoliko imena nisu identična, koristi se operacija preimenovanja atributa kako bi se uskladili za spoj.

Primer: Ilustracija prirodnog spoja prikazana je na osnovu prethodnog primera spajanja relacija *PROFESOR*(ProfID, Prezime, Ime) i *PREDMET*(PredID, Naziv, ProfID)

Atribut na osnovu kog se vrši prirodni spoj je ProfID – njegov naziv je isti u obe relacije, a uslov je jednakost vrednosti ovih atributa u obe relacije. Primenom prirodnog spajanja dobija se nova relacija koja sadrži atribute: ProfID, Prezime, Ime, PredID, Naziv. Redosled atributa u rezultujućoj relaciji je kao kod spoja - navode se najpre atributi iz relacije sa leve strane operacije (*PROFESOR*), a nakon toga iz relacije sa desne strane (*PREDMET*). Atribut ProfID se pojavljuje samo na jednom mestu koje mu je određeno pozicijom u relaciji koja se navodi s leve strane operacije prirodnog spoja.

Prirodni spoj između relacija *PROFESOR* i *PREDMET* može se prikazati kao:

$$Profesor \bowtie Predmet$$

$$Profesor * Predmet$$

Sadržaj rezultujuće relacije je prikazan u nastavku:

Profesor ⋈ Predmet ili ***Profesor * Predmet***

ProfID	Prezime	Ime	PredID	Naziv
10	Milojković	Ana	1000	Inženjerska matematika
12	Stamenković	Milica	1003	Baze podataka
12	Stamenković	Milica	1301	Nerelacione baze podataka

3.2.4 SPOLJAŠNJI SPOJ (OUTER JOIN)

Spoljašnji spoj (Outer join) je operacija koja se zasniva na uslovu spoja koji se odnosi na jednakost atributa, slično kao kod klasičnog spoja. Rezultujuća relacija zavisi od načina realizacije spoljašnjeg spoja. Može uključivati *n*-torke iz neke od relacija čak i ako one ne ispunjavaju uslov jednakosti.

Na osnovu načina realizacije postoje tri varijante spoljašnjeg spoja:

- **Levi (Left) spoj (⟕).** Rezultujuća relacija sadrži sve n-torke iz leve relacije, bez obzira na to da li ispunjavaju uslov spoja i n-torke iz desne relacije koje zadovoljavaju uslov. Za n-torke s leve strane koje ne ispunjavaju uslov, upisuje NULL vrednost na desnoj strani.
- **Desni (Right) spoj (⟖).** Rezultujuća relacija sadrži sve n-torke iz desne relacije, bez obzira na to da li ispunjavaju uslov spoja i n-torke iz leve relacije koje zadovoljavaju uslov. Za n-torke s desne strane koje ne ispunjavaju uslov, upisuje NULL vrednost na levoj strani.
- **Puni (Full) spoj (⟗).** Rezultujuća relacija sadrži sve n-torke bez obzira na to da li ispunjavaju uslov spoja. Neuparene n-torke dobijaju NULL vrednost na suprotnoj strani.

Spoljašnji spojevi se primenjuju i u slučaju kada relacije nisu potpuno kompatibilne. Omogućavaju uniju i svih *n*-torki iz relacija koje su delimično kompatibilne, što znači da su neki atributi međusobno kompatibilni, dok drugi nisu.

3.2.5 DELJENJE (/)

Deljenje je najkompleksnija operacija relacione algebre i ne može se primeniti na proizvoljne relacije. Da bi operacija deljenja $r \,/\, s$ bila moguće, svi atributi relacije *r* moraju biti prisutni u relaciji *s*. Deljenje nije iz osnovnog skupa operacija i može se izraziti pomoću operacije projekcije, Dekartovog proizvoda i razlike na sledeći način:

$$\pi_y(r) \rightarrow q_1$$

$$\pi_y(s \times q_1) - r) \rightarrow q_2$$

$$(q_1 - q_2) \rightarrow q$$

Primer: U ovom primeru je ilustrovana operacija deljenja nad relacijama *r* i *s*.

r

A	**B**	**C**
Ana	RT	Baze podataka
Marko	NRT	Programiranje
Ivan	IS	Matematika
Ana	RT	Programiranje
Željko	IS	Baze podataka
Ana	RT	Matematika

s

C
Baze podataka
Programiranje
Matematika

r/s

A	**B**
Ana	RT

Rezultat operacije deljenja sadrži jedino *n*-torku u relaciji *r* nad atributima *A* i *B*: *(Ana,RT)* zato što je u kombinaciji sa sve tri *n*-torke nad atributom *C* u relaciji *s*.

3.3 PITANJA I ZADACI

1. Navedi elementarne operacije relacione algebre.
2. Navedi izvedene operacije relacione algebre.
3. Koje operacije spadaju u kategoriju unarnih, a koje u kategoriju binarnih operacija?
4. Šta je prost uslov, a šta složen uslov?
5. Za koje relacije se može reći da su unijski kompatibilne relacije?
6. Relacija *predmet* definisana je šemom PREDMET (PID, Naziv, ESPB, ProfID). Trenutni sadržaj relacije je prikazan u nastavku:

PredID	**Naziv**	**ESPB**		**ProfID**
1000	Inženjerska matematika	5		10
1003	Baze podataka	6		12
1301	Nerelacione baze podataka	7		12
1501	Programiranje	6		17

Prikazati rezultat operacije $\sigma_{ESPB>6\ AND\ ProfID=12}(PREDMET)$

7. Za relaciju iz zadataka 6 napisati operaciju projekcije koja prikazuje podatke o nazivu predmeta, broju ESPB i identifikacionom broju predmetnog profesora.
8. Iz relacije PREDMET (PredmetID, Naziv, ESPB, Status, ProfesorID) izdvojiti nazive predmeta i broj ESPB bodova koji imaju status obaveznih predmeta.
9. Iz relacija SMER(IDSP, NazivS, Oznaka) i STUDENT(IDS, Prezime, Ime, IDSP) izdvojiti podatke o studentima i nazivima studijskih programa na koji su upisani.
10. Koju operaciju spajanja treba primeniti za prikaz podataka o smerovima uključujući i smerove na kojima nema upisanih studenata?

4 DIZAJN BAZE PODATAKA

Cilj ovog poglavlja je detaljno objašnjenje procesa modelovanja i projektovanja baza podataka. Istaknuti su ključni koraci neophodni za kreiranje konceptualnog modela. Fokus je na objašnjenju semantičkog modela entiteta i veza podataka na konceptualnom nivou. Detaljno su analizirani koncepti entitet, atribut, domen, uz poseban osvrt na slučajeve gde se entiteti mogu definisati kao atributi i obrnuto. Ukazano je na razliku između pojma entiteta i instance. Detaljno su objašnjeni koncepti veza između entiteta, ograničenja i tipovi veza. Nadalje, razmotrene su različite grafičke notacije semantičkog modela entiteta i veza.

4.1 MODELOVANJE I PROJEKTOVANJE BAZE PODATAKA

Prilikom kreiranja baze podataka, lako je upasti u zamku pokušavajući na brzinu odraditi posao, bez prethodno sprovedenog prosesa dizajniranja i projektovanja. Značaj ovih postupka najbolje se može videti nakon završetka projekta, kada je zbog loše dizajnirane baze podataka neophodno često raditi skupe i komplikovane redizajne i reimplementacije.

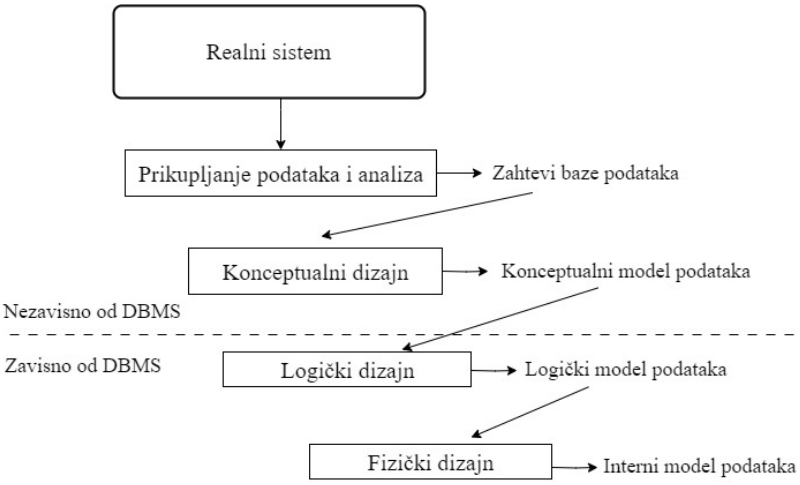
Dizajniranje i projektovanje baze podataka je slično izgradnji kuće. Nerealno je početi sa zidanjem bez prethodno urađenog detaljnog planiranja. Takođe, dobar dizajn omogućava da kasnije originalnu građevinu proširite, odnosno dozidate, bez potrebe za rušenjem prethodno izgrađenog.

Iako se termini "projektovanje baza podataka" i "modelovanje baza podataka" često koriste kao sinonimi, postoje određene nijanse u njihovim značenjima.

Modelovanje baza podataka obuhvata proces stvaranja apstraktnih prikaza podataka, obično u formi dijagrama, koji predstavljaju entitete, atribute i njihove međusobne veze. Ovaj konceptualni model služi kao osnova za dalji razvoj baze podataka, ali ne uključuje sve tehničke detalje implementacije.

Projektovanje baza podataka podrazumeva prelazak sa konceptualnog modela na konkretan plan ili šemu koja će biti implementirana u bazi podataka koja je implementirana u realnom okruženju. Uzimaju se u obzir tehnički aspekti poput tipova podataka, indeksiranja, normalizacije, referencijalnog integriteta i drugih detalja potrebnih za efikasno funkcionisanje baze podataka.

Na 4.1 je prikazan dijagram koji obuhvata postupak modelovanja i projektovanja baza podataka.



Slika 4.1 Proces modelovanja i projektovanja baza podataka

Sa slike 4.1 se može videti da je modelovanje baza podataka proces koji se fokusira na apstrakciju podataka i njihovih odnosa, dok je projektovanje baza podataka konkretnija faza koja se bavi tehničkim detaljima implementacije na osnovu konceptualnog modela.

4.1.1 MODELOVANJE BAZE PODATAKA

Modelovanje baze podataka predstavlja proces koji u suštini obuhvata tri ključna koraka:

- **Izbor.** Bitni objekti se izdvajaju iz mnoštva objekata u stvarnom svetu, formirajući manji skup objekata koji čine osnovu modela. Ovaj proces ima za cilj smanjenje složenosti stvarnog sistema. Selekcija obuhvata ne samo same objekte, već i njihove karakteristike, kao i međusobne veze između objekata.
- **Dodeljivanje imena**. Svakom objektu, vezi između objekata i svojstvu (atributu) dodeljuju se odgovarajuća imena. Ovaj korak omogućava precizno identifikovanje i razumevanje svakog elementa unutar modela. Imenovanje obuhvata sve, od konkretnih objekata do apstraktnih veza.
- **Klasifikacija.** Nehomogeni skup objekata i njihovih odnosa klasifikuju se u homogene klase i tipove objekata. Klasifikacija zavisi od specifičnog područja primene modela. Razvrstavanjem elemenata u određene kategorije omogućava se organizacija i lakše razumevanje kompleksnosti sistema.

Navedena tri koraka procesa modelovanja se ostvaruju kroz fazu analize zahteva i kreiranja konceptualnog modela. Ovaj pristup omogućava strukturirano pristupanje

razvoju baze podataka i osigurava da model adekvatno odražava potrebe stvarnog sistema.

Analiza zahteva predstavlja ključnu fazu u procesu modelovanja baza podataka. Pre nego što se razmotri kako će se sistem implementirati, važno je prvo razumeti šta je tačno potrebno postići. Ova faza uključuje detaljno prikupljanje neophodnih podataka i sprovođenje temeljne analize kako bi se jasno definisali zahtevi sistema. Sakupljanje relevantnih podataka i njihova analiza omogućavaju identifikovanje ključnih elemenata koji će činiti osnovu baze podataka. Tokom ove faze, sprovodi se dijalog sa krajnjim korisnicima i zainteresovanim stranama kako bi se bolje razumele potrebe realnog sistema. Identifikuju se ključni objekti, njihovi atributi i međusobne veze. Ovaj proces omogućava stvaranje jasnog konceptualnog modela koji će odražavati stvarne potrebe organizacije ili sistema.

Analiza zahteva postavlja temelj za sledeću faze modelovanja odnosno izradu konceptualnog dizajna, kao i za logičko i fizičko projektovanje baze podataka. Jasno definisani zahtevi omogućavaju precizno oblikovanje strukture baze podataka i implementaciju sistema koji efikasno podržava poslovne procese organizacije.

4.1.2 KONCEPTUALNI MODEL

Na osnovu prikupljenih i analiziranih zahteva, pristupa se stvaranju konceptualnog modela. Ovde se koriste alati poput dijagrama entiteta-veza (ER) za vizualizaciju ključnih entiteta, atributa i njihovih međusobnih veza. Cilj je stvoriti apstraktnu reprezentaciju podataka bez obzira na tehničke detalje.

Konceptualni model obuhvata ključne aspekte sistema:

- **Struktura podataka.** Statistički opis stanja sistema koji čine objekti, njihove karakteristike, veze između objekata i njihove osobine.
- **Operacije nad strukturom modela.** Definiše radnje koje prouzrokuju kretanje i promene u sistemu, odnosno dinamiku realnog sistema.
- **Ograničenja.** Pravila koja razdvajaju dozvoljena od nedozvoljenih stanja u realnom sistemu. Ograničenja su inherentni deo strukture modela podataka.
- **Posledice ograničenja iz realnog sveta.** Refleksija načina na koji ograničenja utiču na stvarni sistem, kako bi se obezbedila doslednost i tačnost informacija.

Konceptualni model nije vezan za određeni sistem za upravljanje bazama podataka (*DBMS*) i ne određuje fizički oblik u kojem se podaci čuvaju.

Proces izrade konceptualnog modela uključuje sledeće korake:

- **Identifikacija entiteta**. Kroz ključne reči tokom intervjua sa korisnicima, određuju se objekti koji će biti deo modela, uzimajući u obzir podatke koji treba da budu skladišteni.
- **Identifikacija atributa.** Definisanje karakteristika svakog entiteta.

- **Identifikacija ključeva.** Određivanje ključeva koji će jedinstveno identifikovati svaki entitet.
- **Definisanje veza.** Utvrđivanje odnosa između entiteta.
- **Određivanje kardinalnosti veza.** Preciziranje broja veza između entiteta.
- **Normalizacija.** Proces uklanjanja redundancije i poboljšanja strukture podataka.
- **Definisanje referencijalnog integriteta veza.** Obezbeđenje konzistentnosti među entitetima.

Ovi koraci omogućavaju stvaranje konceptualnog modela koji precizno odražava potrebe i strukturu stvarnog sistema.

4.1.3 PROJEKTOVANE BAZE PODATAKA

Projektovanje baze podataka predstavlja suštinski deo procesa razvoja informacionih sistema i obuhvata niz ključnih faza koje se temelje na konceptualnom modelu, proizašlom iz prethodnog procesa modelovanja. Ovaj proces ima za cilj pretvaranje apstraktnog konceptualnog modela u praktičnu i efikasnu strukturu podataka, prilagođenu za skladištenje i upravljanje informacijama.

Faze projektovanja baze podataka uključuju:

- **Logičko projektovanje:**
  - Identifikacija entiteta, atributa, ključeva i njihovih međusobnih odnosa iz konceptualnog modela.
  - Definisanje strukture podataka kroz određivanje tabela, veza i ograničenja.
  - Primena normalizacije kako bi se elimisala redundancija i postigla efikasnost.
- **Fizičko projektovanje**:
  - Definisanje konkretnih tehničkih detalja kao što su tipovi podataka, veličine polja, indeksiranje i druge karakteristike.
  - Razmatranje performansi baze podataka i optimizacija strukture za brži pristup i upite.
  - Razmatranje faktora kao što su sigurnost, integritet podataka i pristup podacima.
- **Implementaciju:**
  - Kreiranje baze podataka na osnovu definisanog logičkog i fizičkog modela.
  - Uvođenje podataka u bazu putem odgovarajućih interfejsa.
  - Implementacija sigurnosnih mehanizama i kontrola pristupa.
- **Održavanje:**
  - Periodično ažuriranje šeme baze podataka kako bi se prilagodila promenama u zahtevima sistema.
  - Praćenje performansi i optimizacija kako bi se održala efikasnost sistema.

- Rešavanje problema i implementacija izmena u skladu sa potrebama korisnika.

Projektovanje baze podataka je esencijalna faza u životnom ciklusu informacionih sistema, jer pravilno dizajnirana baza podataka direktno utiče na performanse, doslednost i tačnost podataka u celokupnom informacionom sistemu. Ovaj proces zahteva temeljno razumevanje potreba organizacije, preciznost u definisanju strukture podataka i kontinuirano praćenje promena kako bi se osigurala dugotrajna funkcionalnost sistema.

4.2 ENTITY RELATIONSHIP (ER) MODEL

Model entiteta i veza (engl. *Entity Relationship Model*, ER model) predstavlja jedan od najrasprostranjenijih semantičkih modela podataka koji se primenjuje na konceptualnom nivou. Razvijen je kako bi olakšao dizajn baza podataka omogućavajući grafički prikaz putem ER dijagrama za konceptualno predstavljanje strukture podataka. Prvi put je predstavljen od strane Chena 1976. godine, a od tada su razvijene različite verzije, kao i različite grafičke notacije. U literaturi se često može naići i na alternativne nazive, kao što su "model entiteta i veza" ili "model objekti-veze".

U ER modelu, struktura podataka se predstavlja grafički kao "dijagram entitet-veza" koristeći tri osnovna tipa elemenata: skupove entiteta, atribute i veze. Osnovna ideja zasnovana na je identifikaciji entiteta (objekata realnog sveta) i veza između njih unutar određenog sistema ili segmenta stvarnog sveta koji se mapira u bazu podataka. Entiteti mogu predstavljati realne subjekte, objekte, događaje, pojave ili apstraktne pojmove u okviru mini sveta. ER model se zasniva na detaljnom proučavanju entiteta i njihovih međusobnih odnosa. Obezbeđuje jasno definisanje strukture podataka, čime se olakšava implementacija baze podataka i omogućava efikasno upravljanje podacima u sistemu.

Struktura ER model obuhvata osnovne koncepte poput entiteta, tipova entiteta, skupova entiteta, atributa, veza, tipova veza i skupova veza. Ovaj model pruža osnovni okvir za definisanje strukture podataka i međusobnih odnosa unutar baze podataka. Što se tiče ograničenja, ER model ima dobro definisana pravila unutar komponente integriteta. Ova ograničenja se obično eksplicitno definišu i odnose se na integritet domena (dozvoljeni opseg vrednosti atributa), kardinalnost tipa veza (broj entiteta povezanih putem veze), participaciju tipa veza (obavezno ili opciono učešće entiteta u vezi), slabi tip entiteta (entitet koji ne može postojati bez povezanog entiteta) i ključ tipa entiteta (identifikacioni atributi koji jedinstveno identifikuju entitet). Ova ograničenja igraju ključnu ulogu u očuvanju doslednosti i tačnosti podataka u okviru baze podataka.

4.2.1 ENTITET

Entitet predstavlja objekat posmatranja koji se može jednoznačno identifikovati i razlikovati. Ovaj pojam ima ključnu ulogu u bazi podataka, pružajući konceptualni okvir za skladištenje i praćenje odgovarajućih podataka. Entiteti su ograničeni na objekte koji se u relacionoj bazi podataka predstavljaju upotrebom tabela u narednim fazama razvoja. Svaki entitet mora imati jedinstveno ime, osiguravajući da ne mogu postojati dva identična entiteta u okviru iste baze podataka.

Entitet može označavati fizički prisutan objekat, kao što su stvarni entiteti (npr. osoba, vozilo, kuća ili zaposleni), ili konceptualno postojan objekat, kao što su apstraktni entiteti (npr. kompanija, radno mesto ili predmet na univerzitetu).

Stvarni entiteti mogu biti prirodni, poput reka, planina, mora, šuma, ili veštački, kao što su telefonske ili vodovodne instalacije, putevi, vozila, zgrade.

Apstraktni entiteti obuhvataju različite kategorije, uključujući događaje (npr. osiguranje vozila, popis stanovništva, prodaja robe, uplate na žiro račun, polaganje ispita, overa semestra, upis na fakultet), pojmove (npr. nauka, znanje, učenje, obuka, profit, republika, opština, kompanija, vremenska prognoza), stanja (npr. radi na projektu, prisustvuje predavanju, boravi u bolnici) i uloge (npr. službenik, klijent, doktor, pacijent, zaposleni, profesor, student, stanovnik).

Potencijalni entiteti u stvarnom sistemu mogu obuhvatiti:

- Organizacione jedinice poput fakulteta, katedre, biblioteke itd.
- Lokacije kao što su učionice, laboratorije, trgovinske radnje, banke itd.
- Različite uloge kao što su radnik, službenik, student, profesor, stanovnik itd.
- Događaji koji se zabeležavaju poput ispita, utakmica, predavanja, glasanja i slično.
- Različite uređaje kao što su računari, skeneri, radari, antene, senzori itd.

Primer: Potencijalni entiteti za deo realnog sistema Fakultet:

- STUDENT
- STUDIJSKI PROGRAM
- PREDMET
- PROFESOR
- PREDAVANJE
- PROSTORIJA
- ISPIT

Primer: Potencijalni entiteti za deo realnog sistema Hotel:

- SOBA
- GOST
- REZERVACIJA

- ZAPOSLENI
- RESTORAN

Primer: Potencijalni entiteti za deo realnog sistema Ordinacija:

- PACIJENT
- LEKAR
- PREGLED
- KARTON

Skup entiteta

Skup entiteta obuhvata sve entitete koji pripadaju određenom tipu entiteta u datom trenutku u bazi podataka. Na primer, skup entiteta "Studenti" može sadržavati sve entitete koji predstavljaju studente određenog fakulteta. Entiteti unutar istog skupa dele slične karakteristike ili pripadaju istom konceptualnom tipu. Na primer, entiteti u skupu "Studenti" mogu deliti zajedničke atribute kao što su "Ime", "Broj Indeksa" itd.

Pojam skupa entiteta često se koristi kako bi se organizovali i klasifikovali entiteti na način koji olakšava manipulaciju podacima u bazi. Svaki entitet u skupu poseduje određene atribute koji ga karakterišu. Skup entiteta nije statičan i može se menjati tokom vremena u skladu sa dodavanjem, ažuriranjem ili brisanjem entiteta iz baze podataka.

U procesu modelovanja često se koristi termin skup entiteta u apstraktnom smislu, ne referišući se na određeni skup pojedinačnih entiteta. Termin proširenje skupa entiteta ukazuje na stvarnu kolekciju entiteta koji pripadaju skupu entiteta. Na taj način, skup profesora na fakultetu čini proširenje skupa entiteta "profesor". Skupovi entiteta ne moraju biti disjunktni. Na primer, moguće je definisati skup entiteta svih osoba na univerzitetu (osoba). Entitet osobe može biti profesor, student, oboje ili ništa od navedenog.

4.2.2 INSTANCA

Entiteti imaju instance, koje predstavljaju konkretne pojave ili jedinstvene pojavne oblike tog entiteta. Svaka instanca entiteta je specifična pojava ili konkretan primer entiteta u stvarnom svetu. Na primer, entitet "Student" ima svoje instance koje predstavljaju konkretne studente na određenom fakultetu.

Svaki student je jedna instanca entiteta "Student" i karakteriše ga jedinstven niz atributa, kao što su ime, broj indeksa, datum rođenja itd. Instances entiteta omogućavaju konkretizaciju i praćenje stvarnih entiteta u bazi podataka.

Na slici 4.2.2.1 su dati primeri entiteta i njihovih mogućih instanci.

Entitet	**Instanca**
OSOBA	Ana Petrović, Marko Jovanović, Elena Popović...
BREND	Nike, Champion, Adidas...
PROIZVOD	Cipele, Čizme, Patike...
POSAO	Električar, Vozač, Bravar...
ŽIVOTINJA	Pas, Mačka, Krava...
PUBLIKACIJA	Knjiga, Članak, E – knjiga....

Slika 4.2.2.1 Primeri entiteta i njihovih mogućih instanci

Razmišljanje o tome da li neki pojam predstavlja entitet ili instancu zavisi od konteksta sistema za koji se modeluje baza podataka. Uobičajeno, entiteti su opšti pojmovi koji obuhvataju više instanci, dok su instance konkretne pojave ili pojedinci unutar tih opštih pojmova.

U primeru psa, ako je potrebno pratiti različite vrste životinja, "ŽIVOTINJA" bi bio entitet, a "PAS" bi bio jedna od instanci tog entiteta. Ako je fokus na različitim rasama pasa, "PAS" bi bio entitet, a instance bi bile različite rase, kao što su "TERIJER", "PUDLA", "LABRADOR", itd.

Važno je jasno definisati entitete i odrediti kako se oni odnose jedni prema drugima, uzimajući u obzir specifičnosti sistema koji se modeluje. Ovakav pristup doprinosi preciznosti i doslednosti u strukturi baze podataka.

4.2.3 ATRIBUTI

Entitet je predstavljen skupom atributa. Atributi su opisna svojstva koja poseduje svaki član skupa entiteta. Atribut se može smatrati karakteristikom entiteta koji se posmatra, prikupljajući podatke o njemu. Svi entiteti unutar jednog skupa dele barem jedno zajedničko svojstvo, na osnovu kojeg su svrstani u istu grupu.

Svaki atribut ima jedinstvenu vrednost za jednu instancu entiteta. Određivanje atributa za skup entiteta izražava da baza podataka čuva slične informacije o svakom entitetu u skupu entiteta; međutim, svaki entitet može imati svoju vrednost za svaki atribut. Svaki entitet ima vrednost za svaki svoj atribut.

Atribut predstavlja specifičan deo informacije koji može:

- opisivati entitet,
- kvantifikovati entitet,

- kvalifikovati entitet,
- klasifikovati entitet,
- specifikovati entitet.

Atributi imaju vrednosti, koje mogu biti brojevi, nizovi znakova, datumi, zvukovi, slike, i drugi formati podataka. Vrednost atributa se određuje dodelom odgovarajućeg tipa podatka.

Primer: Atributi entiteta

- Entitet RADNIK može imati svojstva, odnosno karakteristike koji predstavljaju atribute: MatičniBrojRadnika, Ime, Prezime, DatumRođenja,…
- Entitet STUDENT može imati atribute: MatičniBrojStudenta, Ime, Prezime, BrojIndeksa,...

Mogući atributi skupa entiteta RADNIK su Id, NazivOdeljenja, Plata. U realnom sistemu, postojali bi dodatni atributi kao što su BrojUlice, BrojStana, Država, PoštanskiBroj i Zemlja, ali ih izostavljamo kako bismo održali jednostavnost primera.

Svaki konkretan entitet ima vrednosti za svaki od atributa koji ga opisuje. Na primer, jedan od studenta ima broj ideksa 13, njegovo ime je Lazar (odnosno vrednost atributa Ime je Lazar), prezime Petrović, godina studija je prva.

Relevantni /Nerelevantni atributi

Relevantni atributi entiteta su oni koji su bitni i značajni za opisivanje i identifikaciju entiteta u određenom kontekstu. Ovi atributi pružaju važne informacije i doprinose razumevanju entiteta u okviru sistema ili baze podataka. Nerelevantni atributi entiteta su oni koji, iako postoje kao svojstva entiteta, nisu od suštinskog značaja za opisivanje ili identifikaciju entiteta u datom kontekstu. U nekim situacijama, određeni atributi mogu biti nevažni ili nepotreban teret informacija.

Na primer, u kontekstu entiteta "Student", relevantni atributi mogu uključivati ime, prezime, broj indeksa, godinu studija, dok bi neki kontakt podaci ili boja očiju mogli biti smatrani nerelevantnim u okviru sistema za evidenciju studenata. Važno je odabrati relevantne atribute kako bi se održala efikasnost i preciznost u radu sa bazom podataka

Primer: Posmatrajmo entitet RADNIK

Sa aspekta zarade, atribut "adresa" nije relevantan atribut, ali sa aspekta organizacije prevoza atribut "adresa" predstavlja relevantan atribut.

Primer: Posmatrajmo entitet ODELJENJE

Sa aspekta odeljenja za plate, atribut "vrsta bolesti" nije relevantan, ali sa aspekta zdravstvenog odeljenja istog preduzeća predstavlja relevantan atribut.

Mera relevantnih atributa odnosi se na procenu važnosti i značaja pojedinih atributa u sklopu određenog konteksta ili sistema. Ova mera pomaže u identifikaciji i selekciji atributa koji su ključni za analizu, modeliranje ili rešavanje određenog problema, dok istovremeno omogućava isključivanje manje bitnih atributa. Određivanje mere relevantnosti atributa može uključivati analizu podataka, istraživanje domena problema, i konsultaciju sa stručnjacima. Različiti atributi mogu imati različitu važnost u različitim kontekstima, pa je važno uzeti u obzir specifičnosti problema ili sistema. Ova mera pomaže u optimizaciji resursa, poboljšava efikasnost analize podataka, i čini proces donošenja odluka ili projektovanja sistema usmerenijim i preciznijim.

Kada model ima nedovoljan broj atributa, postoji nekoliko značajnih posledica. Prvo, model će biti jednostavan za predstavljanje i analizu, što može biti korisno u određenim situacijama. Međutim, mala količina atributa takođe dovodi do male verodostojnosti modela, jer se ne obuhvataju svi relevantni aspekti sistema ili procesa. Ograničen je broj upotrebljivih informacija, što može otežati donošenje odluka i smanjiti vrednost baze podataka.

S druge strane, kada model ima previše atributa, verodostojnost modela se povećava jer obuhvata širok spektar informacija. Međutim, kompleksnost modela postaje velika, što može otežati manipulaciju podacima i analizu. Manipulacija podacima postaje teško izvodljiva, a informacije mogu postati konfuzne ako se ne vodi računa o relevantnosti svakog pojedinačnog atributa. Ovde je ključna ravnoteža između obuhvatanja svih relevantnih informacija i održavanja modela dovoljno jednostavnim za praktičnu primenu.

Prepoznavanje prave mere relevantnih atributa u modelovanju je od suštinskog značaja. To omogućava projektantima baza podataka da stvore efikasne i korisne modele, gde se obuhvataju bitni aspekti sistema bez nepotrebnog opterećenja informacijama koje nisu od suštinskog značaja. Ovo vodi do bolje upravljane bazom podataka, olakšava analizu podataka i donošenje odluka, i doprinosi ukupnoj efikasnosti informacionog sistema.

Prost/Složen atribut

Prost atribut predstavlja osnovnu jedinicu koja se ne može dalje rastavljati na delove bez gubitka svakog značenja. Ovaj tip atributa može se opisati samo jednim elementarnim (atomarnim) podatkom. Primeri prostih atributa uključuju merljive vrednosti kao što su visina (u centimetrima), ocena, smer i slično. Svaki od ovih atributa predstavlja pojedinačnu vrednost koja nije podložna daljem dekomponovanju.

Složen atribut, s druge strane, sastoji se od više prostih atributa i može se rastaviti na jednostavnije komponente. Ovi atributi predstavljaju višedimenzionalne informacije. Na primer, adresa može biti složen atribut koji se sastoji od pojedinačnih elemenata kao što su ulica, broj, mesto itd. Slično tome, datum rođenja se često sastoji od tri

prosta atributa: dan, mesec i godina. Ovaj tip atributa omogućava detaljnije opisivanje informacija koje nisu jednostavno merljive.

Primer: Prosti i složeni atributi

Prosti atributi: Ocena, Boja, Naziv

Složeni atributi: Adresa, ImeRadnika, DatumRođenja

Složeni atributi se mogu razložiti na manje proste atribute sa nezavisnim značenjem. Evo primera razlaganja složenih atributa na manje proste:

- Adresa: (Ulica, Broj, BrojStana, Sprat, Grad)
- ImeRadnika: (Ime, SrednjeSlovo, Prezime)
- DatumRođenja: (Dan, Mesec, Godina)

Postavlja se pitanje kada i zašto koristiti složene atribute u modelovanju baze podataka. Složeni atributi su korisni u situacijama gde korisnici ponekad pristupaju atributu kao celini, dok ponekad žele pristup pojedinačnim komponentama tog atributa. Ako se složeni atribut uvek koristi kao celina, preporučljivo je modelovati ga kao prost atribut.

Na primer, razmotrimo adresu studenta. Ako ne postoji potreba za nezavisnim referenciranjem pojedinačnih komponenti adrese (ulice, broja, mesta, itd.), tada bi adresu studenta trebalo modelovati kao prost atribut. Ovo olakšava rad s podacima i čini model jednostavnijim, posebno kada se cela adresa koristi kao jedna celina u većini situacija. Međutim, u situacijama gde je potrebno pristupiti pojedinačnim komponentama adrese, korisno je koristiti složene atribute radi preciznijeg modelovanja podataka.

Mandatorni /Opcionalni atribut

Mandatorni atribut je atribut čija vrednost uvek mora biti prisutna za svaki entitet u određenom skupu entiteta. Drugim rečima, entitet ne može postojati bez vrednosti za mandatorni atribut. Na primer, u većini sistema koji prate informacije o zaposlenima, ime zaposlenog je neophodno i ne može ostati nepopunjeno.

Opcionalni atribut, s druge strane, nije obavezan i može ostati bez vrednosti za određeni entitet. Opcionalni atributi su oni koji mogu ostati nepopunjeni, a entitet može postojati čak i ako nema vrednost za opcionalni atribut. U takvim situacijama se koristi specijalna vrednost - Null vrednost atributa. Praktično, atribut nema definisanu vrednost u dva slučaja: ako se radi o neodgovarajućem svojstvu za konkretan entitet, ili ako vrednost atributa postoji, ali je nepoznata.

Null vrednost može imati dva značenja:

- Postojeća ali nepoznata vrednost: Vrednost atributa za posmatrani entitet ne postoji ili još uvek nije poznata. Na primer, broj telefona često nije neophodan u određenim situacijama, kao što je slučaj u mobilnim ili bežičnim aplikacijama.

Datum venčanja za entitet OSOBA je opcionalan jer neke osobe nisu u braku. Datum diplomiranja za entitet STUDENT takođe može biti nepoznat za studente koji su tek upisali studije.

- Neodgovarajuća vrednost atributa: Vrednost atributa za posmatrani entitet nije primenjiva. Na primer, ako za entitet RADNIK imamo atribut u kome se čuva naziv fakulteta koji je radnik završio, svi radnici sa srednjom školskom spremom će imati *Null* vrednost za taj atribut.

Važno je razlikovati mandatorne i opcionalne atribute kako bi se precizno definisala struktura podataka i omogućilo prilagođavanje modela stvarnim potrebama sistema. Na primer, razmotrimo entitet STUDENT sa atributom "BrojTelefona". Ako je za potrebe konkretnog sistema broj telefona neophodan za svakog studenta, tada bi taj atribut bio mandatorni. E-mail adresa može biti mandatorni atribut za entitet ZAPOSLENI u modelu e-mail aplikacije, gde je ta informacija od suštinskog značaja. Takođe, svaka osoba mora imati datum rođenja, dok dodatni datumi kao što su datum venčanja ili datum diplomiranja mogu biti opcionalni atributi, jer nisu obavezni za svaku osobu. Razlikovanje mandatornih i opcionalnih atributa je ključno za precizno definisanje strukture podataka u bazi. Omogućava bolje upravljanje podacima i prilagođavanje modela stvarnim zahtevima sistema.

Primer: Značenje *Null* vrednosti

Slučaj neodgovarajuće vrednosti atributa:

- Ako zaposlena MILENA PETROVIĆ nema službeni telefon, atribut TELEFON za ovog zaposlenog će imati NULL vrednost.
- Ako zaposlena MILENA PETROVIĆ nije još uvek raspoređena ni u jedno odeljenje, atribut SIFRA ODELJENJA će imati NULL vrednost

Postojeća ali nepoznata vrednost:

- Ako je e-mail adresa zaposlene MILENE PETROVIĆ nije poznata, atribut MAIL će imati *NULL* vrednost.

Promenljivi/Nepromenljivi atribut

Promenljivi atributi, kao što su godine starosti, karakterišu se stalnim promenama u vrednostima tokom vremena. Sa druge strane, nepromenljivi atributi, poput datuma naručivanja ili datuma rođenja, retko ili gotovo nikada ne doživljavaju promene.

U situacijama kada se suočavate sa izborom između ovih atributa, preporučljivo je koristiti nepromenljive. Na primer, umesto korišćenja godina starosti, preporučuje se upotreba datuma rođenja. Ovo je zbog toga što godine starosti zahtevaju konstantno ažuriranje, dok datum rođenja ostaje nepromenjen i retko zahteva modifikacije. Korišćenje nepromenljivih atributa ima svoje prednosti, posebno u pojednostavljivanju upravljanja podacima i smanjenju potrebe za čestim ažuriranjem

vrednosti atributa. Ova praksa doprinosi stabilnosti i doslednosti podataka u bazi tokom vremena.

Jednovrednosni /Viševrednosni atribut

Jednovrednosni atributi omogućavaju entitetu da sadrži samo jednu vrednost. Sa druge strane, viševrednosni atributi omogućavaju entitetu da ima više vrednosti za isti atribut. Ova razlika ima značaj u strukturi podataka, gde jednovrednosni atributi pojednostavljuju organizaciju informacija, dok viševrednosni pružaju mogućnost bogatijeg opisivanja entiteta sa različitim vrednostima atributa. Na primer, datum rođenja je jednovrednosni atribut jer svaka osoba ima samo jedan datum rođenja. S druge strane, u entitetu STUDENT, atributi kao što su E-mail adresa i datum diplomiranja mogu biti viševrednosni, jer student može imati više mejl adresa i može diplomirati na različitim fakultetima ili nivoima studija.

Izvedeni atribut

Vrednost za ovu vrstu atributa može se izvesti iz vrednosti drugih povezanih atributa ili entiteta. Na primer, pretpostavimo da skup entiteta PROFESOR ima atribut StudentiDiplomci, koji predstavlja koliko studenata je radilo završni rad kod nekog profesora. Vrednost ovog atributa se može izvesti brojanjem broja entiteta STUDENT koji su diplomirali kod određenog profesora. Kao još jedan primer, pretpostavimo da skup entiteta PROFESOR ima atribut Starost koji pokazuje godine starosti profesora. Ako skup entiteta PROFESOR ima atribut DatumRođenja, možemo izračunati Starost na osnovu datuma rođenja i trenutnog datuma. Dakle, Starost je izvedeni atribut. U ovom slučaju, DatumRođenja se može nazivati osnovnim atributom ili atributom koji se čuva. Vrednost izvedenog atributa se ne čuva, već se izračunava kada je potrebna.

Domen atributa

Domen atributa predstavlja skup svih mogućih vrednosti koje određeni atribut može poprimiti. Domen je suženiji pojam u odnosu na tip podataka i označava se sa D_i. Domen atributa A_i, obeležava se sa $Dom(A_i)$ i sastoji se od pojedinačnih vrednosti koje atribut može imati:

$$Dom(A_i) = (Vrednost_1, Vrednost_2, \dots, Vrednost_n)$$

Na primer, atributi kao što su Ulica i Prezime spadaju u tekstualni tip podataka, ali imaju potpuno različite domene vrednosti. Naučno zvanje može biti tipa "text" i imati domen vrednosti {docent, vanredni profesor, redovni profesor}. S druge strane, atribut Poštanski broj je petocifren, ali ne znači da svaki petocifreni broj može biti njegova vrednost; određeni petocifreni brojevi čine validan domen za ovaj atribut. Važno je precizno definisati domene atributa kako bi se osigurala tačnost i relevantnost podataka u bazi.

Primer: Entitet STUDENT (BrojIndeksa, Ime, Prezime, Sp, Adresa)

Atribut sp je tekstualnog tipa i sadrži skraćenice za naziv studijskog programa. Za ovaj atribut se definiše odgovarajući domen vrednosti koji može poprimiti:

Dom (sp) = (Avt,Asuv,Elite,Is,Net,Nrt,Rt,Elin,Min,Rin)

Primer: Entitet OSOBA (Jmbg, Ime, Prezime, Grad)

Atribut grad je tekstualnog tipa i sadrži samo određene gradove. Za ovaj atribut se definiše odgovarajući domen vrednosti koji može poprimiti:

Dom (Grad) = (Beograd, Novi Sad, Niš, Čačak, Užice, Valjevo, Kruševac)

Najčešće korišćeni domeni u kontekstu baza podataka obuhvataju:

- Skup celih brojeva: Domen koji sadrži samo cele brojeve bez decimala, kao što su 1, -5, 100.
- Skup decimalnih brojeva: Domen koji obuhvata brojeve sa decimalama, kao što su 3.14, -0.5, 2.71828.
- String – niz alfanumeričkih karaktera: Domen koji obuhvata tekstualne vrednosti, kao što su "Hello, World!", "123abc", "SQL".
- Datum: Domen koji predstavlja vrednosti datuma, kao što su "2024-01-31", "15. mart 1990."
- Logički podatak: Domen koji obuhvata logičke vrednosti, obično True (istina) ili False (neistina).
- Novčani: Domen koji se koristi za vrednosti koje predstavljaju novčane iznose, na primer, 100.00 ili -25.50.
- Korisnički domeni: Domeni koji su specificirani prema potrebama korisnika i predstavljaju prilagođene skupove vrednosti, npr. status narudžbine (u obradi, isporučeno, otkazano).

Svi ovi domeni igraju ključnu ulogu u definisanju tipova podataka u bazi podataka i omogućavaju precizno upravljanje različitim vrstama informacija.

Ograničenja domena su prisutna u skoro svim modelima podataka, a opšti oblik ograničenja obuhvata tip podatka, dužinu podatka, uslov. Tip i dužina podatka predstavljaju standardna ograničenja domena. Tip podatka definiše vrstu znakova kojima se izražava vrednost atributa, dok dužina određuje maksimalni broj znakova koji se može koristiti za izražavanje vrednosti atributa. Često, ova ograničenja nisu dovoljno precizna, pa se dodaje treća komponenta – uslov, koja preciznije definiše ograničenje domena.

Standardni tipovi podataka sa određenim dužinama podataka uključuju:

- INTEGER (broj cifara)
- REAL (broj cifara ispred i iza decimalne tačke)
- CHARACTER (broj znakova)

- DATE
- LOGICAL...

Ova ograničenja su ključna za tačno definisanje i održavanje integriteta podataka u bazi podataka, pružajući jasne smernice o vrsti, dužini i dodatnim uslovima za svaki atribut u sistemu.

Jedinstveni identifikator (UID)

Jedinstveni identifikator (UID) je atribut u ER modelu koji ima jedinstvenu vrednost za svaki entitet unutar određenog skupa entiteta. Ovaj atribut se često koristi kako bi se omogućila jedinstvena identifikacija svakog entiteta unutar modela podataka.

Primenom jedinstvenog identifikatora, obezbeđuje se da svaki red u tabeli (entitetu) ima jedinstvenu vrednost za taj atribut. To olakšava upravljanje podacima, omogućava brzo pretraživanje i identifikaciju entiteta, i čini model podataka konzistentnijim. Potrebno je prepoznati atribute koja se mogu koristiti za identifikaciju entiteta unutar skupa. Minimalni skup atributa koji ima jedinstvenu vrednost za sve pojave određenog tipa entiteta predstavlja ključ odnosno jedinstveni identifikator (UID) tog tipa entiteta. Svaki tip entiteta ima barem jedan UID, a može imati i više ključeva kandidata. Jedan od tih ključeva kandidata se bira za primarni ključ u logičkom modelu baze podataka.

Na primer, za entitet STUDENT u ER modelu, jedinstveni identifikator može biti atribut BrojIndeksa, što znači da svaki student ima jedinstven broj indeksa koji ga identifikuje unutar sistema. Ovaj jedinstveni identifikator je od suštinskog značaja za obezbeđivanje preciznosti i doslednosti u radu sa podacima u bazi podataka.

Entitet ili Atribut

Ponekad može biti izazovno razlikovati entitete od atributa, posebno kada atribut postaje dovoljno složen i uključuje dodatne atribute. Ako atribut ima dovoljno informacija i složenosti da se posmatra kao samostalan entitet, može se tretirati kao novi entitet. Na primer, adresa može biti i entitet i atribut, u zavisnosti od konteksta:

Primer: Adresa kao entitet

Ako je potrebno voditi evidenciju o poslovanju prodavnica po gradovima ili broju stanovništva po gradovima, adresa može biti entitet sa svojim atributima.

ADRESA (Ulica, Broj, PoštanskiBroj, Grad)

Primer: Adresa kao atribut

Ako se adresa koristi kao deo informacija o zaposlenima u nekom preduzeću, tada bi adresa bila atribut entiteta RADNIK.

RADNIK (Ime, Prezime, Adresa, Kvalifikacija, DatumZaposlenja)

Primer: Ulica kao entitet

Ako se vode opsežni podaci o urbanizmu ili sektorima za urbanizam, ulica može biti samostalan entitet sa svojim atributima.

ULICA (Naziv, Dužina, Širina, VrstaKolovoza, GodinaIzgradnje)

Primer: Ulica kao atribut

Ako se podaci o zgradama vode na nivou sektora opštine, ulica može biti atribut entiteta ZGRADA.

ZGRADA (Ulica, KucniBroj, GodinaIzgradnje, BrSpratova, BrStanova)

Prednosti i mane definsanja određenog pojma kao atributa variraju u zavisnosti od potreba sistema. Ako se određeni pojam definiše kao atribut, može se postići višestruka upotreba istog koda za različite entitete, što može olakšati održavanje i organizaciju podataka. Međutim, može doći i do komplikacija u kodiranju, posebno ako su potrebne različite informacije u zavisnosti od konteksta. Ova praksa može dovesti do složenijeg koda za izdvajanje specifičnih informacija.

4.3 VEZE IZMEĐU ENTITETA

Realni sistem podrazumeva postojanje određenih odnosa među entitetima. U ER modelu, asocijacije unutar jednog skupa entiteta, kao i između entiteta u više skupova, predstavljaju **skup veza**. Skup veza je kolekcija veza istog tipa. Ovaj skup može uključivati relacije između entiteta koji se ne razlikuju po svojim vrednostima.

Ako su E_1, E_2, ..., E_n skupovi entiteta, tada je skup veza *R* podskup od:

$$\{(e_1, e_2, \dots, e_n) | e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

gde n - torka $(e_1, e_2, \dots, e_n)$ predstavlja vezu.

Veza između skupova entiteta naziva se **participacija**; odnosno, skupovi entiteta E_1, E_2, ..., E_n učestvuju (participiraju) u skupu veza R. Svaki entitet u n-torci ima svoju ulogu koja se može navesti na ER dijagramu. Ako se uloge entiteta eksplicitno navedu, redosled navođenja entiteta nije bitan. Između dva ista skupa entiteta može postojati više različitih skupova veza. Veza koja se ostvaruje između entiteta istog skupa naziva se rekurzivnom. Pojave skupa veza odnose se na konkretne veze između pojedinih entiteta odgovarajućih skupova entiteta koji učestvuju u vezi.

Primer*:* Posmatrajmo dva entiteta STUDENT i PREDMET

Može se definisati sledeći skup veza:

- Pohađa (STUDENT, PREDMET) - definiše vezu studenta sa predmetom
- Sluša (PREDMET, STUDENT) - definiše vezu predmeta sa studentom

Rekurzivna veza za entitet PREDMET:

- JePreduslov (PREDMET, PREDMET) – definiše odnos između predmeta koji je preduslov za slušanje nekog drugog predmeta

U realnom sistemu između ova dva entiteta postoje odnosi:

- "student pohađa predmet"
- "predmet sluša student"
- "predmet je preduslov predmetu"

Neka je Lazar elemenat skupa STUDENT, a Baze podataka, Programiranje elementi skupa PREDMET.

Veza (Lazar,Baze podataka) može imati značenje "Student Lazar pohađa Baze podataka".

Veza (Baze podataka,Lazar) može imati značenje "Predmet Baze podataka sluša Lazar".

Veza (Programiranje,Baze podataka) može imati značenje "Predmet Programiranje je preduslov predmetu Baze podataka".

Stepen skupa veza je broj tipova entiteta koji učestvuju u vezi. Skup veza **stepena jedan** naziva se **rekurzivni**, skup veza **stepena dva** se naziva **binarni**, skup veza **stepena tri** se naziva **ternarni**. Stepen jedan (rekurzivni) označava vrstu veze u kojoj je se asocijacija uspostavlja unutar istog entiteta. Binarni skup veza predstavlja vrstu veza koje određuju odnos između dva entiteta. Ternarni skup veza obuhvata tri entiteta i prikazuje kompleksne odnose između njih. U praksi se najčešće koriste binarni skup veza.

Primer: Skup veza stepena jedan - rekurzivni tip veze

- Veza radnika i njegovog nadređenog koji je takođe radnik (podaci o svim zaposlenima, bez obzira da li je rukovodilac ili ne, se čuvaju u jednoj tabeli):
  - JeRukovodilac(RADNIK,RADNIK)
- Veza između dve osobe koje su u braku (podaci o osobama bez obzira da li su u braku ili ne se čuvaju u jednoj tabeli):
  - UBraku(OSOBA,OSOBA)
- Veza između predmeta i preduslovnog predmeta (podaci o predmetima se čuvaju u jednoj tabeli; ne moraju svi predmeti da imaju preduslov):
  - JePreduslov(PREDMET,PREDMET)

Primer: Skup veza stepena dva – binarni tip veze

- Pohađa (STUDENT, PREDMET)
- Sluša (PREDMET, STUDENT)
- JeRaspoređen (RADNIK, ODELJENJE)

- Učestvuje (RADNIK,PROJEKAT)

Primer: Skup veza stepena tri – ternarni tip veze

- Predaje (PROFESOR, PREDMET, STUDENT): Profesor predaje predmet studentu.
- Ocenjuje (PROFESOR, STUDENT, PROJEKAT): Profesor daje ocenu studentu za projekat.
- RadiNaProjektu (RADNIK,PROJEKAT,ODELJENJE): Radnik radi na određenom projektu u određenom odeljenju.

4.3.1 ATRIBUTI SKUPA VEZA

Ponekad je korisno, čak i nužno, vezati atribute za odnose umesto za bilo koji od skupova entiteta koje taj odnos povezuje.Razmotrimo primer ternare veze *predaje* (PROFESOR, PREDMET, STUDENT) koja predstavlja predavanja koje profesor drži za određeni predmet studentima koji su ga izabrali da pohađaju. Možda želimo zabeležiti termin i mesto održavanja predavanja. Međutim, ne možemo ga povezati sa profesorom; profesor može da predaje različite predmete u različitim terminima i učionicima. Slično tome, nema smisla povezivati studenta sa terminim i mestom predavanja. Može pohađati samo predavanja iz jednog predmeta koji određeni profesor predaje ili može pohađati predavanja različitih predmeta kod različitih profesora. Rešenje je da termin i mesto održavanja predavanja predstavljaju atribute veza *predaje*. Veza će da ima atribute *termin* i *učionica*, a skupovi entiteta PROFESOR, PREDMET, STUDENT imaju iste atribute.

Uopšteno, možemo dodati jedan ili više atributa na bilo koju vezu. Vrednosti ovih atributa su funkcionalno određene celim n-torkama u skupu veza za taj odnos. U nekim slučajevima, atributi se mogu odrediti podskupom skupova entiteta uključenih u vezu, ali se pretpostavlja da se to ne može postići samo jednim skupom entiteta (ili bi bilo logičnije postaviti atribut na taj skup entiteta).

Nije uvek potrebno dodavati atribute na veze. Umesto toga, možemo kreirati novi skup entiteta, čiji entiteti imaju atribute dodeljene odnosu. Ako uključimo ovaj skup entiteta u odnos, možemo izostaviti atribute na samom odnosu. Međutim, atributi na odnosu su korisna konvencija, koju ćemo i dalje koristiti gde je to prikladno.

4.3.2 OGRANIČENJA SKUPA VEZA

Ograničenja nad skupom veza definišu pravila koja utiču na način kako se veze mogu uspostavljati ili kako se entiteti mogu povezivati u okviru ER modela podataka. Postavljaju pravila ponašanja, uslove, restrikcije kojima mora odgovarati skup veza. Ova ograničenja doprinose preciznijem modelovanju podataka i koriste se za održavanje doslednosti i integriteta podataka.

Primer: Ograničenja nad skupom veza koja važe za deo realnog sistema koji se odnosi na fakultet.

- Student može pohađati jedan ili više predmeta.
- Predmet može biti slušan od strane nijednog ili više studenata.
- Profesor može predavati jedan ili više predmeta.
- Predmet može biti dodeljen nijednom ili više profesora.
- Predavanja iz jednog predmeta moraju se održavati u jednom terminu i jednoj učionici.
- Student može polagati ispit u nijednom ili u više ispitnih rokova.

Osnovna ograničenja koja se definišu nad skupom veza su:

- Kardinalnost (brojnost) veze
- Participacija (učešće) veze

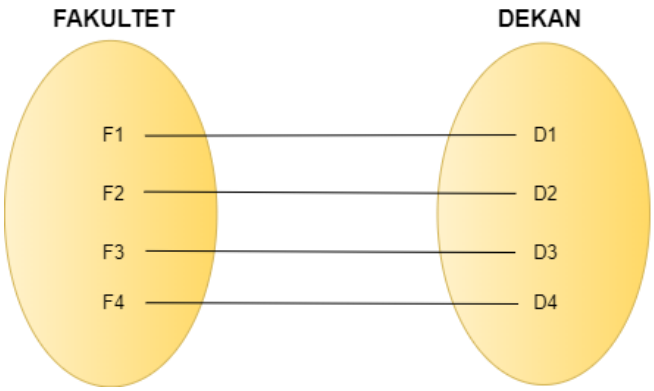
4.3.3 KARDINALNOST (BROJNOST) VEZE

Kardinalnost veze u modelovanju baza podataka se odnosi na broj entiteta koji mogu učestvovati u vezi. Ovo oblikuje odnos između entiteta u kontekstu određene veze. Kardinalnost se obično izražava kao minimum i maksimum entiteta koji mogu učestvovati u vezi.

Mogući odnosi kardinaliteta binarnog skupa veza R(E1,E2) su:

- jedan-prema-jedan (1 : 1)
- više-prema-jedan (N : 1)
- jedan-prema-više (1 : N)
- više-prema-više (N : M), N ≥0 i M≥0

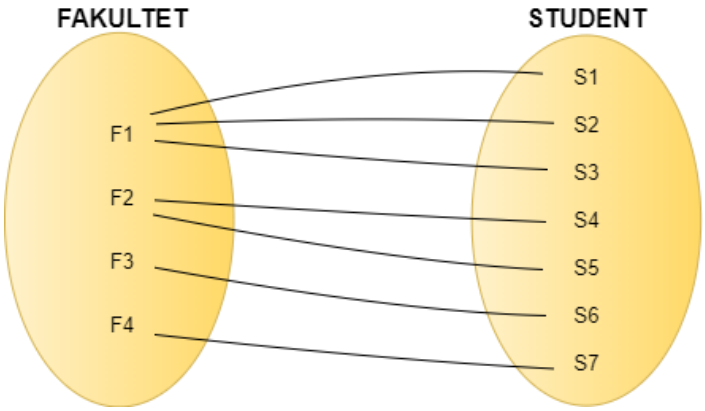
Jedan prema jedan (1:1). Veza 1:1 označava da svaki elemenat prvog skupa može biti povezan tačno sa jednim elementom u drugom skupu. Istovremeno svaki element drugog skupa može biti povezan sa najviše jednim elementom prvog skupa.



Slika 4.3.2.1 Ilustracija veze 1:1 entiteta FAKULTET i DEKAN

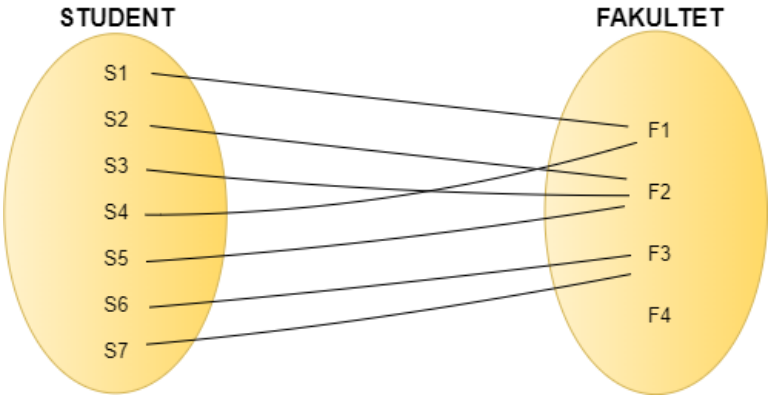
Na slici 4.3.2.1 je prikazan primer veze 1:1 između entiteta FAKULTET i DEKAN. Jedan fakultet ima tačno jednog dekana i jedan dekan je zadužen za tačno jedan fakultet.

Više-prema-jedan (N : 1). Veza N:1 označava da svaki element prvog skupa može biti povezan sa najviše jednim elementom drugog skupa. Istovremeno svaki element drugog skupa može biti povezan sa više elementa prvog skupa. Na slici 4.3.2.2 je prikazan primer veze N:1 između entiteta FAKULTET i STUDENT. Fakultet može da ima više studenata, a student pripada samo jednom fakultetu.



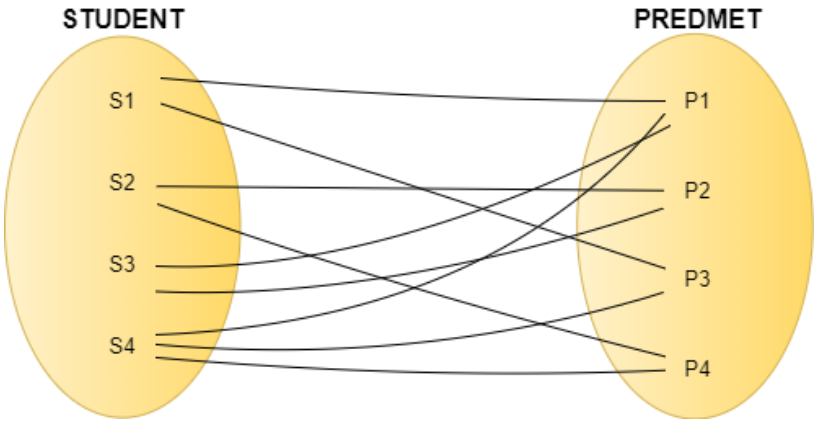
Slika 4.3.2.2 Ilustracija veze N:1 entiteta FAKULTET i STUDENT

Jedan prema više (1 : N). Veza 1:N označava da svaki element prvog skupa može biti povezan sa više elemenata drugog skupa. Istovremeno svaki element drugog skupa može biti povezan samo sa jednim elementom prvog skupa. Na slici 4.3.2.3 je prikazan primer veze 1:N između entiteta STUDENT i FAKULTET. Student pripada samo jednom fakultetu, a fakultet može imati više studenata.



Slika 4.3.2.3 Ilustracija veze 1:N entiteta STUDENT i FAKULTET

Više prema više (N : M). Veza N:M označava da svaki element prvog skupa entiteta može biti povezan sa više elemenata drugog skupa, i obrnuto, svaki elemenat drugog skupa može biti povezan sa više elemenata prvog skupa. Na slici 4.3.2.4 je prikazan primer veze N:M između entiteta STUDENT i PREDMET. Student može da pohađa više predmeta, a predmet sluša više studenata.



Slika 4.3.2.4 Ilustracija veze N:M entiteta STUDENT i PREDMET

4.3.4 PARTICIPACIJA (UČEŠĆE) VEZE

Participacija u vezi predstavlja određivanje obaveznosti ili opcionalnosti povezanosti između entiteta u skupu veza. Ovaj koncept definiše minimalni broj instanci sa kojima entitet može biti povezan, odnosno koliko je njegova povezanost sa drugim entitetima obavezna. Često se naziva i minimalnim ograničenjem kardinaliteta, s obzirom da specificira minimalan broj instanci koje entitet mora imati u vezi. U mnogim kontekstima, ovaj aspekt participacije se dodatno naziva i opcionalnost veze, jer određuje da li postojanje određenog entiteta zavisi od njegove povezanosti sa drugim entitetom u skupu veza. Participacija je važan aspekt u definisanju pravila veza između entiteta, pružajući informacije o tome da li je određena veza obavezna ili opciona za entitete koji učestvuju u njoj.

Postoje dva osnovna tipa participacije:

- **Obavezna participacija (Mandatorna veza).** U slučaju obavezne participacije, svaki entitet u skupu veza mora imati povezan entitet u drugom skupu, odnosno povezanost je obavezna. Ovaj tip participacije takođe se naziva i egzistencijalna participacija. Objašnjenje naziva sledi u nastavku. Neka je skup veza između entiteta E1 i E2 označen sa R(E1, E2). Ako svaki entitet iz skupa entiteta E1 mora biti povezan sa nekim entitetom iz skupa entiteta E2 preko veze iz skupa veza R, to znači da egzistencija (postojanje) nekog entiteta u E1 zavisi od toga da li je povezan sa nekim entitetom iz E2.
- **Opciona participacija (Opcionalna veza).** Ovaj tip učešća u vezi omogućava da neki entiteti u skupu veza nemaju povezane entitete u drugom skupu.

Posmatrajmo opet skup veza R(E_1, E_2). Opciona participacija omogućava da ne mora svaki entitet iz skupa E_1 da bude povezan sa nekim entitetom iz skupa E_2 preko R. Dodatno se može zaključiti da u slučaju ovog tipa participacije u skupu entiteta E_1 mogu postojati entiteti koji nisu povezani ni sa jednim entitetom iz skupa entiteta E_2 preko skupa veza R.

Primer: Obavezna participacija

Ako svaki entitet u skupu A mora biti povezan sa tačno jednim entitetom u skupu B, i obrnuto, tada je to obavezna participacija. Na primer, svaki profesor mora imati , a svaka "Kancelarija" mora pripadati jednom "Profesoru".

Primer: Opciona participacija

Ako svaki entitet u skupu A može biti povezan sa više entiteta u skupu B, ali ne mora biti povezan sa nijednim, tada je to opciona participacija. Na primer, predmet može pohađati mnogo studenata, ali neki studenti ne moraju izabrati da slušaj neki predmet.

4.3.5 SLABI TIP ENTIETA

Slabi tip entiteta predstavlja poseban tip entiteta koji nema sopstvene ključne atribute u tradicionalnom smislu, kako smo su do sada definisani i korišćeni. Identifikacija entiteta slabog tipa vrši se putem veze sa drugim tipom entiteta, bilo regularnim ili slabim, zajedno sa dodatnim atributom. Ovaj drugi tip entiteta naziva se **identifikujući tip entiteta** (vlasnički, roditeljski, jak ili dominantni). Veza između slabog tipa entiteta i jakog naziva se **identifikujuća veza**. Bitno je napomenuti da slab tip entiteta uvek potpuno participira u identifikujućem tipu veze. Slabi tip entiteta karakteriše **parcijalni ključ**, poznat i kao **diskriminator**. Parcijalni ključ predstavlja podskup atributa slabog tipa entiteta koji na jedinstven način identifikuje pojedinačne entitete u vezi sa istim vlasničkim tipom entiteta. U najgorem slučaju, kada nije moguće izdvojiti određeni podskup atributa, parcijalni ključ obuhvata sve atribute slabog tipa entiteta.

Primer: Posmatrajmo slučaj modela sistema prodavnice u kome se beleže podaci o dobavljačima bez obzira da li dobavljaju neke artikle, a podaci o artiklima se beleže samo ako postoji podatak o njegovom dobavljaču.

U slučaju posmatranog sistema, dobavljač (DOBAVLJAC) se smatra jakim i dominantnim entitetom jer podaci o dobavljačima mogu postojati nezavisno od veze sa određenim artiklom. Sa druge strane, artikal (ARTIKAL) je slab entitet, jer njegovo postojanje zavisi od veze sa dobavljačem (DOBAVLJAC). Slabi entitet nema sopstveni ključ, već se identifikuje putem veze sa jakim entitetom, u ovom slučaju, sa dobavljačem.

Primer: Posmatrajmo slučaj modela sistema banke u kome se beleže podaci o klinetima bez obzira da li su izvršili neku transakciju, a podaci o transakcijama se beleže samo kada je klijent realizuje.

U slučaju posmatranog sistema, klijent (KLIJENT) se smatra jakim i dominantnim entitetom jer podaci o klijentima mogu postojati nezavisno od veze sa određenom transakcijom. Sa druge strane, transakcija (TRANSAKCIJA) je slab entitet, jer njegovo postojanje zavisi od veze sa klijentom (KLIJENT). Slabi entitet TRANSAKCIJA nema sopstveni jedinstven identifikator (ključ), već se identifikuje putem veze sa klijentom.

4.3.6 GRAFIČKA NOTACIJA ER DIJAGRAMA

Šema konceptualnog ER modela podataka predstavlja se dijagramom. ER dijagram je grafički prikaz modela podataka koji koristi simbole za entitete, veze i atribute, i na taj način ilustruje strukturu podataka i odnose između njih. Pomaže u vizualizaciji i razumevanju procesa funkcionisanja sistema koji se modeluje.

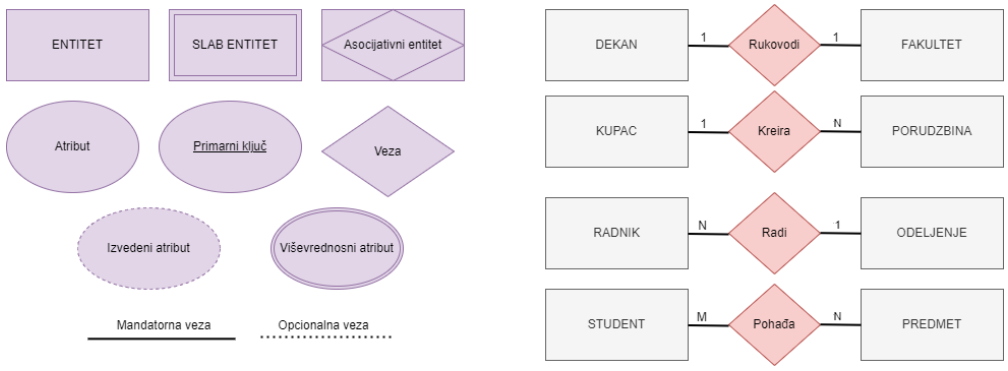
Pravila koja su korišćena u prethodnom tekstu, koristiće se i dalje za obeležavanje entiteta, atributa, veza i ključeva u kreiranju ER dijagrama.

- Entiteti:
  - Imena entiteta trebaju biti imenice u jednini.
  - Entitete obeležavamo velikim slovima.
  - STUDENT, PREDMET, PROFESOR, ISPIT, ROK.
- Atributi i veze:
  - Atributi i veze se mogu obeležavati na dva načina:
    - Nizom reči koje se pišu velikim slovima sa znakom za podvlačenje ili crticom između reči.
      - IME_STUDENTA, NAZIV_PREDMETA, BROJ_INDEKSA ili IME-STUDENTA, NAZIV-PREDMETA, BROJ-INDEKSA.
    - Nizom reči koje se pišu malim slovima, osim prvog slova svake reči, bez crtice, znaka za podvlačenje ili bilo kog drugog delimetera između reči.
      - ImeStudenta, NazivPredmeta, BrojIndeksa.
  - Atributi i nazivi veza su imenice u jednini, a uloge u vezama glagoli.
- Domen atributa:
  - Domen atributa A obeležavamo sa Dom(A).
- Tip entiteta:
  - Tip entiteta E sa atributima A_1, A_2, ..., A_n označavaćemo kao E(A_1, A_2, ..., A_n).
- Ključevi:
  - Atributi koji čine primarni ključ entiteta biće podvučeni.
  - Atributi koji čine strani ključ entiteta biće italic.

Primenom ovih pravila, modelovanje postaje jasno i dosledno, što olakšava razumevanje ER dijagrama.

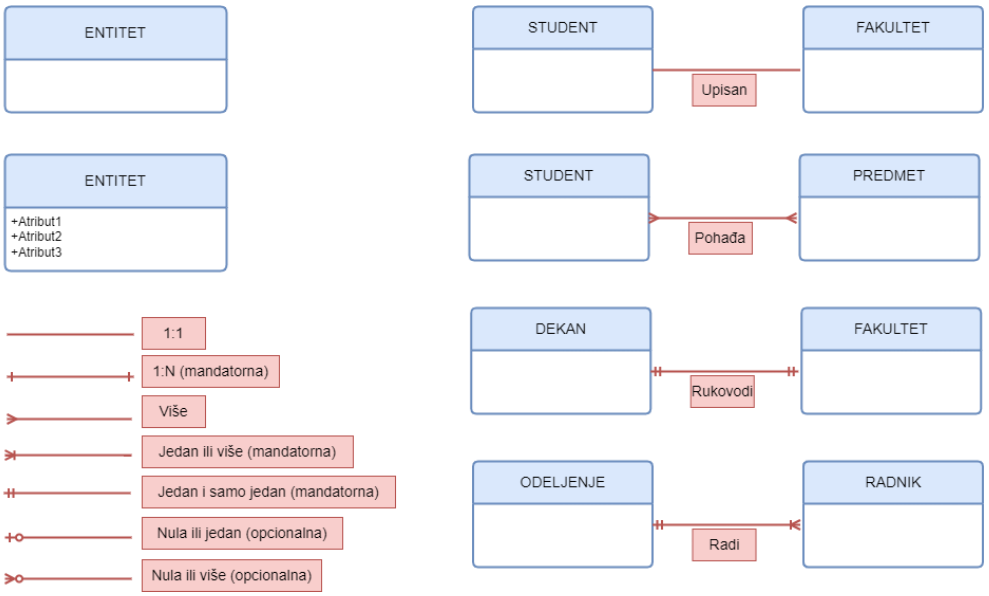
ERD (engl.*Entity-Relationship Diagram*) simboli nekoliko najčešće korišćenih stilova notacija obuhvataju osnovne elemente koji se koriste u ER dijagramima.

Na slici 4.3.6.1 su prikazani osnovni simboli za kreiranje ER dijagrama primenom Chen notacije.



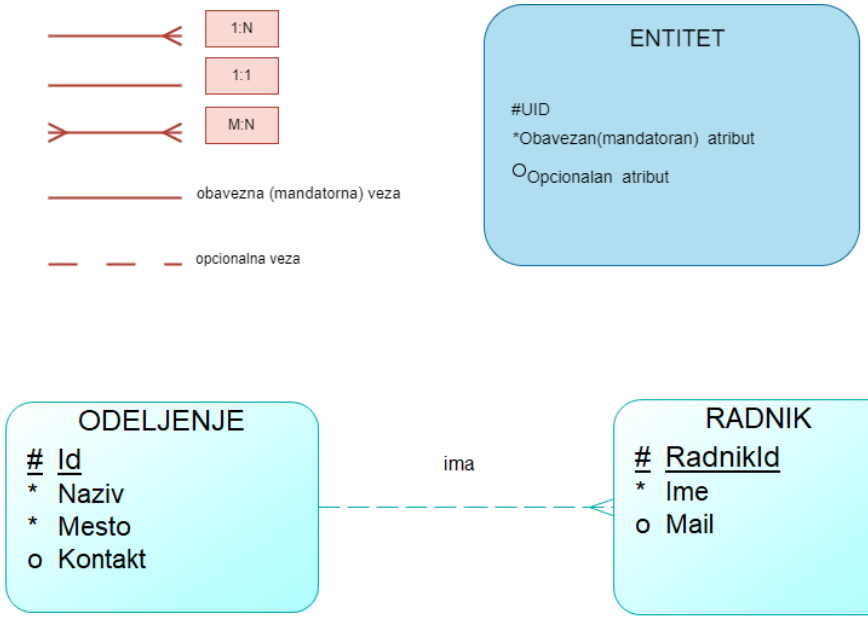
Slika 4.3.6.1 Chen notacija

Na slici 4.3.6.2 su prikazani osnovni simboli za kreiranje ER dijagrama primenom Crow's Foot notacije.



Slika 4.3.6.2 Crow's Foot notacija

Na slici 4.3.6.3 su prikazani osnovni simboli za kreiranje ER dijagrama primenom Barker's notacije



Slika 4.3.6.3 Barker notacija

4.3.7 PRIMER MODELOVANJA I PROJEKTOVANJA BAZE PODATAKA

Primer: Na osnovu zahteva projektovati ER dijagrame baze podataka MUZEJ UMETNOSTI.

Zahtevi korisničkog scenarija funkcionisanja realnog sistema:

- Čuvaju se podaci o slikarima, slikama, muzejima u kojima se nalaze njihove slike.
- Za slikare se pamti ime, nacionalnost, datum rođenja i datum smrti (ako je poznat).
- Za muzeje se pamti lokacija, kao i specijalnost, ako postoji.
- Za slike se čuvaju podaci o slikaru, dimenzijama (visina i širina u centimetrima), godini kada je nastala, naslov i stil kome pripada.
- Za svaki stil se beleži naziv stila i opis istog.
- Jedan slikar može da naslika više slika, a jednu sliku je naslikao samo jedan slikar.
- Jedna slika mora da pripada samo jednom stilu, a više slika može biti istog stila.
- Svakoj slici mora da bude pridružen podatak o stilu kome pripada.
- Mogu biti zabeleženi stilovi za koje nije trenutno evdentirana ni jedna slika.

- Slike mogu da budu izložene u različitim muzejima u različitim terminima izložbi.
- Muzeji mogu da organizuju više različitih izložbi.

U procesu konceptualnog modelovanja baze podataka prvo treba prepoznati tipove entiteta. U ovom slučaju to su: slikari, slike, stilovi, muzeji. Ime slikara može da bude složen atribut. Defininje skupova veza zasniva se na analizi postavljenih zahteva u korisničkom scenariju u kome je opisan način funkcionisanja realnog sistema koji se modeluje.

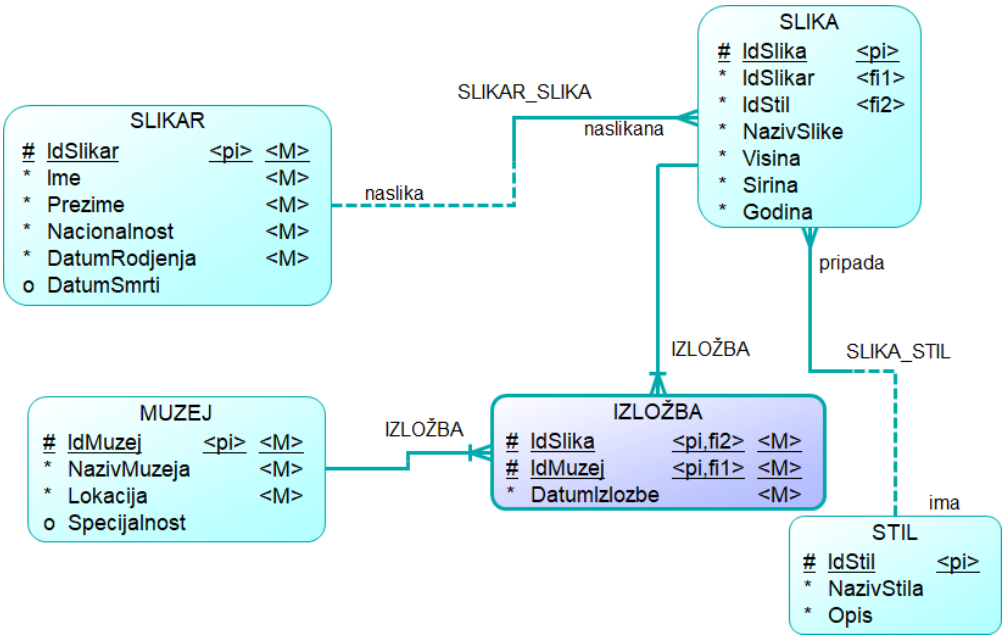
Veza između dva tipa entitea se identifikuje na osnovu zahteva koji ukazuju da vrednost koja treba da se čuva u nekom tipu entiteta predstavlja vrednost atributa drugog tipa entitata. U zahtevima stoji da sliku naslika jedan slikar, a slikar može da naslika više slika što znači da se u tipu entiteta slike čuva i podatak o slikaru koji je naslikao. Za slike se čuva i podatak o stilu što ukazuje na vezu sa tipom entiteta u kojima se beleže podaci o stilovima u slikarstvu.

Kod veza muzeja i slike, gde se definiše u kom muzeju je izložena određena slika, treba čuvati informaciju o datumu postavke izložbe. Vrednost datuma izložbe se razlikuje za svaku pojedinačnu vezu u tom skupu veza i može se posmatrati kao atribut veze.

Potrebno je odrediti i ograničenja za svaki skup veze. Veza slike i stila je 1:N (slika pripada jednom stilu, a jedan stil sadrži više slika), gde ne moraju svi stilovi da imaju dodeljene slike (parcijalno učešće stilova), a svaka slika mora da ima stil (totalno učešće slika).

Odnos slika i muzeja na osnovu zahteva definiše kardinalitet M:N. Muzeji realizuju više izložbi, a slike mogu da budu izložene u više muzeja u različitim terminima izložbi. Participacija slika i muzeja u vezi je delimična zato što je u zahtevima navedeno da slike mogu da budu izložene u različitim muzejima u različitim terminima izložbi, a da muzeji mogu da organizuju više različitih izložbi.

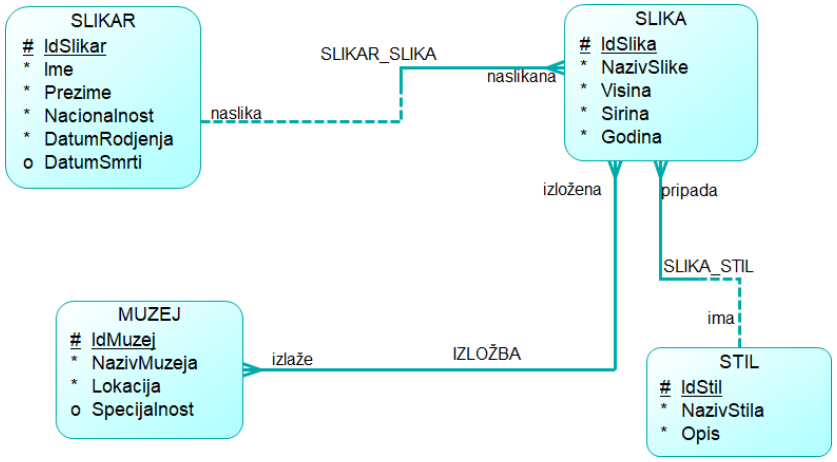
ER dijagram konceptualnog modela baze podataka MUZEJ_IZLOZBA je prikazan na slici 4.3.7.1. Za kreiranje dijagrama je korišćena Barker`s notacija.



Slika 4.3.7.1 Konceptualni model baze podataka MUZEJ_IZLOZBA

Na osnovu konceptualnog modela kreiran je logički model u okviru koga se prikazuje realizacija veze M:N između entiteta MUZEJ i entitea SLIKA.

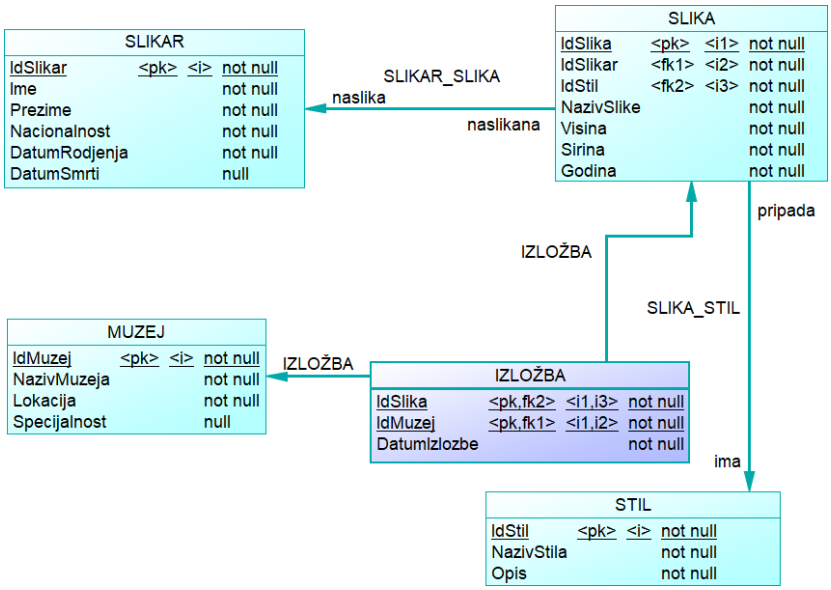
Na slici 4.3.7.2 je prikazan ER dijagram logičkog modela sa simbolima Barker`s notacije. Može primetiti da realizacija veze M:N, u logičkom modelu, podrazumeva kreiranje veznog entieta.



Slika 4.3.7.2 Logički model baze podataka MUZEJ_IZLOZBA

Zahtevi koji se odnose na identifikovanje autora i stila svake slike (veze SLIKAR_SLIKA, SLIKA_STIL), u logičkom modelu, se označavaju stranim ključevima idSlikar i IdStil u entitetu SLIKA.

Na osnovu logičkog model generiše se fizički model sa postavljanjem karakteristika koje se odnose na izabrani DBMS (slika 4.3.7.3).



Slika 4.3.7.3 Fizički model baze podataka MUZEJ_IZLOZBA

Primer: Na osnovu zahteva projektovati ER dijagrame baze podataka u kojoj se čuvaju podaci o studentima, predmetima koje su izabrali da pohađaju, profesorima, ispitima, ispitnom rokovima i ostvarenim ocenama. Projektovana baza podataka predstavlja deo informacionog sistema nekog fakulteta.

Zahtevi korisničkog scenarija funkcionisanja realnog sistema:

- Za studente se beleže lični podaci, broj indeksa, kontakt telefon.
- Za predmete se čuvaju podaci o nazivu, broju ESPB bodova, da li ima preduslov, podatak o nastavniku kome je dodeljen predmet.
- Za profesore se beleže lični podaci, titula, datumi izbora u zvanje, broj kabineta, kontakt mejl.
- Svaki profesor ima svoj kabinet, a u jednom kabinetu može da sedi jedan ili više profesora.
- Za svaki kabinet beleži se podatak o oznaci kabineta i maksimalnom broju nastavnika koji mogu da "sede"
- Profesor može da drži jedan ili više predmeta, a predmet može da drži jedan ili više profesora.
- Za ispitne rokove se čuvaju podaci o nazivu roka i školskoj godini.
- Za ispite se beleže podaci o predmetu, ispitnom roku, terminu i sali gde se održava ispit.
- Za određen ispit definiše se tačno jedan termin i mesto.
- Za svaku salu čuvaju se podaci o oznaci sale i maksimalnom broju mesta.
- Zapisnik sadrži podatke o ispitu, studentu i oceni koju je ostvario.
- Student na ispitu može da ostvari ocene 5, 6, 7, 8, 9,10.
- Ako student nije polagao ispit, a isti je prijavio upisuje se vrednost 3.
- Studenti mogu da izaberu jedan ili više predmeta da slušaju.
- Predmet može da izabere nula, jedan ili više studenata.
- Studenti mogu da prijave nula, jedan ili više ispita da polažu u jednom ispitnom roku.
- Zakazan ispit može da nema prijavljenih studenata.

Proces modelovanja i projektovanja baze podataka započinjemo fazom kreiranja konceptualnog modela. Na osnovu postavljenih zahteva identifikuju se tipovi entiteta: studenti, predmeti, profesori, kabineti, izbori u zvanja, ispiti, ispitni rokovi, sale, školske godine. Lični podaci studenta i profesora mogu da budu složeni atributi. Uzimajući u obzir da broj indeksa sadrži skraćenicu studijskog programa, upisni broj i godinu upisa, takođe se smatra složenim atributom.

Postojanje preduslova za neki predmet ukazuje na potrebu za postavljanje rekurzivne veze na tip entiteta PREDMET. To znači da se u entitetu PREDMET pamti i podatak o preduslovu za neki predmet. Kako je u zahtevima navedeno da profesor ima samo jedan kabinet, a u jednom kabinetu može da „sedi" više profesora znači da se u tipu entiteta profesor čuva i podatak o kabinetu. Za profesore se pamte svi datumi izbora

u isto ili različito zvanje što ukazuje na vezu između tipa entiteta profesori i zvanja. Kod ove veze potrebno je pamtititi i datum izbora u zvanje koji predstavlja atribut ostvarene veze.

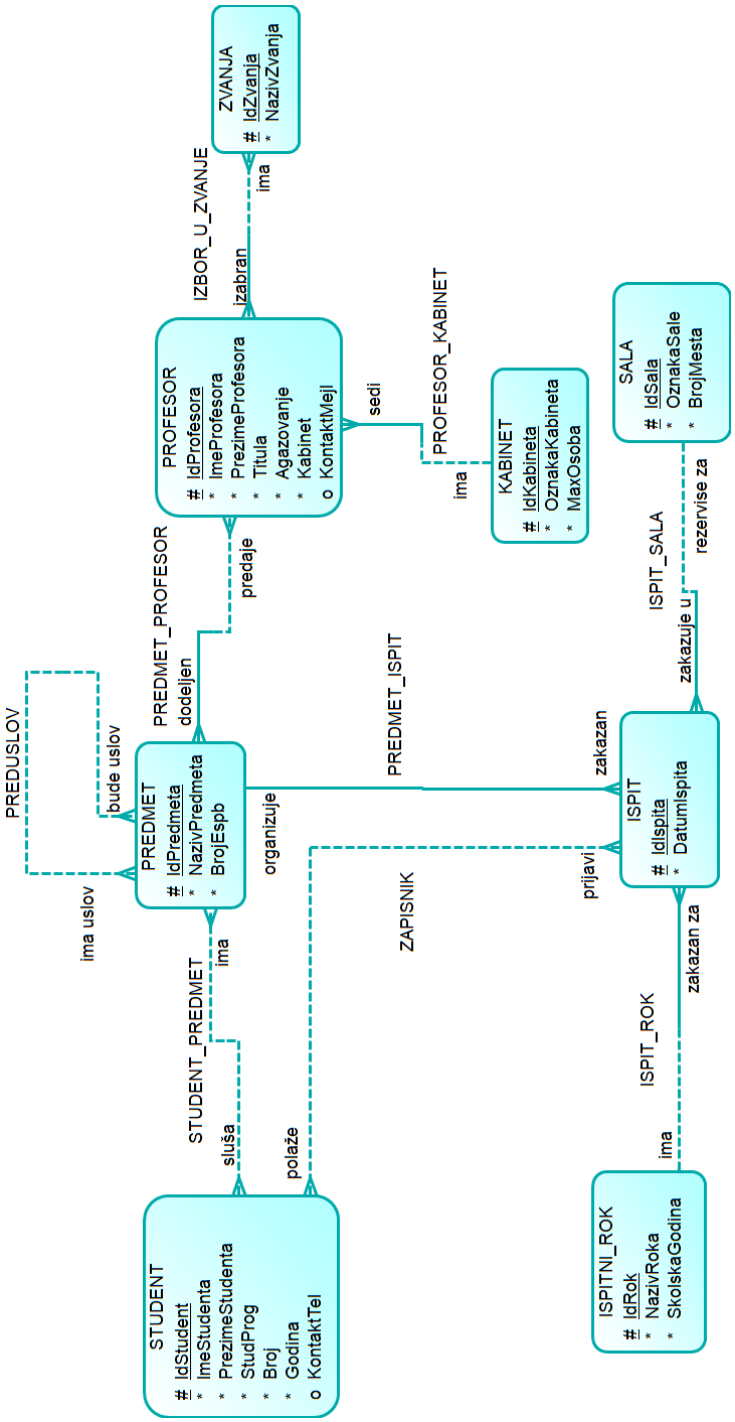
Za ispite se čuva podatak o ispitnom roku, terminu, mestu održavanja što ukazuje na vezu sa tipovima entiteta u kojima se beleže podaci o ispitnim rokovima i salama. Kod veze studenta i predmeta, gde se definiše koji student je izabrao koji predmet, treba čuvati i podatak o godini studija. Vrednost godine studija može da se razlikuje za pojedinačne veze u skupu koji se odnosi na različite godine pa se može posmatrati kao atribut veze.

Uspostavljanjem veze između tipova entiteta STUDENT i ISPIT biće obezbeđeno generisanje tipova entiteta ZAPISNIK u okviru koga je potrebno čuvati i podataka o oceni što će predstavljati atribut veze. Na osnovu veze između tipova entiteta PROFESOR i PREDMET obezbeđuje se čuvanje podataka koji profesor je zadužen za koji predmet.

Potrebno je odrediti i ograničenja za svaki skup veze. Veza profesora i kabineta je 1:N (profesor „sedi“ u jednom kabinetu, a u jednom kabinetu može da „sedi“ više profesora), gde ne moraju svi kabineti da imaju dodeljene profesore (parcijalno učešće kabineta), a svaki profesor mora da ima dodeljen kabinet (totalno učešće profesora). Postavljeni zahtevi za odnose studenta i predmeta, profesora i predmeta, studenata i ispita definišu kardinalitet M:N. Participacija navedenih entiteta u vezama je delimična zato što je u zahtevima navedeno opcionalno učešće u vezama.

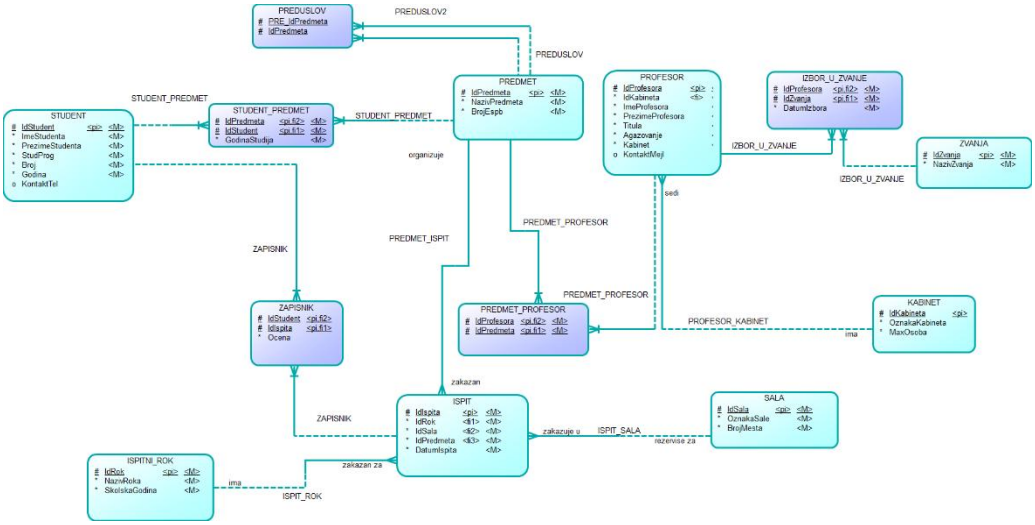
ER dijagram konceptualnog modela baze podataka FAKULTET je prikazan na slici 4.3.7.4. Za kreiranje dijagrama je korišćena Barker`s notacija.

Na osnovu konceptualnog modela kreiran je logički model. Realizacija veza M:N između entiteta podrzumevala je kreiranje odgovarajućih veznih entiteta. Za neke vezne entitete definisani su odgovarajući atributi veze.



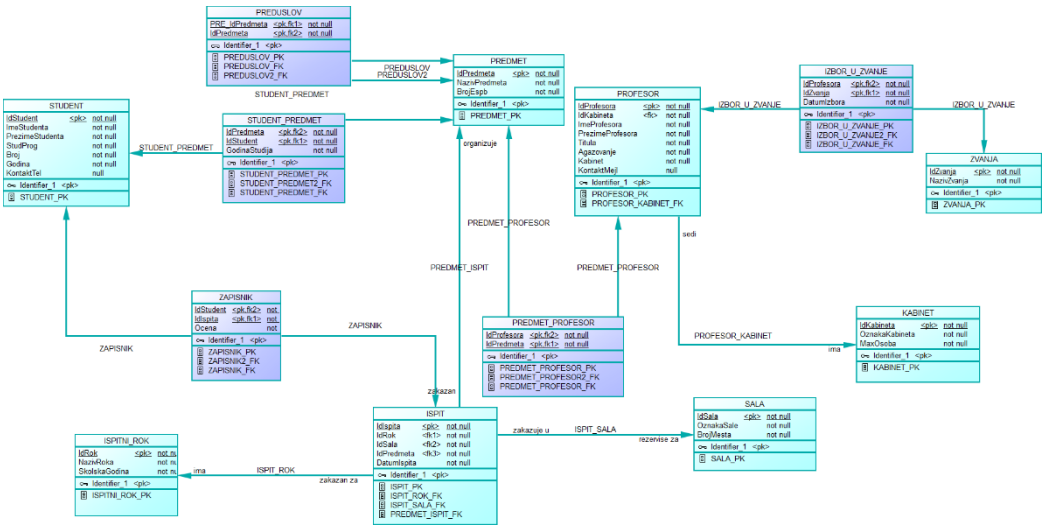
Slika 4.3.7.4 Konceptualni model baze podataka FAKULTET

Na slici 4.3.7.5 je prikazan ER dijagram logičkog modela sa simbolima Barker`s notacije.



Slika 4.3.7.5 Logički model baze podataka FAKULTET

Na osnovu logičkog model generisan je odgovarajući fizički model sa postavljanjem karakteristika koje se odnose na izabrani DBMS (slika 4.3.7.6).



Slika 4.3.7.6 Fizički model baze podataka FAKULTET

4.4 PITANJA I ZADACI

1. Objasniti proces modelovanja baze podataka.
2. Šta predstavlja konceptualni model?
3. Navesti faze projektovanja baze podataka.
4. Objasniti pojam modela entiteta i veza.
5. Šta predstavlja entitet?
6. Objasniti pojam atributa entiteta.
7. U kojim situacijama je atribut treba da postane poseban entitet?
8. Koja se ograničenja definišu nad skupom veza?
9. Šta predstavlja slab tip entiteta?
10. Na osnovu zahteva projektovati ER dijagrame baze podataka u kojoj se čuvaju podaci o pacijentima, pregledima i lekarima. Projektovana baza podataka predstavlja deo informacionog sistema ordinacije. Zahtevi korisničkog scenarija funkcionisanja realnog sistema:
    - Za pacijente se beleže lični podaci, ime, prezime, datum rođenja, kontakt telefon.
    - Za lekare se beleže lični podaci, oblast specijalizacije, kontakt mejl.
    - Cena pregleda zavisi od usluge koju je izabrao pacijent.
    - Za svaku uslugu se beleži naziv (npr. vađenje krvi, ultrazvuk, specijalistički pregled i slično), cena i oblast (npr.laboratorija, pedijatrija,...).
    - Za svaku oblast specijalizacije beleži se naziv (npr. pedijatrija, fizijatrija i reumatologija,...).
    - Lekar je specijaliziran za jednu oblast, a u jednoj oblasti može da radi jedan ili više lekara.
    - Pacijent može da obavi više pregleda kod različitih lekara.
    - Lekar može da pregleda više pacijenata.
    - Za svaki pregled se beleže podaci o pacijentu, lekaru koji je obavio bregled, vrsti usluge i datumu pregleda. Cena pregleda zavisi od usluge koju je pacijent odabrao.

5 NORMALIZACIJA

Cilj ovog poglavlja je fokusiran na detaljno objašnjenje procesa normalizacije relacionog modela podataka. Analizirani su problemi koji proizlaze iz redundantnosti podataka i anomalija uzrokovanih slabo organizovanom strukturom relacija. Definisani su i razjašnjeni koncepti funkcionalnih zavisnosti, kao i različite vrste istih. Takođe, analizirane su formalne metode poznate kao normalne forme koje se koriste u procesu normalizacije. Detaljno je obrađena prva, druga, treća i Boyce-Coddovu normalna forma. Razmotren je značaj normalizacije, kao i osnovne karakteristike i procesa denormalizacije relacionog modela podataka.

5.1 UVOD

Postupak pravilnog modelovanja smatra se najzahtevnijom fazom u kreiranju relacionih baza podataka. Osnovni cilj modelovanja jeste identifikacija objekata, njihovih međusobnih veza i ograničenja koje treba uzeti u obzir iz stvarnog sveta. Kreiranje optimalnog konceptualnog modela, zasnovanog na zahtevima korisničkih scenarija, omogućava definisanje odgovarajućih šema relacija u relacionom modelu podataka za određeni sistem.

Ključni aspekt pravilnog dizajna baze podataka jeste osiguranje da kreirane šeme relacija imaju optimalnu strukturu. To znači da prilikom rada sa bazom podataka, bilo da je u pitanju unos, izmena ili brisanje podataka, ne dolazi do neočekivanih problema ili anomalija. Loše dizajnirane šeme mogu rezultirati lošijim performansama sistema, greškama i neočekivanim rezultatima. Ispunjavanjem teorijskih koncepata relacionog modela podataka, kao što su jedinstvenost atributa i *n*-torki, identifikacioni integritet primarnih ključeva, kao i referencijalni integritet stranih ključeva, postiže se definisanje efikasnih i optimalno dobrih šema relacija.

Važno je naglasiti da ne postoje specifični zahtevi za ostale atribute u šemi, što može dovesti do pogrešnih zaključaka da ti atributi mogu imati proizvoljne vrednosti. Ukoliko se u šemama relacija pojave dodatne zavisnosti koje narušavaju osnovne principe relacionog modela podataka, kao što su zavisnosti između neključnih atributa ili između delova primarnog ključa i ostalih atributa, dolazi do pojave relacija loše strukture. Relacije loše strukture mogu prozrokovati probleme koji dovode do neispravnih rezultata ili gubitka podataka.

Da bi se takvi problemi rešili, primenjujuju se teorijski koncepti za identifikovanje funkcionalnih zavisnosti i sprovođenje procesa normalizacije. Normalizacija je postupak kojim se šeme relacija loše strukture rastavljaju na dve ili više šema uz zadovoljavanje uslova određene normalne forme. Ovaj proces omogućava

izbegavanje anomalija u radu sa bazom podataka i dobijanje tačnih informacija kroz upite.

Prvobitno su predstavljene tri normalne forme: prva (1NF), druga (2NF) i treća (3NF) normalna forma. Kasnije su Raymond Boyce i Edgar F. Codd razvili strožiju definiciju treće normalne forme poznatu kao Boyce-Codd normalna forma (BCNF). Pored toga, postoje i više normalnih formi koje prevazilaze BCNF, kao što su četvrta (4NF) i peta (5NF) normalna forma. Međutim, ove forme se bave situacijama koje su retke u praksi.

Normalizacija je proces transformacije proizvoljne nenormalizovane relacije u skup manjih, normalizovanih relacija. Sprovođenje procesa normalizacije podrazumeva sprečavanje gubljenja podataka koji su zabeleženi u originalnoj relaciji. Bitno je osigurati da se originalna polazna relacija može rekonstruisati na osnovu rezultata normalizacije. U procesu kreiranja tabela u relacionom modelu podataka, ključna je samo prva normalna forma (1NF). Sve ostale normalne forme su opcione. Ipak, kako bi se izbegle anomalije ažuriranja, preporučuje se normalizacija bar do treće normalne forme (3NF).

Implementacijom teorijskih koncepata relacione algebre, normalizacija se može posmatrati kao proces dekompozicije šeme $R(A_1, A_2, \ldots, A_n)$ relacije r. Nove šeme R_i sadrže podskupove atributa iz R, a njihova unija predstavlja osnovnu šemu R. Rezultujuće relacije r_1,r_2, ...,r_k nastaju projekcijama relacije r na podskupove atributa R_1,R_2, ...,R_k. Prirodnim spajanjem rezultujućih novih relacija mora se dobiti originalna relacija r.

Normalizacija omogućava optimizaciju strukture baze podataka kako bi ona bila efikasna za različite vrste upita nad podacima. Smanjuje redudantnost podataka i sprečava pojavu problema prilikom unosa, izmene i brisanja podataka. Ovaj proces uključuje kontinuiranu reorganizaciju baze radi eliminisanja atributa koji sadrže višestruke vrednosti ili duplikate. Iako potpuna normalizacija nije obavezna, preporučuje se radi smanjenja mogućnosti problema. Normalizacija se zasniva na hijerarhiji normalnih formi, pri čemu svaka sledeća forma rešava specifične probleme koji nisu obuhvaćeni prethodnom. Prve tri normalne forme smatraju se osnovnim pravilima za uspostavljanje relacionog modela. Povećanje nivoa normalizacije može otežati pristup specifičnim podacima i povećati korišćenje resursa prilikom izvršavanja upita.

5.2 REDUNDANSA I ANOMALIJE

Redundantni podaci odnose se na ponavljanje istih ili sličnih podataka u različitim delovima baze podataka. Oni mogu prouzrokovati niz problema, uključujući gubitak efikasnosti, povećanu složenost baze podataka i teškoće u održavanju podataka. Inkonzistentnost podataka, povećanje veličine baze podataka i sporiji upiti su neki od problema koje redundansa može izazvati. Takođe, redundantni podaci zauzimaju

prostor na disku i uzrokuju značajne poteškoće u održavanju sistema. Na primer, ukoliko se podaci koji postoje na više mesta moraju menjati, ti podaci moraju biti promenjeni na svim tim lokacijama na identičan način. Kao posledica redundanse, često se susrećemo sa problemima poznatim kao višestruko unošenje, višestruko menjanje i višestruko uklanjanje. Ovi problemi predstavljaju anomalije koje se javljaju u podacima zbog loše osmišljene strukture baze podataka.

5.2.1 ANOMALIJE

Anomalije u bazama podataka predstavljaju probleme koji se javljaju kada se izvršavaju operacije poput dodavanja, ažuriranja ili brisanja podataka, što rezultira neregularnim ili neočekivanim rezultatima. Ove anomalije su često posledica loše dizajnirane strukture baze podataka. U nastavku teksta biće objašnjene tri osnovne vrste anomalija.

Anomalije pri dodavanju podataka. Javljaju se prilikom dodavanja novih podataka u relaciju sa loše osmišljenom strukturom, a što može da prouzrokuje višestruko unošenje istih podataka za različite redove zapisa. Na primer, posmatrajmo sistem za vođenje podataka o zaposlenima. Neka je relacija odnosno tabela *zaposleni* loše osmišljena i pored neophodnih podataka o zaposlenom beleži i podatke o odeljenju u koje je zaposleni raspoređen. Svaki unos podataka o novom zaposlenom uključivao bi i unos podataka o odeljenju koji bi se ponavljali. Do anomalije može doći prilikom unosa podataka o novom zaposlenom kada se unesu podaci koji se razlikuju od već postojećih. Postavlja se pitanje koji red zapisa sadrži ispravne podatke. Primer pojave anomalije prilikom unosa novih podataka prikazan je u tabeli *zaposleni* koja je prikazana u nastavku:

Tabela *zaposleni*

RID	Ime	Posao	OID	NazivO	MestoO
1	Lazar	programer	10	IT sektor	Voždovac
2	Luka	vozač	20	Logistika	Novi Beograd
3	Aca	DBA dizajner	10	Računovodstvo	Novi Beograd

Anomalije pri ažuriranju podataka. Nastaju kada se vrše promene u podacima, ali zbog loše strukture baze podataka, ažuriranja ne mogu biti konzistentna. Na primer, ako se promeni naziv odeljenja Logistika u Transport, a ta promena nije reflektovana u svim redovima gde se taj naziv odeljenja pojavljuje, dolazi do neslaganja i gubitka konzistentnosti.

Primer pojave anomalije prilikom ažuriranja podataka prikazan je u tabeli *zaposleni* koja je prikazana u nastavku.

Tabela *zaposleni*

RID	Ime	Posao	OID	NazivO	MestoO
1	Lazar	programer	10	IT sektor	Voždovac
2	Luka	vozač	20	Transport	Novi Beograd
3	Aca	DBA dizajner	10	IT sektor	Voždovac
4	Mirko	magacioner	20	Transport	Novi Beograd
5	Žika	domar	20	Logistika	Novi Beograd

Anomalije pri brisanju podataka. Javljaju se kada se pokuša brisanje podataka iz tabele, ali zbog nedostatka referencijalne integritetne zaštite ili drugih problema u strukturi baze podataka, brisanje može dovesti do gubitka važnih podataka ili neregularnih stanja. Uzmimo primer gašenja odeljenja Logistika. Brisanje podataka o odeljenju, zbog loše osmišljene struktura tabele zaposleni iz analiziranog primera, prouzrokuje i brisanje podataka o zaposlenima. Ukoliko podaci o zaposlenima iz odeljenja koji se gasi ne budu prethodno sačuvani, dolazi do gubitka i neregularnosti podataka.

Primer pojave anomalije prilikom brisanja podataka prikazan je u tabeli zaposleni koja je prikazana u nastavku:

Tabela *zaposleni*

RID	Ime	Posao	OID	NazivO	MestoO
1	Lazar	programer	10	IT sektor	Voždovac
2	Luka	vozač	20	Logistika	Novi Beograd
3	Aca	DBA dizajner	10	IT sektor	Voždovac
4	Mirko	magacioner	20	Logistika	Novi Beograd
5	Žika	domar	20	Logistika	Novi Beograd

Dobar dizajn baze podataka podrazumeva:

- **Strukturu šeme zasnovanu na pravilima integriteta podataka.** Uključuje definisanje primarnih ključeva, spoljašnjih ključeva i drugih ograničenja kako bi se obezbedio integritet podataka i sprečilo unošenje neregularnih ili nekonzistentnih podataka.
- **Minimalnu redundanciju podataka.** Dobar dizajn baze podataka minimizuje redundanciju podataka, tj. sprečava višestruko skladištenje istih ili sličnih podataka. Ovo ne samo da štedi prostor za skladištenje, već i olakšava održavanje i ažuriranje podataka.
- **Skladištenje svih neophodnih podataka.** Svaki bitan podatak treba da bude adekvatno zabeležen i skladišten u bazi podataka. To uključuje sve podatke koje su potrebne za pravilno funkcionisanje sistema ili za izvršavanje različitih operacija.
- **Izbegavanje pojave anomalija.** Dobar dizajn baze podataka treba da spreči pojavu anomalija kao što su problemi pri dodavanju, ažuriranju ili brisanju

podataka. To se postiže primenom mehanizma normalizacije, pravilnih ograničenja integriteta podataka i pažljivog planiranja strukture baze podataka.

U suštini, dobar dizajn baze podataka teži ka stvaranju strukture koja efikasno organizuje podatke, održava njihovu tačnost i integritet, minimalizuje redundansu i sprečava pojavu anomalija prilikom manipulacije podacima.

Primer: Neka je relacija *izborni_predmet* definisana nad šemom IZBORNI_PREDMET (BrInd, Prezime, Ime, PID, Naziv) tako da sadrži podatke o studentu (broj indeksa, prezime, ime) i podatke o predmetu koji je birao (šifra predmeta, naziv). Jedan student može da izabere više predmeta i jedan predmet može da izabere više studenata. Za potrebe ilustracije anomalija u podacima relacija i šema relacije IZBORNI_PREDMET je osmišljena sa lošom strukturom.

Sadržaj relacije *izborni_predmet* prikazan je u nastavku:

BrID	**Prezime**	**Ime**	**BrTel**	**PID**	**Naziv**
RT-23/21	Marković	Ivan	069123456	B12	Baze podataka
NRT-12/21	Petrović	Marko	061234567	B12	Baze podataka
RT-23/21	Marković	Ivan	069123456	P23	Programiranje
IS-34/21	Milović	Milica	064567890	A14	Analiza podataka
RT-23/21	Marković	Ivan	069123456	A14	Analiza podataka

Na osnovu prikaza sadržaja relacije IZBORNI_PREDMET evidentna je pojava redundansa odnosno višestrukog ponavljanja podataka. Redundansa prouzrokuje probleme prilikom manipulacije i ažuriranja podataka i to:

- Višestruko unošenje podataka. Podaci o studentu se unose se onoliko puta koliko je predmeta izabrao. Podaci o predmetu se takođe unose za svakog studenta koji je birao predmet.
- Višestruko ažuriranje podataka. Izmena nekog od podataka koji se odnose na studenta (npr. broj telefona) potrebno je da bude izvršena u svim redovima zapisa gde se pojavljuju podaci o studentu za koga se menjaju podaci;
- Višestruko brisanje podataka. Brisanje podataka o predmetu podrazumeva brisanje redova zapisao onoliko puta koliko je predmet izabran.

Izbegavanje višestrukog unosa podataka o predmetu ili studentu (podaci se beleže samo priliko prvog upisa, a u ostalim zapisima se unosi NULL), gube se informacije koje se dobijaju izvršavanjem određenih upita.

Na primer:

BrID	**Prezime**	**Ime**	**BrTel**	**PID**	**Naziv**
RT-23/21	Marković	Ivan	069123456	B12	Baze podataka
NRT-12/21	Petrović	Marko	061234567	B12	NULL
RT-23/21	NULL	NULL	NULL	P23	Programiranje
IS-34/21	Milović	Milica	064567890	A14	Analiza podataka
RT-23/21	NULL	NULL	NULL	A14	NULL

U ovom primeru, upit za izdvajanje podataka o studentima koji su izabrali predmet Baze podataka ili predmet Analiza podataka mogao bi generisati nepotpune podatke. Na primer, student Petrović Marko je izabrao predmet Baze podataka, ali zbog izbegavanja višestrukog unosa i unosa NULL vrednosti za atribut Naziv, zapis o ovom studentu ne bi se pojavio u prikazu.

Loše osmišljena struktura relacije IZBORNI_PREDMET prozrokuje pojavu sledećih problema:

- Anomalija unosa: Unos podataka o novom studentu podrazumeva i unos podataka o izabranom predmetu što u može da bude problem ukoliko student izborne predmete ne bira prilikom upisa.
- Anomalija brisanja: Ukoliko se određeni predmet više ne drži, uklanjanjem podataka o tom predmetu bi se istovremeno uklonili i svi podaci o studentu.

Uzimajući u obzir da student može da izabere više predmeta, a da predmet može da bude izabran od strane više studenata, relacija IZBORNI_PREDMET treba da bude rezultat realizacije veze više prema više. Kako bi se izbeglo višestruko unošenje i ažuriranje podataka kao i gubljenje vezanih podataka prilikom brisanja, podaci o predmetima i studentima bi trebalo da se čuvaju u posebnim relacijama. Relacija IZBORNI_PREDMET, kao rezultat realizacije veze više prema više, treba da beleži samo podatke o ključevima koji ukazuju na studenta i predmet koji je izabrao.

5.3 FUNKCIONALNE ZAVISNOSTI

Funkcionalne zavisnosti su formalno definisane od strane kanadskog matematičara Williama W. Armstronga u radu iz 1974. godine pod nazivom *"Dependency Structures of Database Relationships"*. Poznate su kao Armstrongove aksiome, koje uključuju reflektivnost, pravilo uvećanja, tranzitivnost, uniju, dekompoziciju i pseudotranzitivnost. Primenom Armstrongovih aksioma na početni skup funkcionalnih zavisnosti, mogu se izvesti nove funkcionalne zavisnosti unutar šeme relacije. Identifikovanje postojanja funkcionalnih zavisnosti predstavlja osnovu za primenu mehanizma normalizacije. Definiše se kao povezanost između atributa u kojoj jedan ili više atributa određuje vrednost drugog atributa u istoj relaciji.

Formalna definicija funkcionalne zavisnosti:

Neka je r relacija definisana nad šemom relacije $R(A; C)$, X i Y podskupovi skupa atributa $A = \{A_1, \dots, A_n\}$, a C skup ograničenja.

Funkcionalna zavisnost između atributa X i atributa Y znači da se svakom elementu iz skupa domena atributa X pridružuje jedan i samo jedan elemenat iz skupa domena atributa Y. Može se reći da vrednost atributa X određuje vrednost atributa Y i označava se:

$$X \rightarrow Y$$

Atribut sa leve strane funkcionalne zavisnosti predstavlja početnu tačku i naziva se **odrednica** ili **determinanta**. Determinanta određuje odnosno determiniše vrednost atributa sa desne strane, koji se naziva **zavisni atribut**. Značenje navedenog izraza $X \rightarrow Y$ se može interpretirati na dva načina i to:

- Y funkcionalno zavisi od X ili
- X funkcionalno određuje Y.

Funkcionalne zavisnosti se mogu podeliti u sledeće kategorije:

- **Jednostavne.** Ovo su osnovne zavisnosti gde jedan atribut determiniše vrednost drugog atributa.
- **Kompozitne.** Javljaju se kada više atributa zajedno determinišu vrednost drugog atributa.
- **Parcijalne.** Predstavljaju specifičan tip zavisnosti između atributa u relaciji. Javljuju se kada vrednost nekog atributa u relaciji funkcionalno zavisi od dela složenog ključa, ali ne i od celog složenog ključa.
- **Tranzitivne.** Nastaju kada jedan atribut determiniše drugi atribut, koji dalje determiniše treći atribut.

Razumevanje funkcionalnih zavisnosti je ključno za dizajniranje optimalno dobre strukture relacionih baza podataka. Omogućavaju identifikaciju ključeva kandidata, definisanje primarnih ključeva, sprovođenje procesa normalizacije u cilju ostvarivanja minimalne redundanse i sprečavanje pojave anomalije podataka.

Glavni cilj identifikovanja funkcionalnih zavisnosti je očuvanje integriteta podataka. Tokom procesa definisanja ključeva kandidata, posebna pažnja se posvećuje pronalaženju funkcionalnih zavisnosti koje važe za sve moguće vrednosti atributa u relaciji kako bi se postavila osnovna ograničenja integriteta. Jedno od najvažnijih ograničenja integriteta jeste identifikacija kandidata ključeva, od kojih se jedan bira kao primarni ključ relacije. U procesu normalizacije, na osnovu identifikovanih funkcionalnih zavisnosti, relacije se razlažu na manje, bolje organizovane relacije. Ovo omogućava efikasniju manipulaciju i upravljanje podacima kao i održavanje integriteta baze.

Primer: U prethodnom primeru su opisani problemi koji nastaju zbog loše osmišljene šeme relacije IZBORNI_PREDMET(BrInd, Prezime, Ime, PID, Naziv). Primarni ključ šeme relacije IZBORNI_PREDMET čine dva atributa: broj indeksa i šifra predmeta koji je student izabrao (BrInd, PID). Može se reći da definisani primarni ključ ukazuje na svaki zapis relacije IZBORNI_PREDMET odnosno jedinistveno određuje prezime, ime studenta i naziv izabranog predmeta.

Na osnovu semantike atributa, mogu se identifikovati sledeće funkcionalne zavisnosti:

- BrInd → Ime
- BrInd → Prezime
- PID → Naziv

Navedene funkcionalne zavisnosti se mogu protumačiti na sledeći način:

- Broj indeksa jednoznačno određuje ime studenta.
- Broj indeksa jednoznačno određuje prezime studenta.
- Šifra predmeta jednoznačno određuje naziv predmeta.

Na osnovu prethodno navedenog, u šemi relacije IZBORNI_PREDMET, evidentno je da samo broj indeksa precizno određuje ime i prezime studenta. To znači da neključni atributi Prezime i Ime funkcionalno zavise od dela složenog ključa. Takođe, atribut PID koji je deo složenog primarnog ključa determiniše samo jedan neključni atribut Naziv što znači da vrednost ovog atributa funkcionalno zavisi samo od dela primarnog ključa. Utvrđene funkcionalne zavisnosti predstavljaju parcijalne funkcionalne zavisnosti i ukazuju na loše osmišljenu strukturu relacije IZBORNI_PREDMET. Postupak uklanjanja parcijalnih funkcionalnih zavisnosti podrazumeva primenu mehanizma normalizacije kako bi se orginalna polazna relacija razložila na manje, normalizovane relacije.

5.4 SPROVOĐENJE POSTUPKA NORMALIZACIJE

Nakon što je kreiran model baze podataka, ključno je sprovesti analizu projektovane šeme. Ova analiza fokusira se na primenu postupaka koji osiguravaju poštovanje striktnih principa relacionog modela podataka. Kao rezultat analize, rezultujući model baze podataka treba da obuhvati dobro definisane i organizovane relacije. Cilj je smanjiti pojavu redundantnosti podataka na minimum, kao i sprečiti pojavu anomalija.

Normalizacija je formalna metoda koja se oslanja na funkcionalne zavisnosti, odnosno na veze između ključeva i ostalih atributa relacije. Osnovni princip na kojem počiva mehanizam normalizacije podataka jeste da dobro formirana relacija ne bi trebalo da obuhvata više od jednog poslovnog koncepta. Tokom procesa normalizacije, ključno je identifikovati sve nepoželjne funkcionalne zavisnosti u relacijama. Svaka relacija koja sadrži nepoželjne funkcionalne zavisnosti treba da

bude razložena na više manjih, dobro formiranih relacija. Postupak razdvajanja se zasniva na strogo formalnim metodama kako bi se dobile nove šeme relacija koje ne sadrže početne nepoželjne zavisnosti. Od ključnog je značaja sačuvati sve atribute kako bi se izbegao gubitak podataka. Efikasnost postupka normalizacije se može oceniti kroz sposobnost rekonstrukcije početne šeme relacija iz razloženih.

Proces normalizacije sprovodi se u više faza kako bi se potvrdilo da li relacijska šema zadovoljava određenu normalnu formu, pri čemu svaka faza odgovara određenoj normalnoj formi. Normalna forma predstavlja standardizovanu organizaciju podataka u relaciji radi eliminisanja redundancije i očuvanja integriteta podataka. Svaka normalna forma definiše specifične uslove koje relacija treba da ispuni kako bi se smatrala normalizovanom prema tom standardu. Mehanizam normalizacije podrazumeva sprovođenje procesa koji se odvija odozgo nadole i obuhvata evaluaciju svake relacije prema kriterijumima za normalne forme. Može se smatrati postupkom dizajna relacija putem analize.

E. Kod je predložio tri normalne forme koje je nazvao prva (1NF), druga (2NF) i treća normalna forma (3NF). Kasnije su Bojs i Kod predložili jaču definiciju treće normalne forme Bojs-Kod normalnu formu (BCNF). Sve ove normalne forme zasnovane su na jednom analitičkom alatu: funkcionalnim zavisnostima između atributa relacije. Naknadno su predložene četvrta normalna forma (4NF) i peta normalna forma (5NF), zasnovane na konceptima viševrednosnih zavisnosti i zavisnosti pridruživanja. Svaka naredna normalna forma postavlja strožije uslove i uvodi više restrikcija, što smanjuje mogućnost pojave anomalija u bazi podataka.

Normalizacija nije samo mehanički postupak. Neophodno je utvrditi i postojanje "prirodnih zavisnosti". U prethodno razmatranom primeru atribut koji beleži podatak o broju indeksa (BrInd) uvek mora biti povezan sa imenom i prezimenom studenta. Ova tip zavisnosti se smatra prirodnim i ovakava povezanost između atributa ne sme biti izgubljena. Može se zaključiti da sprovođenjem odgovarajućih formalnih metoda za rastavljanje relacija, važno je ne samo uklonite nepoželjne funkcionlane zavisnosti nego i sačuvati prirodne funkcionalne zavisnosti.

Uobičajeni cilj normalizacije jeste dostizanje treće normalne forme (3NF) koja se smatra optimalnim balansom između funkcionalnosti i jednostavnosti implementacije. U praksi, normalizacija iznad 3NF može dodatno otežati dizajn baze podataka i na taj način uticati na funkcionalnost sistema. Neophodno je istaći da su svi nivoi normalizacije kumulativni, što implicira da baza podataka koja zadovoljava 3NF mora takođe da ispuni sve zahteve prethodnih nivoa normalizacije.

5.4.1 PRVA NORMALNA FORMA (1NF)

Prva normalna forma (1NF) se smatra sastavnim delom formalne definicije relacije u relacionom modelu. Definisana je s ciljem sprečavanja prisustva višestrukih vrednosti atributa, složenih atributa i njihovih kombinacija. Propisuje pravilo da svaki

atribut mora sadržati samo atomarne (jednostavne, nedeljive) vrednosti, te da vrednost svakog atributa u n-torki mora biti jedinstvena vrednost iz domena tog atributa. U skladu s tim, zabranjeno je prisustvo višestrukih ili složenih atributa, kao i relacija kao atributa unutar n-torki ili njihovih kombinacija. Prva normalna forma (1NF) onemogućava prisustvo relacija unutar drugih relacija ili relacija kao vrednosti atributa unutar redova zapisa. Jedine vrednosti atributa koje su dozvoljene u okviru 1NF su pojedinačne atomarne (nedeljive) vrednosti.

Primer: Data je šema relacije Kupovina (PID, DatumP, KID, Kupac, AID, ArtNaziv, Cena, Kolicina).

Trenutni sadržaj relacije *kupovina* je prikazan na slici 5.4.1.1.

ID	DatumP	KID	Kupac	AID	ArtNaziv	Cena	Kolicina
1006	21.10.2018.	22	Marković Luka	7 5 8	Olovka Sveska Blok	50 120 70	5 2 1
1007	07.07.2019.	29	Lazović Ivana	10 1	Rezač Naliv pero	100 450	2 1
1008	21.12.2018.	22	Marković Luka	7 5 8	Olovka Sveska Blok	50 120 70	2 1 3

Slika 5.4.1.1 Relacija kupovina koja nije u 1NF

Analizom relacije *kupovina* može se zaključiti:

- Primarni ključ relacije je atribut PID koji jedinstveno identifikuje svaki zapis reda.
- Vrednosti svakog atributa zadovoljavaju postavljeno ograničenje za tip podataka.

Većina upita nad ovakvom relacijom su kompleksni i složeni. Ukoliko je potrebno izdvojiti porudžbine kupca čije je prezime Marković neophodno je koristiti funkciju za izdvajanje dela stringa kako bi se iz atributa Kupac izdvojio podatak samo o prezimenu.

Sledeći problem se javlja u slučaju postavljanja upita za prikaz ukupne količine olovaka koje je kupio kupac čije je prezime Marković. Navedeni problemi su posledica postojanja složenog atributa (atribut Kupac) i pojave više vrednosti u atributima koji beleže podatke o nazivu artikla, ceni i količini. Ovo ukazuje da relacija *kupovina* ne zadovoljava pravila prve normalne forme.

Da bi šema relacije *kupovina* bila u 1NF definiše se relacija u kojoj dolazi do ponavljanja vrednosti atributa u redovima zapisa. Prethodno definisan primarni ključ PID ne identifikuje jedinstveno sve redove zapisa.

Definisan je novi primarni ključ što je u ovom slučaju složeni ključ koji se sastoji od atributa PID, DatumP, KID, AID. Atribut su prosti (osim atributa DatumP koji nije

potrebno razdvajati na proste atribute dan, mesec i godina). Svi atributi beleže atomarne vrednosti za svaku *n*-torku relacije.

Trenutni sadržaj relacije *kupovina* u 1NF je prikazan na slici 5.4.1.2.

PID	DatumP	KID	KupacPrez	KupacIme	AID	ArtNaziv	Cena	Kolicina
1006	21.10.2018.	22	Marković	Luka	7	Olovka	50	5
1006	21.10.2018.	22	Marković	Luka	5	Sveska	120	2
1006	21.10.2018.	22	Marković	Luka	8	Blok	70	1
1007	07.07.2019.	29	Lazović	Ivana	10	Rezač	100	2
1007	07.07.2019.	29	Lazović	Ivana	1	Naliv pero	450	1
1008	21.12.2018.	22	Marković	Luka	7	Olovka	50	2
1008	21.12.2018.	22	Marković	Luka	5	Sveska	120	1
1008	21.12.2018.	22	Marković	Luka	8	Blok	70	3

Slika 5.4.1.2 Relacija kupovina u 1NF

Dobijena šema relacije jeste u prvoj normalnoj formi, ali i dalje ima lošu strukturu. Postupak poboljšanja strukture relacije *kupovina,* u kojoj je evidentno ponavljanje podataka, podrazumeva formiranje novih, manjih i bolje organizovanih relacija. Primarni ključ novih relacija će da bude kombinacija primarnog ključa originalne relacije i atributa iz novoformirane relacije za jedinstvenu identifikaciju.

U relaciji *kupovina* identifikovane su i parcijalne funkcionalne zavisnosti neključnih atributa od dela složenog primarnog ključa:

- KID → KupacPrez, KupacIme
- AID → ArtNaziv, Cena
- PID, DatumpP,KID → Kolicina

Postupak eliminisanja parcijalnih funkcionalnih zavisnosti se realizuje primenom pravila druge normalne forme.

5.4.2 DRUGA NORMALNA FORMA (2NF)

Druga normalna forma (2NF) se oslanja na koncept potpune funkcionalne zavisnosti. Relacija se smatra u drugoj normalnoj formi kada svi neključni atributi potpuno zavise od celog primarnog ključa. Drugim rečima, ne dozvoljavaju se delimične zavisnosti neključnih atributa od bilo kog kandidata ključa u datoj šemi relacije. Ovaj koncept posebno je važan za relacije sa složenim primarnim ključevima, sastavljenim od dva ili više atributa. Relacija ispunjava uslove za 2NF ako već zadovoljava pravila 1NF i ako svaki atribut koji nije deo primarnog ključa potpuno zavisi od primarnog ključa. Relacija čiji primarni ključ sadrži samo jedan atribut automatski se smatra najmanje u 2NF, pod uslovom da već ispunjava zahteve 1NF. U slučaju relacije koja nije u 2NF, moguće su anomalije pri ažuriranju podataka.

Primer: Relacija *kupovina* definisana šemom relacije Kupovina (PID, DatumP, KID, KupacPrez, KupacIme, AID, ArtNaziv, Cena, Kolicina) je u 1NF. Uočene su parcijalne funkcionalne zavisnosti koje treba eliminisati.

- KID → KupacPrez, KupacIme
- AID → ArtNaziv, Cena
- PID, DatumpP,KID → Kolicina

Eliminisanje parcijalanih funkcionalnih zavisnosti podrazumeva razlaganje početne relacije *kupovina* na manje, bolje organizovane relacije. Novodobijene manje relacije treba da budu formirane tako da je u njima eliminisano postojanje parcijalnih funkcionalnih zavisnosti.

Relacija *artikal* je definisana šemom Artikal (AID, ArtNaziv, Cena). Relacije ispunjava uslove 2NF: zadovoljava uslove 1NF i nema parcijalnih funkcionalnih zavisnosti neključnih atributa od dela primarnog ključa. U ovom slučaju, relacija *artikal* ima definisan primarni ključ koji sadrži samo jedan atribut što eliminiše mogućnost pojave parcijalnih funkcionalnih zavisnosti.

Relacija *stavkePorudzbine* je definisana šemom stavkePorudzbine (PID, AID, Kolicina). Relacije ispunjava uslove 2NF: zadovoljava uslove 1NF i nema parcijalnih funkcionalnih zavisnosti neključnih atributa od dela primarnog ključa.

Neključni atribut Kolicina zavisi od celog složenog ključa što se može prikazati kao potpuna funkcionalna zavisnost:

PID, AID → Kolicina

Relacija *porudzbine* je definisana šemom Porudzbine (PID, DatumP, KID, KupacPrez, KupacIme). Relacije ispunjava uslove 2NF: zadovoljava uslove 1NF i nema parcijalnih funkcionalnih zavisnosti neključnih atributa od dela primarnog ključa. Kao i kod relacije *artikal* i za relaciju *porudzbine* definisan je primarni ključ koji sadrži samo jedan atribut što eliminiše mogućnost pojave parcijalnih funkcionalnih zavisnosti.

Relacije *artikal* i *stavkePorudzbine* zadovoljavaju pravila i 3NF, što znači da u ovim relacijama nisu identifikovane tranzitivne funkcionalne zavisnosti između neključnih atributa. Međutim relacija *porudzbine* zadovoljava uslove 2NF, ali ne i 3NF što podrazumeva postojanje tranzitivnih funkcionalnih zavisnosti.

5.4.3 TREĆA NORMALNA FORMA (3NF)

Treća normalna forma (3NF) se zasniva na konceptu tranzitivne zavisnosti. Funkcionalna zavisnost X → Y u šemi relacije R je tranzitivna zavisnost ako postoji skup atributa Z u R koji nije kandidatni ključ niti podskup bilo kog ključa R, i ako važe i X → Z i Z → Y.

Prema prvobitnoj definiciji Edgara Koda, šema relacije R smatra se da je u trećoj normalnoj formi (3NF) ukoliko ispunjava drugu normalnu formu (2NF) i ako nijedan atribut koji nije deo ključa nije u tranzitivnoj zavisnosti sa primarnim ključem. Drugim rečima, šema relacije se smatra u 3NF ako je već u 1NF i 2NF, a nema tranzitivnih zavisnosti koje bi doveli do toga da neki neključni atribut bude zavisan od bilo kog kandidata ključa, uključujući primarni ključ.

U šemama relacija koje su normalizovane do 2NF, može se pojaviti problem anomalijskog ažuriranja zbog tranzitivnih zavisnosti. Ovi problemi se moraju rešiti putem normalizacije do treće normalne forme (3NF). Normalizacija šeme relacije koja nije u 3NF obuhvata eliminisanje tranzitivnih zavisnosti razlaganjem relacije na manje relacije kako bi se tranzitivno zavisni atributi izmestili iz polazne relacije.

Postupak je sličan postupku normalizacije do druge normalne forme. Identifikuju se tranzitivne zavisnosti, formira se nova relacija sa atributima koji se izdvajaju iz polazne relacije. Iz polazne relacije se izbacuju atributi koji su bili tranzitivno zavisni, a koji su prethodno izdvojeni u posebnu novoformiranu relaciju.

Primer: U prethodnom primeru utvrđeno je da relacija *porudzbine* definisana šemom Porudzbine (PID, DatumP, KID, KupacPrez, KupacIme) ne zadovoljava uslove 3NF zbog postojanje tranzitivnih funkcionalnih zavisnosti.

Uočena je sledeća tranzitivna funkcionalna zavisnosti:

PID → KID → KupacPrez, KupacIme

Postupka normalizacije relacije *porudzbine* do treće normalne forme podrazumeva razbijanje relacije *porudzbine* na dve relacije. Kreirane su relacija *porudzbenica* definsana šemom relacije Porudzbenica (PID, DatumP, KID) i relacija *kupac* definisana šemom relacije Kupac(KID, KupacPrez, KupacIme). Novoformirane relacije *porudzbenica* i *kupac* normalizovane su do 3NF što podrazumeva da su u 1NF i 2NF i da nema identifikovanih tranzitivnih funkcionalni zavisnosti neključnih atributa sa primarnim ključem.

5.4.4 PREGLED NORMALNIH FORMI, PRAVILA I POSTUPAK NORMALIZACIJE

Dizajniranje šema relacija predstavlja značajan deo postupka modelovanja relacionih baza podataka. Na osnovu prethodno navedenog može se zaključiti da šeme relacija treba budu tako formirane da je onemogućena pojava kako parcijalnih tako i tranzitivnih zavisnosti, jer ovi tipovi zavisnosti dovode do problema sa ažuriranjem podataka.

U tabeli 5.4.4.1 dat je prikaz formalnih metoda osnovnih normalnih formi. Izdvojena su pravila za proveru da li je šema relacije u 1NF, 2NF, 3NF kao i postupak normalizacije za svaku normalnu formu.

Tabela 5.4.4.1 Pregled osnovnih normalnih formi

Normalna forma	Pravila	Postupak normalizacije
Prva normalna forma (1NF)	Relacija ne sme sadržati višestruke vrednosti atributa ili ugnježđene relacije.	Formirati proste atrbute koji sadrže atomarne vrednosti. Zbog pojave redundantosti podataka prilikom pretvaranja viševrednosnih u proste atributa i eliminisanja ugneždenih relaciju formirati nove relacije.
Druga normalna forma (2NF)	U relacijama gde primarni ključ sadrži više atributa, nijedan neključni atribut ne sme biti funkcionalno zavisan od dela primarnog ključa.	Atribute koji su funkcionalno zavisni od dela ključa treba razdvojiti i organizovati u nove relacije. Originalnu relacija sa primarnim ključem i atributima koji su u potpunosti funkcionalno zavisni od njega treba zadržati.
Treća normalna forma (3NF)	Relacija ne sme imati neključni atribut koji je funkcionalno determinisan od drugog neključnog atributa (ili od skupa neključnih atributa).	Neophodno je razdvojiti i organizovati relaciju tako da ne postoji tranzitivna zavisnost neključnog atributa u odnosu na primarni ključ.

5.4.5 BOYCE-CODD NORMALNA FORMA (BCNF)

Bojs-Kodova normalna forma (BCNF) predstavlja strožiju verziju treće normalne forme (3NF). Fokusira se na eliminaciju određenih vrsta redundantnosti podataka u relacijama i dodatnih tranzitivnih zavisnosti.

Relacija se smatra u Bojs-Kodovoj normalnoj formi (BCNF) ako svaka zavisnost između neključnih atributa zavisi isključivo od kandidatnih ključeva. Drugim rečima, u relaciji u BCNF, ne postoji nijedna funkcionalna zavisnost između neključnih atributa koji su delimično zavisni od nekog dela kandidatnog ključa.

Glavna ideja BCNF-a je da se eliminišu takozvane "višeznačne" funkcionalne zavisnosti, gde se neključni atributi mogu funkcionalno zavisiti od samo jednog dela kandidatnog ključa, a ne od celog ključa. Ova normalna forma osigurava minimalnu redundanciju podataka i olakšava održavanje integriteta podataka.

Da bi se relacija dovela u BCNF, obično je potrebno dekomponovati ili podeliti originalnu relaciju na više manjih relacija, tako da svaka nova relacija zadovoljava uslove BCNF-a. Ipak, u određenim situacijama, možda nije poželjno sprovoditi normalizaciju sve do BCNF-a, jer takav postupak može dovesti do gubitka nekih prirodnih funkcionalnih zavisnosti. U tim slučajevima, neophodno je uneti odgovarajuća ograničenja direktno u kod, kako bi se izbeglo dovođenje baze podataka u nekonzistentno stanje tokom ažuriranja.

5.5 DENORMALIZACIJA

Denormalizacija predstavlja proces optimizacije baze podataka, usmeren na poboljšanje performansi sistema. Ona se ostvaruje transformacijom relacija u slabiju normalnu formu. Za razliku od procesa normalizacije koji ima za cilj strukturiranje efikasnijih šema relacija, denormalizacija se primenjuje s namerom smanjenja stepena normalizacije, obično radi unapređenja performansi sistema.

Suprotno normalizaciji, denormalizacija podrazumeva dodavanje redundancije podataka u bazu kako bi se ubrzali i pojednostavili upiti. Ovaj proces može uključivati spajanje relacija koje su prethodno dekomponovane tokom normalizacije, dodavanje redundantnih atributa ili čak čuvanje izračunatih vrednosti radi bržeg pristupa.

Denormalizacija se obično primenjuje u situacijama gde su performanse ključne, kao što su sistemi koji zahtevaju brze upite ili analitiku velikih podataka. Ipak, zbog promene strukture baze podataka koju denormalizacija donosi, moguće je da će doći do narušavanja funkcionalnosti sistema ili problema u procesu izvršavanja, jer pojedini upiti mogu početi raditi sporije usled promena.

Čak i kada performanse nisu direktno narušene, mogu se pojaviti anomalije pri unosu, izmeni i brisanju podataka, što može ugroziti referencijalni integritet baze podataka. U takvim situacijama, neophodno je implementirati ograničenja koja osiguravaju referencijalni integritet, umesto oslanjanja samo na denormalizaciju kao rešenje za većinu problema.

Važno je imati na umu da loše performanse baze podataka ne moraju uvek biti direktna posledica potrebe za denormalizacijom. Često je potrebno redizajnirati strukturu baze podataka zbog neadekvatno osmišljenog modela koji ne može pružiti potrebne performanse.

Ipak, denormalizacija ne garantuje unapređenje performansi. Problemi koje je baza podataka imala mogu ostati nepromenjeni ili se čak pogoršati nakon denormalizacije. Stoga je pravilno balansiranje između normalizacije i denormalizacije ključno za efikasno upravljanje bazama podataka.

5.6 PITANJA I ZADACI

1. U slučaju pojave redundanse u podacima, koji se problemi javljaju?
2. Šta su anomalije podataka?
3. Koji probemi mogu nastati prilikom pojave anomalije u podacima?
4. Šta je glavni uzrok pojave anomalija u podacima, a šta je posledica istih?
5. Na koji način se rešavaju problemi anomalija i redundanse?
6. Objasniti koncept funkcionalnih zavisnosti.
7. Navesti tipove i karaketristike funkcionlanih zavisnosti.
8. Objasniti metod primene prve normalne forme.
9. Objasniti metod primene druge normalne forme.
10. Objasniti metod primene treće normalne forme.

6 OSNOVE SQL UPITNOG JEZIKA

Cilj ovog poglavlja je pružiti temeljno objašnjenje SQL naredbi koje omogućavaju manipulaciju podacima iz jedne tabele relacione baze podataka. Opisani su tipovi podataka podržani u većini DBMS sistema. Definisane su DDL naredbe CREATE, ALTER i DROP, koje se koriste za kreiranje, izmenu i brisanje baza podataka i tabela. Detaljno je objašnjena kompletna struktura SELECT naredbe, a takođe su definisane i ugrađene funkcije za rad s numeričkim, datumskim i znakovnim podacima. Posebna pažnja posvećena je agregatnim funkcijama, kao i primeni GROUP BY i HAVING klauzula unutar njih.

6.1 UVOD

SQL (engl. *Structured Query Language*) je standardizovan, deklarativni jezik za upravljanje relacionim bazama podataka što znači da korisnici definišu šta žele da urade, a ne kako to da urade. Primenom odgovarajućih tipova komandi koristi se za manipulaciju, upravljanje, modifikaciju i pretraživanje podataka, kao i za kreiranje različitih objekta baze podataka i kontrolu pristupa istim. Koncepti koji su prethodili SQL-u datiraju iz 1970.-ih godina. Relacioni model podataka, razvijen od strane Edgara F. Codda u IBM-u, bio je osnova za SQL. IBM je razvio prvi komercijalni sistem za upravljanje bazama podataka (DBMS) poznat kao System R, koji je uključivao rani oblik SQL-a. Prva verzija SQL-a, nazvana SEQUEL (engl. *Structured English Query Language*), razvijena je u IBM Research Laboratory u 1970-im godinama. Ovaj jezik je bio inspirisan engleskim jezikom i omogućavao je korisnicima da definišu i manipulišu podacima pomoću jednostavnih i intuitivnih komandi.

Kako je popularnost relacionih baza podataka rasla, potreba za standardizacijom SQL-a postala je očigledna. Prvi zvanični standard za SQL, nazvan SQL-86, objavljen je 1986. godine. Tokom narednih godina, SQL je proširen i unapređen kroz različite verzije standarda. SQL-89 i SQL-92 dodali su nove funkcionalnosti i sintaksu. Ovi standardi definišu osnovne sintakse i funkcionalnosti SQL-a, dok specifične implementacije mogu imati svoje dodatne funkcije i proširenja. Svaka nova verzija standarda nastoji da unapredi SQL jezik i prilagodi ga zahtevima modernih sistema i aplikacija. Većina proizvođača DBMS-a proširuje SQL standard dodajući proceduralne elemente, kontrolne strukture, korisnički definisane tipove podataka i druge elemente. Proširenja SQL jezika koja su postala deo standarda uključuju sledeće:

- SQL:1999: Ovo proširenje donelo je značajna poboljšanja, kao što su podrška za prozorske funkcije, rekurzivne upite, definisanje tipova podataka, procedura i funkcija, kao i dodatne funkcije za upravljanje podacima.
- SQL:2003: Ovo proširenje dodalo je podršku za standardizovane JSON funkcije, podršku za XML tipove podataka, i proširene mogućnosti upravljanja transakcijama.
- SQL:2008: Ovaj standard donosi dodatne funkcije i poboljšanja u sintaksi i performansama SQL upita, kao i podršku za XML i bolje definisanje proceduralnih elemenata.
- SQL:2011: Ovaj standard donosi unapređene mogućnosti za upravljanje temporalnim podacima, kao i poboljšanja u podršci za XML i JSON.
- SQL:2016: Ovo proširenje donosi nove funkcionalnosti, uključujući podršku za JSON tipove podataka, podršku za razne SQL upite kao što su GRANT i REVOKE, kao i podršku za poboljšano upravljanje temporalnim podacima.
- SQL:2019: Najnoviji standard donosi dalja poboljšanja u podršci za JSON, podršku za inline SQL (SQL on the fly), kao i dodatne funkcije za analizu podataka.

SQL je postao osnovni alat za rad sa relacionim bazama podataka i ima široku primenu u različitim industrijama i oblastima, uključujući razvoj softvera, analizu podataka, upravljanje sistemima, izveštavanje i još mnogo toga. Iako su se pojavile i druge vrste baza podataka, SQL i dalje ostaje jedan od najvažnijih okruženja za upravljanje podacima zbog svoje jednostavnosti, moći i standardizacije.

SQL naredbe se dele na kategorije na osnovu kojih se definišu podjezici SQL-a: DDL, DML, DCL, TCL

DDL (engl. *Data Definition Language*). Predstavlja naredbe za definisanje, ažuriranje, brisanje objekata baze podataka. U ovu kategoriju spadaju: CREATE, ALTER, DROP ,TRUNCATE .

DML (engl. *Data Manipulation Language*). Obuhvata naredbe za manipulaciju i upravljanje podacima koji se čuvaju u struktuiranom obliku. Uključuje naredbe čitanja, dodavanja, ažuriranja i brisanja podataka. U ovu kategoriju spadaju: SELECT, INSERT INTO, UPDATE, DELETE.

DCL (engl. *Data Control Language*): Ove naredbe se koriste za kontrolu pristupa bazi podataka. To uključuje naredbe za upravljanje privilegijama korisnika i sistemskim podešavanjaima. U ovu kategoriju spadaju: GRANT i REVOKE.

TCL (engl. *Transaction Control Language*): Predstavlja naredbe koje se koriste za upravljanje transakcijama u bazi podataka. Uključuje naredbe koje omogućavaju početak i izvršavanje transakcije, postavljanje i ukljanjanje tačaka transakcije, podešavanje opcija transakcije, poništavanje transakcije u slučaju greške. U ovu

kategoriju spadaju: BEGIN TRANSACTION, COMMIT, SAVEPOINT, RELEASE SAVEPOINT, SET TRANSACTION, ROLLBACK.

6.2 TIPOVI PODATAKA

Numerički, tekstualni i vremenski predstavljaju osnovne kategorije tipova podataka koje podržava većina relacionih baza podataka. U zavisnosti od implementacije, postoje i dodatni tipovi karakteristični za određeni sistem (MySQL, Microsoft SQL Server, PostgreSQL, Oracle, itd.).

6.2.1 NUMERIČKI TIPOVI PODATAKA

Numerički tipovi podataka omogućavaju čuvanje numeričkih vrednosti, kao što su celi brojevi, decimalni brojevi ili tačne vrednosti. Ovi tipovi podataka obično se koriste za čuvanje količina, izračunavanje rezultata ili definisanje numeričkih identifikatora.

U numeričke tipove podataka spadaju:

- **INTEGER.** Ovaj tip podataka se koristi za čuvanje celobrojnih vrednosti. Obično je 32-bitni ili 64-bitni, u zavisnosti od implementacije baze podataka. Na primer, u MySQL-u, tip podataka INT može da čuva celobrojne vrednosti u opsegu od -2^{31} do $2^{31}-1$ (za 32-bitni INT) ili od -2^{63} do $-2^{63}-1$ (za 64-bitni INT).
- **DECIMAL(p, s).** Koristi se za čuvanje decimalnih brojeva sa tačnom preciznošću. Parametri definišu ukupan broj cifara (p) i broj decimalnih mesta (s) koje se mogu čuvati. Na primer, DECIMAL(10, 2) može čuvati brojeve sa 10 cifara, od kojih 2 mogu biti decimalna mesta.
- **FLOAT(p).** Koristi se za čuvanje približnih decimalnih vrednosti. Parametar p definiše broj bitova koji se koriste za čuvanje broja, što utiče na preciznost. Na primer, FLOAT(24) može čuvati brojeve sa približno 7 decimalnih mesta.
- **DOUBLE(p).** Slično FLOAT-u, ovaj tip podataka koristi se za čuvanje približnih decimalnih vrednosti, ali sa većom preciznošću. Parametar p definiše broj bitova koji se koriste za čuvanje broja. Na primer, DOUBLE(53) može čuvati brojeve sa približno 15-16 decimalnih mesta. FLOAT obično koristi manje memorije od DOUBLE, ali može imati manju preciznost. DOUBLE koristi 64 bita memorije i ima veću preciznost od FLOAT. REAL je alternativa za FLOAT u nekim bazama podataka.
- **NUMERIC(p, s).** Ovaj tip podataka sličan je DECIMAL-u i koristi se za čuvanje tačnih decimalnih vrednosti. Parametar p definiše ukupan broj cifara, s broj decimalnih mesta koje se mogu čuvati. DECIMAL ili NUMERIC obično se koriste kada je potrebna tačna preciznost u aritmetičkim operacijama, jer oni čuvaju broj decimalnih mesta tačno onako kako je definisano.

- **BIT / BOOLEAN.** Omogućava čuvanje logičkih vrednosti. BIT obično može čuvati samo jedan bit (0 ili 1), dok BOOLEAN može imati vrednosti TRUE, FALSE ili UNKNOWN.
- **SERIAL/AUTO_INCREMENT.** Koriste se za automatsko generisanje jedinstvenih identifikatora pri unosu novog reda u tabelu. Implementira se u kombinaciji sa tipom INTEGER.
- **MONEY / CURRENCY.** Ovi tipovi podataka se koriste za čuvanje finansijskih vrednosti. Obično su implementirani kao decimalni brojevi sa fiksnim brojem decimalnih mesta, kako bi se očuvala tačnost u finansijskim proračunima.

TINYINT, SMALLINT i BIGINT su tipovi podataka koji se koriste za čuvanje celobrojnih vrednosti, ali sa različitim opsezima vrednosti u poređenju sa standardnim INTEGER tipom podataka.

- **TINYINT.** Omogućava čuvanje celobrojnih vrednosti veoma malog opsega. Obično koristi 1 bajt memorije i može čuvati celobrojne vrednosti u opsegu od -128 do 127 (ako se koristi znakom) ili od 0 do 255 (ako se koristi bez znaka).
- **SMALLINT.** Obezbeđuje čuvanje celobrojnih vrednosti većeg opsega u poređenju sa TINYINT-om, ali manjeg od INT-a opsega. Obično koristi 2 bajta memorije i može čuvati celobrojne vrednosti u opsegu od -32.768 do 32.767 (ako se koristi znakom) ili od 0 do 65.535 (ako se koristi bez znaka).
- **BIGINT.** Ovaj tip podataka se upotrebljava za čuvanje celobrojnih vrednosti veoma velikog opsega. Obično koristi 8 bajtova memorije i može čuvati celobrojne vrednosti u opsegu od -9.223.372.036.854.775.808 do 9.223.372.036.854.775.808 (ako se koristi znakom) ili od 0 do 18.446.744.073.709.551.615. (ako se koristi bez znaka).

Ovi tipovi podataka su korisni kada je potrebno čuvati celobrojne vrednosti različitih opsega u zavisnosti od zahteva aplikacije ili specifičnosti podataka koji se obrađuju. Na primer, TINYINT može biti koristan za čuvanje statusa ili flagova koji zahtevaju samo mali opseg vrednosti, dok BIGINT može biti koristan za čuvanje identifikatora ili brojeva koji mogu biti veoma veliki.

6.2.2 TEKSTUALNI TIPOVI PODATAKA

Tekstualni tipovi podataka omogućavaju čuvanje nizova karaktera različitih dužina. U tekstualne tipove definisane standardnim SQL – om spadaju:

- **CHAR(n).** Koristi se za čuvanje fiksnog broja karaktera. Parametar n predstavlja maksimalnu dužinu niza karaktera koja se može čuvati u polju. Ako je uneseni tekst kraći od specificirane dužine, prazni prostor se dodaje na kraju.
- **VARCHAR(n).** Slično kao i CHAR koristi se podatke koji su u formi niza karaktera, ali u slučaju ovog tipa podataka čuva se promenljiva dužinu niza karaktera. Parametar n predstavlja maksimalnu dužinu niza karaktera koji se

može čuvati u polju. Memorija se koristi samo za dužinu unetog teksta, bez praznog prostora na kraju.

- **TEXT.** Ovaj tip podataka koristi se za čuvanje većih nizova teksta koji mogu biti promenljive dužine. Obično ima veći kapacitet od VARCHAR, ali može biti manje efikasan u smislu performansi za duže nizove teksta.
- **NCHAR(n) / NVARCHAR(n).** Ovi tipovi podataka su slični CHAR i VARCHAR, ali se koriste za Unicode tekst. Parametar n predstavlja maksimalnu dužinu niza karaktera u broju znakova, pri čemu svaki znak može zauzeti više bajtova u memoriji.
- **CLOB (Character Large Object).** Ovaj tip podataka koristi se za čuvanje veoma velikih nizova teksta. Obično se koristi za tekst koji je duži od onog što TEXT može da podrži.
- **ENUM.** Omogućava definisanje liste mogućih vrednosti koje se mogu čuvati u polju. Svaka vrednost mora biti izabrana iz unapred definisane Svaka vrednost u polju ENUM mora biti izabrana iz unapred definisane liste vrednosti. Jedna od prednosti korišćenja ENUM tipa podataka je efikasnost memorije. Unapred definisane tekstualne vrednosti koje se čuvaju u polju ENUM koriste manje memorijskog prostora od ekvivalentnog VARCHAR polja koje bi čuvalo iste tekstualne vrednosti.

6.2.3 VREMENSKI TIPOVI PODATAKA

Vremenski tipovi podataka omogućavaju čuvanje datuma, vremena ili kombinacije datuma i vremena. Ovi tipovi podataka su od suštinskog značaja za aplikacije koje zahtevaju praćenje vremenskih podataka.

- **DATE.** Ovaj tip podataka se koristi za čuvanje kalendarskog datuma. Obično format datuma je 'YYYY-MM-DD', gde YYYY označava godinu, MM mesec, a DD dan.
- **TIME.** Koristi se za čuvanje vremena u toku dana. Obično se format vremena izražava kao 'HH:MM:SS', gde HH predstavlja sate, MM minute, a SS sekunde.
- **DATETIME.** Koristi se za čuvanje tačnog datuma i vremena. Obično čuva vrednosti u formatu 'YYYY-MM-DD HH:MM:SS', gde YYYY označava godinu, MM mesec, DD dan, HH sati, MM minute i SS sekunde. Ovaj tip podataka ne zavisi od vremenske zone i obično se koristi kada je potrebno tačno čuvanje lokalnog vremena bez obzira na vremensku zonu.
- **TIMESTAMP.** Predstavlja tip podataka koji se takođe koristi za čuvanje informacija o datumu i vremenu. Međutim, razlika u odnosu na DATETIME je u tome što TIMESTAMP može biti u formatu sličnom DATETIME ili u tzv. Unix vremenskom formatu. Unix vremenski format predstavlja broj sekundi koji je prošao od 1. januara 1970. godine (poznat kao UNIX epoha). Ovo omogućava predstavljanje vremena kao jednostavnog celobrojnog broja, što može biti korisno za računanje vremenskih razlika ili sortiranje podataka po vremenu.

Dodatno, TIMESTAMP obično čuva informaciju o vremenskoj zoni, što ga čini korisnim za aplikacije koje zahtevaju praćenje vremena u različitim vremenskim zonama.

- TIM**EZONE / TIME WITH TIME ZONE.** Omogućava čuvanje vremenskih informacija uzimajući u obzir vremensku zonu. Ovo je korisno kada su informacije o vremenu distribuirane na geografski različitim lokacijama.
- INTERVAL. Koristi se za čuvanje intervala vremena između dva trenutka u vremenu. Na primer, može se koristiti za čuvanje trajanja ili razlike između dva datuma.

6.2.4 PROŠIRENI TIPOVI PODATAKA

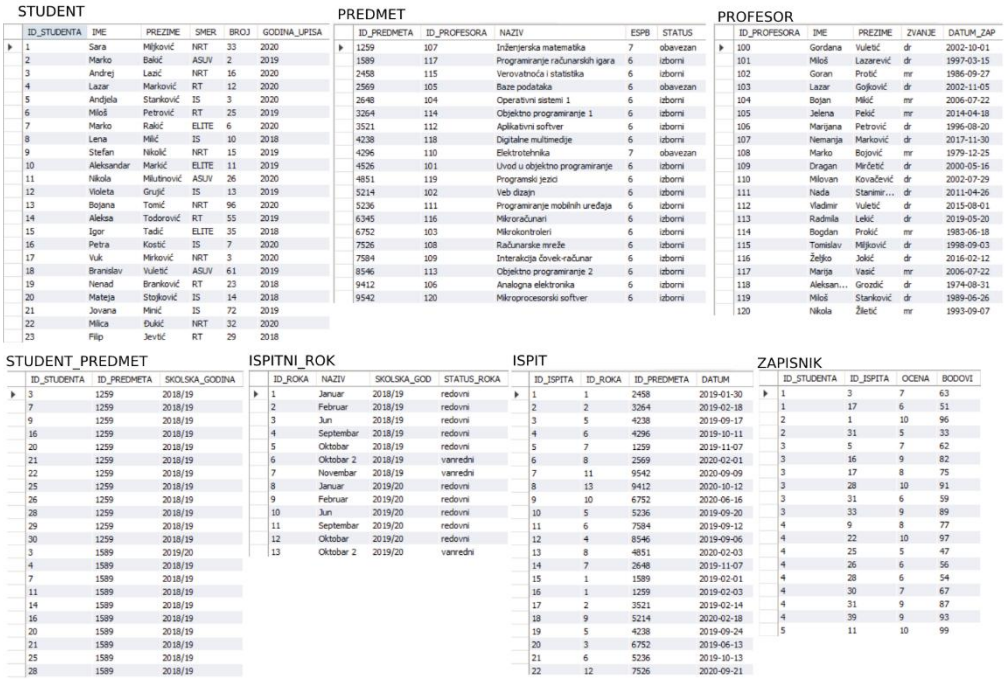
Pored osnovnih tipova podataka, neke baze podataka podržavaju proširene tipove podataka koji mogu biti veoma korisni u određenim scenarijima.

- **UUID** (engl. Universally Unique Identifier) je tip podataka koji se koristi za čuvanje jedinstvenih identifikatora unosa. Ovi identifikatori su globalno jedinstveni i generišu se korišćenjem algoritama koji garantuju jedinstvenost čak i u različitim sistemima ili aplikacijama. UUID-ovi su korisni kada je potrebno jedinstveno identifikovati objekte u sistemu bez potrebe za centralizovanim upravljanem brojevima ili bazom podataka koja čuva ove identifikatore. UUID se sastoji od 32 heksadecimalna karaktera, koji su organizovani u pet grupa različitih dužina, obično razdvojenih crticama.
- **JSON** (engl. JavaScript Object Notation) je tip podataka koji omogućava čuvanje podataka formatu ključ:vrednost. Obezbeđuje mehanizam za čuvanje složenih podataka proizvoljne strukture u poljima relacione tabele. Često se koristi za razmenu podataka između klijentskih i serverskih aplikacija, skladištenje konfiguracionih podataka, čuvanje podataka u bazama podataka, razmenu podataka putem API-ja (engl. Application Programming Interface) i u mnoge druge svrhe.
- **XML** tip podataka omogućava čuvanje podataka u XML formatu. XML (engl.eXtensible Markup Language) je standardni format za razmenu struktuiranih podataka između različitih sistema i aplikacija. XML tip podataka omogućava skladištenje i manipulaciju XML dokumentima direktno u bazi podataka.
- **GEOMETRY** i **GEOGRAPHY** tipovi podataka se koriste za čuvanje prostornih podataka kao što su tačke, linije, poligoni i drugi geometrijski oblici. Ovi tipovi podataka su korisni u geografskim informacionim sistemima ili aplikacijama koje zahtevaju rad sa prostornim podacima.

6.3 BAZA PODATAKA STUDENSKI SERVIS

Baza podataka za realizaciju funkcionisanja nekog fakulteta je jedan od primera koji se koristi u literaturi o bazama podataka. Za potrebe testiranja SQL upita kreirana je baza podataka STUDENSKI SERVIS koja predstavlja deo šeme relacione baze podataka prikazane ER dijagramima u poglavlju 4 (slika 4.3.7.4, slika 4.3.7.5, slika 4.3.7.6).

Na slici 6.3.1 je prikazana pojava ove baze podataka koja prikazuje podatke korišćene u primerima nadalje.



STUDENT

ID_STUDENTA	IME	PREZIME	SMER	BROJ	GODINA_UPISA
1	Sara	Mijković	NRT	33	2020
2	Marko	Bakić	ASUV	2	2019
3	Andrej	Lazić	NRT	16	2020
4	Lazar	Marković	RT	12	2020
5	Andjela	Stanković	IS	3	2020
6	Miloš	Petrović	RT	25	2019
7	Marko	Rakić	ELITE	6	2020
8	Lena	Milić	IS	10	2018
9	Stefan	Nikolić	NRT	15	2019
10	Aleksandar	Marković	ELITE	11	2019
11	Nikola	Milutinović	ASUV	26	2020
12	Violeta	Grujić	IS	13	2019
13	Bojana	Tomić	NRT	96	2020
14	Aleksa	Todorović	RT	55	2019
15	Igor	Tadić	ELITE	35	2018
16	Petra	Kostić	IS	7	2020
17	Vuk	Mirković	NRT	3	2020
18	Branislav	Vuletić	ASUV	61	2019
19	Nenad	Branković	RT	23	2018
20	Mateja	Stojković	IS	14	2018
21	Jovana	Minić	IS	72	2019
22	Milica	Đukić	NRT	32	2020
23	Filip	Jevtić	RT	29	2018

PREDMET

ID_PREDMETA	ID_PROFESORA	NAZIV	ESPB	STATUS
1259	107	Inženjerska matematika	7	obavezan
1589	117	Programiranje računarskih igara	6	izborni
2458	115	Verovatnoća i statistika	6	izborni
2569	105	Baze podataka	6	obavezan
2648	104	Operativni sistemi 1	6	izborni
3264	114	Objektno programiranje 1	6	izborni
3521	112	Aplikativni softver	6	izborni
4238	118	Digitalne multimedije	6	izborni
4296	110	Elektrotehnika	7	obavezan
4526	101	Uvod u objektno programiranje	6	izborni
4851	119	Programski jezici	6	izborni
5214	102	Veb dizajn	6	izborni
5236	111	Programiranje mobilnih uređaja	6	izborni
6345	116	Mikroračunari	6	izborni
6752	103	Mikrokontroleri	6	izborni
7526	108	Računarske mreže	6	izborni
7584	109	Interakcija čovek-računar	6	izborni
8546	113	Objektno programiranje 2	6	izborni
9412	106	Analogna elektronika	6	izborni
9542	120	Mikroprocesorski softver	6	izborni

PROFESOR

ID_PROFESORA	IME	PREZIME	ZVANJE	DATUM_ZAP
100	Gordana	Vuletić	dr	2002-10-01
101	Miloš	Lazarević	dr	1997-03-15
102	Goran	Protić	mr	1986-09-27
103	Lazar	Gojković	dr	2002-11-05
104	Bojan	Milić	mr	2006-07-22
105	Jelena	Pelić	mr	2014-04-18
106	Marijana	Petrović	dr	1996-08-20
107	Nemanja	Marković	dr	2017-11-30
108	Marko	Bojović	mr	1979-12-25
109	Dragan	Mirčetić	dr	2000-05-16
110	Milovan	Kovačević	dr	2002-07-29
111	Nada	Stanimir…	dr	2011-04-26
112	Vladimir	Vuletić	dr	2015-08-01
113	Radmila	Lekić	dr	2019-05-20
114	Bogdan	Prokić	mr	1983-06-18
115	Tomislav	Miljković	dr	1998-09-03
116	Željko	Jokić	dr	2016-02-12
117	Marija	Vasić	mr	2006-07-22
118	Aleksan…	Grozdić	dr	1974-08-31
119	Miloš	Stanković	dr	1989-06-26
120	Nikola	Žiletić	mr	1993-09-07

STUDENT_PREDMET

ID_STUDENTA	ID_PREDMETA	SKOLSKA_GODINA
3	1259	2018/19
7	1259	2018/19
9	1259	2018/19
16	1259	2018/19
20	1259	2018/19
21	1259	2018/19
22	1259	2018/19
25	1259	2018/19
26	1259	2018/19
28	1259	2018/19
29	1259	2018/19
30	1259	2018/19
3	1589	2019/20
4	1589	2018/19
7	1589	2018/19
11	1589	2018/19
14	1589	2018/19
16	1589	2018/19
20	1589	2018/19
21	1589	2018/19
25	1589	2018/19
28	1589	2018/19

ISPITNI_ROK

ID_ROKA	NAZIV	SKOLSKA_GOD	STATUS_ROKA
1	Januar	2018/19	redovni
2	Februar	2018/19	redovni
3	Jun	2018/19	redovni
4	Septembar	2018/19	redovni
5	Oktobar	2018/19	redovni
6	Oktobar 2	2018/19	vanredni
7	Novembar	2018/19	vanredni
8	Januar	2019/20	redovni
9	Februar	2019/20	redovni
10	Jun	2019/20	redovni
11	Septembar	2019/20	redovni
12	Oktobar	2019/20	redovni
13	Oktobar 2	2019/20	vanredni

ISPIT

ID_ISPITA	ID_ROKA	ID_PREDMETA	DATUM
1	1	2458	2019-01-30
2	2	3264	2019-02-18
3	5	4238	2019-09-17
4	6	4296	2019-10-11
5	7	1259	2019-11-07
6	8	2569	2020-02-01
7	11	9542	2020-09-09
8	13	9412	2020-10-12
9	10	6752	2020-06-16
10	5	5236	2019-09-20
11	6	7584	2019-09-12
12	4	8546	2019-09-06
13	8	4851	2020-02-03
14	7	2648	2019-11-07
15	1	1589	2019-02-01
16	1	1259	2019-02-03
17	2	3521	2019-02-14
18	9	5214	2020-02-18
19	5	4238	2019-09-24
20	3	6752	2019-06-13
21	6	5236	2019-10-13
22	12	7526	2020-09-21

ZAPISNIK

ID_STUDENTA	ID_ISPITA	OCENA	BODOVI
1	3	7	63
1	17	6	51
2	1	10	96
2	31	5	33
3	5	7	62
3	16	9	82
3	17	8	75
3	28	10	91
3	31	6	59
3	33	9	89
4	9	8	77
4	22	10	97
4	25	5	47
4	26	6	56
4	28	6	54
4	30	7	67
4	31	9	87
4	39	9	93
5	11	10	99

Slika 6.3.1 Baza podataka STUDENTSKI SERVIS

Tabela STUDENT čuva podatke o studentima. Za svakog studenta se pamti ime, prezime, smer, broj i godina upisa. Za primarni ključ je dodata kolona ID_STUDENTA koja je autoincrement tip kolone. Kolona autoincrement u SQL-u je tip kolone koji se koristi za automatsko generisanje jedinstvenih vrednosti prilikom unošenja novih redova u tabelu. Ova funkcionalnost omogućava da se automatski generiše sledeća vrednost u koloni, bez potrebe za eksplicitnim navođenjem vrednosti prilikom unošenja podataka. U većini sistema za upravljanje bazama podataka, ovaj tip kolone se naziva "AUTO_INCREMENT" (u MySQL-u) ili "IDENTITY" (u Microsoft SQL Server-u). Kada se postavi autoincrement na određenu kolonu, DBMS automatski dodavaje vrednosti u tu kolonu prilikom unošenja novih redova, povećavajući vrednost za jedan za svaki novi red. U tabeli PROFESOR se pamte podaci o profesorima i to: ime, prezime, zvanje i datum zaposlenja. Kao i u slučaju

tabele STUDENT, primarni ključ je postavljen na dodatu kolonu ID_PROFESORA koja je autoincrement tipa. Tabela PREDMET čuva podatke o nazivu predmeta, broju ESPB bodova, statusu predmeta. Primarni ključ je autoincrement kolona ID_PREDMETA. Ova tabela ima jedan strani ključ ID_PROFESORA koji refrencira na istoimenu kolonu koja primarni ključ u tabeli PROFESOR. U tabeli STUDENT_PREDMET čuvaju se podaci izbora predmeta od strane studenata po školskim godinama. Ova tabela predstavlja realizaciju veze M:N između entiteta STUDENT i PREDMET (poglavlje 4, slika 4.3.7.4). Tabela ima složen primarni ključ koji se sastoji od kolona ID_STUDENTA i ID_PREDMETA i dva strana ključa: ID_STUDENTA refrencira na primarni ključ tabele STUDENT i ID_PREDMETA refrencira na primarni ključ tabele PREDMET. Tabela ISPITNI_ROK beleži podatke o nazivu, šksolskoj godini i statusu ispitnog roka. U tabeli ISPIT se čuvaju podaci o zakazanom ispitu, odnosno datum ispita za određeni predmet u određenom ispitnom roku. Podaci o predmetu i ispitnom roku se preuzimaju preko stranih ključeva ID_PREDMETA i ID_ROKA referenciranjem na istoimene primarne ključeve tabela PREDMET i ISPITNI_ROK, respektivno.

6.4 SQL NAREDBE ZA KREIRANJE, IZMENU I BRISANJE OBJEKATA BAZE PODATAKA

U nastavku teksta definisane su i objašnjene DDL SQL naredbe koje se koriste za kreiranje, izmenu i brisanje objekata unutar baza podataka. Kroz ovaj segment, čitaoci će steći razumevanje osnovnih operacija za upravljanje strukturama podataka u bazi podataka pomoću SQL-a.

6.4.1 CREATE DATABASE

Opšta sintaksa SQL naredbe za kreiranje baze podataka u MySQL sistemu:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] ime_baze
[DEFAULT] { CHARACTER SET [=] character_set_name | COLLATE
[=] collation_name }
```

Izrazi CREATE DATABASE ili CREATE SCHEMA su ekvivalentni i koriste se za kreiranje nove baze podataka ili šeme. Upotreba IF NOT EXISTS izraza osigurava da se baza podataka neće kreirati ako već postoji objekat pod tim imenom. Opcija DEFAULT je opcionalna klauzula koja omogućuje postavljanje zadatih postavki za novu bazu podataka. CHARACTER SET [=] character_set_name postavlja znakovni skup (character set) za novu bazu podataka. Znakovni skup određuje karaktere koji se mogu u tekstualnim podacima baze podataka. COLLATE [=] collation_name postavlja način sortiranja i poređenja tekstualnih podataka u bazi podataka.

Za primer koji se obrađuje, SQL naredba za kreiranje baze podataka *studentski_servis* je prikazana u nastavku.

```
CREATE DATABASE studentski_servis
CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
```

Implementacijom prethodno navedene SQL naredbe kreirana je nova baza podataka *studentski_servis* sa postavkama karakterističnog skupa *utf8mb4* i kolacije *utf8mb4_0900_ai_ci*. Ovo osigurava da nova baza podataka podržava pun opseg Unicode znakove i ima odgovarajuće podešenu kolaciju za pravilno sortiranje i poređenje teksta uzimajući u obzir različite jezičke specifičnosti i osiguravajući tačno sortiranje.

6.4.2 USE NAREDBA

Pre nego se započne rad s bazom podataka, bilo da je u pitanju kreiranje određenih objekata poput tabela, pogleda, indeksa i slično ili upravljanje podacima, važno je postaviti odgovarajuću bazu podataka kao aktivnu.

Sintaksa SQL naredbe za aktiviranje baze podataka za upotrebu u daljem radu je:

```
USE ime_baze;
```

U slučaju primera koji se obrađuje za aktiviranje baze studentski_servis koristi se naredba u nastavku.

```
USE studentski_servis;
```

Ukoliko izbrana baza podataka postoji, sistem ispisuje sledeću poruku:

```
0 row(s) affected
```

Ako na serveru ne postoji baza podataka navedena u USE naredbi, sistem ispisuje odgovarajuću grešku.

```
Error Code: 1049. Unknown database 'studentski_s'
```

6.4.3 CREATE TABLE

Osnovni element u svakoj relacionoj bazi podataka jeste tabela. Kreiranje tabele obavlja se korišćenjem SQL naredbe CREATE TABLE.

Sintaksa naredbe za kreiranje tabele u MySQL sistemu je:

```
CREATE TABLE [IF NOT EXISTS] ime_tabele
(naziv_kolone tip_podataka [NOT NULL | NULL]
  DEFAULT vrednost] [CHECK (uslov)] [AUTO_INCREMENT],
   PRIMARY KEY (naziv_kolone)],
   KEY (naziv_kolone)],
   CONSTRAINT ime_konstrainta FOREIGN KEY (naziv_kolone)
   REFERENCES ime_referencirane_tabele(ime_primarnog_ključa)]
);
```

S obzirom da SQL standard i različiti DBMS-ovi obezbeđuju dodatne klauzule koje omogućavaju preciznu kontrolu procesa kreiranja tabele, prethodno navedena sintaksa naredbe CREATE TABLE nije potpuna. Generalno, deklaracija kolone uključuje ime kolone, tip podatka kolone, opcionalnu dužinu (ako je potrebno, zavisno od tipa podatka) i opcionalna ograničenja za kolonu. Deklaracije kolona su razdvojene zarezima.

Nakon deklaracije svih kolona, navode se ograničenja koja važe za celu tabelu. Ograničenja za kolonu su opcionalna i mogu sadržati specifikaciju predefinisanih vrednosti i niz specifikacija ograničenja. Dodatne klauzule i specifikacije mogu se koristiti u CREATE TABLE naredbi kako bi se prilagodili specifičnim potrebama i zahtevima sistema za koji se kreira baza podataka.

Pored definisanja tipa podataka za svaku kolonu, prilikom kreiranja tabele postavljaju se i određena ograničenja. Ograničenja koja se postavljaju nad kolonama su:

- NULL ili NOT NULL – obezbeđuje da kolona može ili ne može imatu NULL vrednosti.
- UNIQUE – obezbeđuje da kolona ima jedinstvene vrednosti (kandidati za ključeve).
- PRIMARY KEY – postavlja kolonu za primarni ključ tabele (definiše se nad jednom kolonom ili kao kompozitni složeni ključ nad više kolona).
- CHECK – postavlja se ograničenje za proveru vrednosti kolone (koristi se kod upisa ili ažuriranja vrednosti).
- DEFAULT – navodi se podrazumevana vrednost za kolonu (kolona uzima ovu vrednost, ako vrednost kolone nije navedena).
- REFERENCES – definiše se da kolona predstavlja spoljni ključ tabele.

Postavljanjem *NULL* ograničenja omogućava se da vrednost kolone može biti nepoznata ili neupisana. *NOT NULL* ograničenje, s druge strane, zabranjuje prazne vrednosti u poljima kolone na koju je primenjeno ovo ograničenje. Ako ništa nije eksplicitno navedeno, podrazumevana postavka je da vrednost može biti *NULL*.

UNIQUE ograničenje garantuje jedinstvenost vrednosti u koloni. Dva reda ne mogu imati istu vrednost za kolonu koja ima postavljeno *UNIQUE* ograničenje. Ovo ograničenje je strožije u odnosu na *NOT NULL*. Kolone na koje je primenjeno *UNIQUE* ograničenje su potencijalni kandidati za primarni ključ. *PRIMARY KEY* ograničenje definiše kolonu ili kombinaciju kolona koja jedinstveno identifikuje svaki red u tabeli. Podrazumevano, kolone označene kao *PRIMARY KEY* su automatski *NOT NULL* i *UNIQUE*. Prilikom kreiranja tabele, deklaracija stranog ključa podrazumeva definisanje *REFERENCES* ograničenja. Navodi se ime tabele koja se referencira, zajedno sa opcionalnim imenom kolone koja predstavlja primarni ključ referencirane tabele.

Bitno je napomenuti da postavljanje stranog ključa prilikom kreiranja tabele implicira da moraju biti dodati zapisi podataka u referenciranu tabelu pre dodavanja podataka u tabelu koja sadrži strani ključ. Ovim se osigurava postojanje mogućih vrednosti koje može imati deklarisani strani ključ tabele na koju se referiše.

U nastavku će biti naveden postupak kreiranja tabela baze podataka *studentski_servis*. Ključevi, indeksi i referenciranje će biti definisano naknadno.

Tabela student

```
CREATE TABLE student (
 ID_STUDENTA int(11) NOT NULL,
 IME varchar(25) NOT NULL,
 PREZIME varchar(40) NOT NULL,
 SMER varchar(5) NOT NULL,
 BROJ int(11) NOT NULL,
 GODINA_UPISA varchar(4) NOT NULL,
 PRIMARY KEY (ID_STUDENTA)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela student sa kolonama:

- ID_STUDENTA je numeričkog celobrojnog tipa INT(11) kao tipa INTEGER sa širinom od 11 znakova (ova širina ne utiče na raspon vrednosti koje kolona može sadržavati, već samo na prikazanu širinu prilikom prikaza rezultata upita.), ne može ostati nepopunjen (NOT NULL), postavljen je za primarni ključ tabele (PRIMARY KEY (ID_STUDENTA));
- IME je tekstualnog tipa VARCHAR , kapaciteta od najviše 25 karaktera i ne može ostati nepopunjeno;
- PREZIME je tekstualnog tipa VARCHAR, kapaciteta od najviše 40 karaktera i ne može ostati nepopunjeno;
- SMER je tekstualnog tipa VARCHAR, kapaciteta od najviše 5 karaktera i ne može ostati nepopunjeno;
- BROJ je numeričkog celobrojnog tipa INT(11) i ne može ostati nepopunjeno;
- GODINA_UPISA je tekstualnog tipa VARCHAR, kapaciteta od najviše 4 karaktera i ne može ostati nepopunjeno;

Tabela profesor

```
CREATE TABLE profesor (
  ID_PROFESORA int(11) NOT NULL,
  IME varchar(25) NOT NULL,
  PREZIME varchar(50) NOT NULL,
  ZVANJE enum('dr','mr') NOT NULL,
  DATUM_ZAP date NOT NULL,
  PRIMARY KEY (ID_PROFESORA)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela profesor sa kolonama:

- ID_ PROFESORA je numeričkog celobrojnog tipa (INT(11)), ne može ostati nepopunjen (NOT NULL), postavljen je za primarni ključ tabele (PRIMARY KEY (ID_PROFESORA));
- IME je tekstualnog tipa VARCHAR, kapaciteta od najviše 25 karaktera i ne može ostati nepopunjeno;
- PREZIME je tekstualnog tip VARCHAR, kapaciteta od najviše 40 karaktera i ne može ostati nepopunjeno;
- ZVANJE je tipa ENUM, sa mogućim vrednostima 'dr' (doktor) ili 'mr' (magistar) i ne može ostati nepopunjeno;
- DATUM_ZAP je datumskog tipa DATE i ne može ostati nepopunjeno;

Tabela predmet

```
CREATE TABLE predmet (
  ID_PREDMETA int(11) NOT NULL,
  ID_PROFESORA int(11) NOT NULL,
  NAZIV varchar(50) NOT NULL,
  ESPB int(11) NOT NULL,
  STATUS enum('obavezan','izborni') NOT NULL,
  PRIMARY KEY (ID_PREDMETA)
 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela predmet sa kolonama:

- ID_PREDMETA je numeričkog celobrojnog tipa (INT(11)), ne može ostati nepopunjen (NOT NULL), postavljen je za primarni ključ tabele (PRIMARY KEY (ID_PREDMETA));
- ID_PROFESORA je numeričkog celobrojnog tipa INT(11) i ne može ostati nepopunjeno;
- NAZIV je tekstualnog tipa VARCHAR, kapaciteta od najviše 50 karaktera i ne može ostati nepopunjeno;
- ESPB je numeričkog celobrojnog tipa INT(11) i ne može ostati nepopunjeno;
- STATUS je tipa ENUM, sa mogućim vrednostima 'obavezan', 'izborni' i ne može ostati nepopunjeno;

Tabela student_predmet

```
CREATE TABLE student_predmet (
  ID_STUDENTA int(11) NOT NULL,
  ID_PREDMETA int(11) NOT NULL,
  SKOLSKA_GODINA varchar(7) NOT NULL,
  PRIMARY KEY (ID_STUDENTA,ID_PREDMETA,SKOLSKA_GODINA)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela student_predmet sa kolonama:

- ID_STUDENTA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, deo složenog primarnog ključa tabele (PRIMARY KEY (ID_STUDENTA,ID_PREDMETA,SKOLSKA_GODINA));
- ID_PREDMETA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, deo složenog primarnog ključa tabele (PRIMARY KEY (ID_STUDENTA,ID_PREDMETA,SKOLSKA_GODINA));
- SKOLSKA_GODINA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, deo složenog primarnog ključa tabele (PRIMARY KEY (ID_STUDENTA,ID_PREDMETA,SKOLSKA_GODINA));

Tabela ispitni_rok

```
CREATE TABLE ispitni_rok(
  ID_ROKA int(11) NOT NULL AUTO_INCREMENT,
  NAZIV varchar(15) NOT NULL,
  SKOLSKA_GOD varchar(7) NOT NULL,
  STATUS_ROKA enum('redovni','vanredni') NOT NULL,
  PRIMARY KEY (ID_ROKA)
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela ispitni_rok sa kolonama:

- ID_ROKA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, postavljen je za primarni ključ tabele (PRIMARY KEY (ID_ROKA)); AUTO_INCREMENT označava da se vrednost ove kolone automatski inkrementira prilikom dodavanja novog reda u tabeli. Ovo osigurava jedinstvenost vrednosti u koloni i automatsko dodeljivanje sledeće dostupne vrednosti.
- NAZIV je tekstualnog tipa VARCHAR, kapaciteta od najviše 15 karaktera i ne može ostati nepopunjeno;
- SKOLSKA_GOD je tekstualnog tipa VARCHAR, kapaciteta od najviše 7 karaktera i ne može ostati nepopunjeno;
- STATUS_ROKA je tipa ENUM, sa mogućim vrednostima 'redovni', 'vanredni' i ne može ostati nepopunjeno;

Tabela ispit

```
CREATE TABLE ispit (
 ID_ISPITA int(11) NOT NULL AUTO_INCREMENT,
 ID_ROKA int(11) NOT NULL,
 ID_PREDMETA int(11) NOT NULL,
 DATUM date NOT NULL,
 PRIMARY KEY (ID_ISPITA)
) ENGINE=InnoDB AUTO_INCREMENT=41 DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela ispit sa kolonama:

- ID_ISPITA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, postavljen je za primarni ključ tabele (PRIMARY KEY (ID_ISPITA)); AUTO_INCREMENT označava da se vrednost ove kolone automatski inkrementira prilikom dodavanja novog reda u tabeli. Ovo osigurava jedinstvenost vrednosti u koloni i automatsko dodeljivanje sledeće dostupne vrednosti.
- ID_ROKA je numeričkog celobrojnog tipa INT(11) i ne može ostati nepopunjeno;
- ID_PREDMETA je numeričkog celobrojnog tipa INT(11) i ne može ostati nepopunjeno;
- DATUM je datumskog tipa DATE i ne može ostati nepopunjeno;

Tabela zapisnik

```
CREATE TABLE zapisnik (
 ID_STUDENTA int(11) NOT NULL,
 ID_ISPITA int(11) NOT NULL,
 OCENA int(11) NOT NULL,
 BODOVI varchar(3) NOT NULL,
 PRIMARY KEY (ID_STUDENTA,ID_ISPITA)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Kreirana je tabela zapisnik sa kolonama:

- ID_STUDENTA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, deo je složenog primarnog ključa tabele (PRIMARY KEY (ID_STUDENTA, ID_ISPITA));
- ID_ISPITA je numeričkog celobrojnog tipa INT(11), ne može ostati nepopunjeno, deo složenog primarnog ključa tabele (PRIMARY KEY (ID_STUDENTA, ID_ISPITA));
- OCENA je numeričkog celobrojnog tipa INT(11) i ne može ostati nepopunjeno;
- BODOVI je tekstualnog tipa VARCHAR, kapaciteta od najviše 3 karaktera i ne može ostati nepopunjeno;

6.4.4 ALTER TABLE

SQL naredba ALTER TABLE se koristi za izmenu strukture postojeće tabele u bazi podataka.

Primeri implementacije ALTER TABLE naredbi uključuju:

- Dodavanje nove kolone: ADD COLUMN
- Izmenu postojeće kolone: MODIFY COLUMN ili CHANGE COLUMN
- Brisanje kolone: DROP COLUMN
- Dodavanje ograničenja: ADD CONSTRAINT
- Izmena primarnog ključa: ADD PRIMARY KEY, DROP PRIMARY KEY
- Izmena stranog ključa: ADD FOREIGN KEY, DROP FOREIGN KEY

Sintaksa naredbe ALTER TABLE

```
ALTER TABLE ime_tabele
ADD [COLUMN] naziv_kolone tip_podataka [nakon kolone],
DROP [COLUMN] naziv_kolone,
MODIFY [COLUMN] naziv_kolone tip_podataka [nakon kolone],
CHANGE [COLUMN] staro_ime_kolone novo_ime_kolone
tip_podataka [nakon_kolone],
ADD [CONSTRAINT] ogranicenje,
DROP [CONSTRAINT] ogranicenje,
ADD [INDEX|KEY] indeks (kolona [tip_indeksa]),
DROP [INDEX|KEY] indeks,
DISABLE KEYS,
ENABLE KEYS,
RENAME [TO] novo_ime_tabele,
ORDER BY kolona [ASC|DESC]
```

Primer: SQL naredba koja u tabelu STUDENT dodaje novu kolonu.

```
ALTER TABLE student
ADD COLUMN JMBG BIGINT;
```

Kod dodavanja nove kolone NOT NULL ograničenja se postavlja naknadno. Ukoliko bi se ograničenje definisalo prilikom primene ALTER TABLE naredbe za dodavanje nove kolone, isto bi bilo narušeno.

Primer: SQL naredba koja modifikuje kolonu i uvodi NOT NULL ograničenje nad kolonom.

```
ALTER TABLE student
MODIFY COLUMN JMBG BIGINT NOT NULL;
```

Promena tipa neke kolone zahteva detaljnu analizu postojećih podataka. DBMS će automatski pokušati konverziju postojećih podataka ukoliko je to moguće. Ukoliko nije moguće prikazuje poruku o grešci.

Primer: SQL naredba koja u tabelu dodaje novo ograničenje kojim se obezbeđuje da se u kolonu JMBG mora uneti trinaest cifara.

```
ALTER TABLE student
ADD CONSTRAINT jmbg_duzina CHECK (LENGTH(JMBG) = 13);
```

Primer: SQL naredba kojom se iz tabele STUDENT briše ograničenje.

```
ALTER TABLE student
DROP CONSTRAINT jmbg_duzina;
```

Primer: SQL naredba koja briše kolonu iz tabele.

```
ALTER TABLE student
DROP COLUMN JMBG;
```

6.4.5 FOREIGN KEY NAREDBA

Prilikom kreiranja tabela baze podataka, kao što smo videli i slučaju baze podataka studentski_servis, u određenim tabelama su definisane kolone iz drugih tabela. Na ovaj način je izvršena priprema za realizaciju veze 1:N primenom ograničenja stranog ključa. Na primer, kolona ID_PROFESORA je predviđena za strani ključ u tabeli PREDMET za realizaciju reference na tabelu PROFESOR Na ovaj način su postavljaju implicitne relacije koje ne obezbeđuju očuvanje integriteta podataka, jer ne postoje provere koje bi bile postavljene ograničenjima nad kolonama stranih ključeva u tabeli.

Primer: SQL naredbs za postavljanje ograničenja stranog ključa u tabeli PREDMET.

```
ALTER TABLE predmet
ADD CONSTRAINT FK_PREDMET_RELATIONS_PROFESOR
FOREIGN KEY(ID_PROFESORA)REFERENCES PROFESOR (ID_PROFESORA)
ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Realizacijom prethodno navedene SQL naredbe tabela PREDMET će biti izmenjena. Dodaje se ograničenje FK_PREDMET_RELATIONS_PROFESOR. Relacioni

DBMS automatski kreira indeks nad kolonom ID_PROFESORA koja predstavlja strani ključ, koji predstavlja referenciru na primarni ključ tabele PROFESOR.

Ograničenje stranog ključa predstavlja ključni mehanizam u bazi podataka koji omogućava precizno definisanje ponašanja vezanih za operacije brisanja i ažuriranja zapisa. Postavke ovih klauzula mogu biti od značaja za očuvanje integriteta podataka i usklađivanje veza između različitih tabela.

- ON DELETE RESTRICT: Ova klauzula onemogućava brisanje zapisa iz tabele PROFESOR ako u tabeli PREDMET postoji zapis čija vrednost u koloni ID_PROFESORA referencira na primarni ključ u tabeli PROFESOR. Drugim rečima, sprečava brisanje profesora koji je povezan sa predmetom.
- ON DELETE CASCADE: Kada se koristi ova klauzula, brisanjem određenih zapisa iz tabele PROFESOR, automatski se brišu i svi zapisi iz tabele PREDMET koji u koloni ID_PROFESORA referenciraju na vrednosti primarnog ključa iz tabele PROFESOR. To omogućava automatsko održavanje konsistentnosti baze podataka.
- ON UPDATE CASCADE: U slučaju da se vrednost primarnog ključa u tabeli PROFESOR ažurira, ova klauzula obezbeđuje da se iste promene reflektuju i na sve zapise u tabeli PREDMET koji u koloni ID_PROFESORA referenciraju na ažurirane vrednosti primarnog ključa iz tabele PROFESOR.
- ON UPDATE RESTRICT: Korišćenjem ove klauzule, onemogućava se ažuriranje vrednosti primarnog ključa u tabeli PROFESOR ako postoji zapis u tabeli PREDMET koji u koloni ID_PROFESORA referencira na te vrednosti. Ovo ograničenje sprečava ažuriranje zapisa podataka o profesorima koji su već povezani sa određenim predmetima.

Prethodno navedene klauzule omogućavaju precizno upravljanje referencijalnim integritetom u bazi podataka i obezbeđuju konzistentnost podataka prilikom izvršavanja operacija brisanja i ažuriranja.

SQL naredbama za postavljanje stranog ključa mogu da budu napravljene sve ostale reference predviđene primerom modela baze podataka studentski_servis.

U nastavku je dat SQL kod kojim su izmenjene tabele student_predmet, zapisnik, ispit. Izmene se odnose na postavljanje veza jedan prema više koje su realizovane dodavanjem ograničenja stranog ključa i posatvljanjem odgovarajućih klauzula za očuvanje referncijalnog integriteta.

```
ALTER TABLE student_predmet
ADD CONSTRAINT FK_STUDENT__RELATIONS_STUDENT
FOREIGN KEY (ID_STUDENTA) REFERENCES STUDENT (ID_STUDENTA)
ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE student_predmet
ADD CONSTRAINT FK_STUDENT__RELATIONS_PREDMET
FOREIGN KEY (ID_PREDMETA)REFERENCES PREDMET (ID_PREDMETA)
ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE zapisnik
ADD CONSTRAINT FK_ZAPISNIK_RELATIONS_STUDENT
FOREIGN KEY (ID_STUDENTA) REFERENCES STUDENT (ID_STUDENTA)
ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE zapisnik
ADD CONSTRAINT FK_ZAPISNIK_RELATIONS_ISPITI
FOREIGN KEY (ID_ISPITA)REFERENCES ISPIT (ID_ISPITA)
ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE ispit
ADD CONSTRAINT FK_ISPITI_RELATIONS_ISPITNI_
FOREIGN KEY (ID_ROKA)REFERENCES ISPITNI_ROK (ID_ROKA)
ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE ispit
ADD CONSTRAINT FK_ISPITI_RELATIONS_PREDMET
FOREIGN KEY (ID_PREDMETA)REFERENCES PREDMET(ID_PREDMETA)
ON DELETE RESTRICT ON UPDATE RESTRICT;
```

6.5 UPRAVLJANJE I MANIPULACIJA PODACIMA

Na početku ovog poglavlja je navedeno da DML obuhvata naredbe za rukovanje podacima koje omogućavaju izveštavanje iz baze podataka i ažuriranje u širem smislu. Izveštavanje iz baze podataka podrazumeva pristup podacima, prikaz pronađenih ili izračunatih sadržaja iz baze podataka. Ažuriranje baze podataka odnosi se na dodavanje novih, kao i brisanje i izmenu postojećih vrednosti podataka.

6.5.1 SELECT NAREDBA

Prikaz podataka je najčešća aktivnost u relacionoj bazi podataka. Ostvaruje se primenom SELECT naredbe koja se smatra ključnom komponentom SQL jezika. Definišu se postavke rezultujućeg prikaza, pa se SELECT naredba može smatrati deklarativnom. To znači da se definiše šta se želi prikazati od podataka, a ne postupak kako se to postiže.

Sistem za upravljanje relacionim bazama podataka je odgovoran za izbor najefikasnijeg načina za dobijanje željenih podataka. SELECT naredba koristi

podatke iz jedne ili više tabela, manipuliše tim podacima, i generiše rezultujući prikaz u obliku tabele. Čak i kada je rezultat izvršenja skalarni podatak, tretira se kao tabela sa jednim redom i jednom kolonom.

Opšta sintaksa SELECT naredbe

```
SELECT ([ALL | DISTINCT] <lista_kolona1>) | *
FROM <lista_tabela>
[WHERE <lista_uslova1>]
[GROUP BY <lista_kolona2>]
[HAVING <lista_uslova2>]
[ORDER BY <lista_kolona3>]
```

U nastavku pojasnićemo značenje osnovnih termina SQL programiranja.

Ključna reč ukazuje na poseban SQL iskaz. Opšta sintaksa uključuje veći broj ključnih reči:

- SELECT – definiše listu kolona koje će biti uključene u rezultujuću tabelu
- FROM – koristi se za definisanje tabela iz kojih se pribavljaju podaci za generisanje rezultujuće tabele. Može uključivati i JOIN klauzule kako bi se povezale više tabela na osnovu određenih uslova.
- WHERE – koristi se za postavljanje uslova na osnovu kojih se ograničava broj redova koji će biti uključeni u rezultujuću tabelu. Eliminiše iz prikaza redove za koje postavljen uslov ne vraća vrednost TRUE.
- GROUP BY – grupiše redove tabele na osnovu vrednosti određene kolone ili kolona.
- HAVING – definiše uslov za prethodno postavljeno grupisanje upotrebom klauzule GROUP BY. Omogućava preciznije filtriranje rezultata grupisanih po određenim kriterijumima.
- ORDER BY – koristi se za sortiranje rezultata po jednoj ili više kolona u određenom redosledu. Navode se kolone po kojima se vrši sortiranje kao i redosled sortiranja (rastući ili opadajući).

Klauzula je deo SQL rečenice. Neki od primera klauzula su:

- SELECT ID_STUDENTA, IME, PREZIME
- WHERE SMER=’RT’
- FROM student
- ORDER BY GODINA_UPISA

Rečenica je jedna klauzula ili kombinacija dve ili više klauzula, kojom se izvršava neka operacija nad bazom podataka. Završava se znakom ;

- SELECT ime FROM student;
- SELECT * FROM student;
- SELECT id_studenta, ime, prezime FROM student ORDER BY smer;

U najjednostavnijem obliku, SELECT rečenica mora da sadrži SELECT klauzulu kojom se specificiraju kolone za prikaz i FROM klauzulu za definisanje tabela kojima pripada kolona/kolone navedene u SELECT klauzuli. Ukoliko se navede više tabela potrebno je specificirati način spajanja tabela. U ovom poglavlju primeri su ograničeni na prikaz podataka iz jedne tabele. U okviru SELECT klauzule primenjuju se neke od opcija: *ALL, DISTINCT, *, izraz, AS naziv_alijasa*.

Upotreba DISTINCT opcije nakon ključne reči SELECT nije strogo definisana i može se koristiti prema potrebi. Na ovaj načini serveru baze podataka se prosleđuje zahtev za pribavljanje i prikaz jedinstvenih redova u rezultujućoj tabeli. Drugim rečima, ako postoje redovi sa istim vrednostima u svim kolonama koje su uključene u upit, samo će se jedan od njih zadržati u rezultatu.

Za prikaz vrednosti svih kolona iz tabele specificirane u FROM klauzuli upita, koristi se simbol "*" (zvezdica). Alternativno, možemo navesti tačan popis svih imena polja (kolona) iz različitih tabela čije vrednosti želimo u rezultujućoj tabeli. Ukoliko nije navedena WHERE klauzula, takođe biće prikazan i svaki red.

Generalno je lakše koristiti SELECT *. Međutim, prilikom pisanja programske logike za aplikaciju, preporučljivo je navođenje svake kolone posebno. S druge strane, klauzula SELECT * značajno utiče na optimizaciju i performanse upita. Navođenje samo kolona koje su potrebne, umesto korišćenja zvezdice, ubrzaće upit, smanjiće veličinu rezultujuće tabele, kao i opterećenje mreže. Problemi sa SELECT * mogu nastati i ako je vaš upit unutar koda, a program dobije drugačiji set kolona iz upita nego što se očekivalo.

Primer: Prikazati podatke o studentima. U ovom primeru rezultujuća tabela sadrži sve kolone iz specificirane tabele u FROM klauzuli.

```
SELECT *
FROM student;
```

Isti sadržaj rezultujuće tabele možemo dobiti navođenjem imena svih kolona u SELECT klauzuli.

```
SELECT ID_STUDENTA, IME, PREZIME, SMER, BROJ, GODINA_UPISA
FROM student;
```

Primer: Prikazati podatke o imenu i prezimenu studenata. U nastavku je dat SQL upit koji omogućava prikaz samo podataka o prezimenima i imenima studenata.

```
SELECT PREZIME,IME
FROM student;
```

Redosled kolona u rezultujućoj tabeli je definisan redosledom navođenja kolona u SELECT klauzuli. Aleternativno prethodnom primeru, u nastavku je dat SQL upit koji prikazuje iste podatke o imenima i prezimena svih studenata ali u drugačijem redosledu.

```
SELECT IME, PREZIME
FROM student;
```

Iz prethodnih primera može se zaključiti da se SELECT rečenica koristi za prikaz vrednosti određenih ili svih kolona neke tabele, pri čemu može da se desi da se neke vrednosti ponavljaju.

Ako je potrebno prikazati samo različite vredosti, a ne i koliko puta se pojavljuju, u SELECT klauzuli koristi se ključna reč DISTINCT.

Primer: Prikazati, bez ponavljanja, skraćenice studijskih programa na kojima ima upisanih studenata. U nastavku je dat upit sa upotrebom DISTINCT ključne reči u SELECT klauzuli koji omogućava prikaz jedinstvenih podataka bez ponavljanja.

```
SELECT DISTINCT SMER
FROM student;
```

6.5.2 WHERE

Izdvajanje podataka iz baze se može ograničiti na skup redova koji će biti prikazani. Ovo se postiže postavljanjem uslova u WHERE klauzuli. WHERE klauzula sadrži uslov koji mora da se ispuni i navodi se odmah posle FROM klauzule u SQL rečenici.

U rezultujuću tabelu će biti uključeni samo redovi koje zadovoljavaju postavljen uslov, odnosno za koje uslov ima vrednost TRUE.

Operandi uslova mogu biti kolone specificirane tabele u FROM klauzuli ili ugnježdena SELECT naredba i operatori poređenja. Neki operatori porede navedenu kolonu ili izraz sa jednom vrednošću (jednoredni operatori), a drugi porede sa skupom ili intervalom vrednosti (višeredni operatori).

Pregled operatora, opis i primer upotrebe je prikazan u tabeli 6.5.2.1.

Tabela 6.5.2.1. Pregled operatora, opis i primer upotrebe

Operator	Opis operatora	Primer upotrebe
=, <, >, <>, <=, >=	Relacioni operatori	ime="Lazar" ocena> 5
AND,OR, NOT	Logički operatori	ocena=10 AND (naziv_predmeta ="Baze podataka" OR naziv_predmeta= "Nerelacione baze podataka")
LIKE	Omogućava pronalaženje traženog teksta u nekom podatku znakovnog tipa.	ime LIKE 'M%'; ime LIKE '%M' ime LIKE '%M%'; Ime LIKE '_'
BETWEEN	Ispituje da li se vrednost datog izraza nalazi između dve zadate vrednosti uključujući i zadate vrednosti	ESPB BETWEEN 60 AND 120
IN	Ispituje da li se vrednost datog izraza nalazi u specificiranoj listi	ocena IN (8,9,10)
IS NULL	Ispituje postojanje polja bez unetih vrednosti u navedenoj kolonu odnosno omogućava proveru da li kolona ima NULL vrednost.	ocena IS NULL

Primer: Prikazati nazive obaveznih predmeta koji imaju više od 6 ESPB bodova.

```
SELECT NAZIV
FROM predmet
WHERE STATUS ='obavezan' AND ESPB>6
```

SQL upit prikazuje kombinovanu upotrebu različitih kategorija operatora. U primeru, za navedeni uslov u WHERE klauzuli su korišćeni relacioni (=) i logički operator AND.

Primer: Prikazati podatke o profesorima koji su doktori nauka ili im ime počinje slovom G.

```
SELECT *
FROM profesor
WHERE IME LIKE 'g%' OR ZVANJE='dr';
```

Iz prethodnih primera se može videti sa se logički operatori AND i OR koriste na standardni način.

Osnovna funkcija NOT operatora je negiranje uslova u upitu. Često se koristi zajedno sa drugim logičkim operatorima kao što su AND, OR, i operatorima poređenja kao što su =, !=, <, >, itd.

Postavlja se ispred operatora odnosno specificiranog uslova.

```
SELECT *
FROM profesor
WHERE IME NOT LIKE 'g%' OR NOT ZVANJE = 'dr';
```

Može se takođe koristiti i za negiranje celog uslova.

```
SELECT *
FROM profesor
WHERE NOT (IME LIKE 'g%' OR PREZIME LIKE 'p%');
```

Operator LIKE u SQL-u koristi se za pretragu i filtriranje tekstualnih podataka na osnovu postavljenih obrazaca (šablona). LIKE operator koristi simbole procenat (%) i donja crta (_) za predstavljanje različitih znakova ili skupova znakova u šablonu pretrage. Simbol procenat (%) zamenjuje nula ili više znakova bilo koje vrste. Na primer, ako koristimo šablon "abc%", to će pronaći sve vrednosti koje počinju sa "abc" i imaju bilo koju kombinaciju znakova nakon toga. Primer "a%b" pronaći će sve vrednosti koje počinju sa "a" i završavaju sa "b", bez obzira na ostatak teksta između. Simbol donja crta (_) zamenjuje tačno jedan znak. Na primer, ako koristimo šablon "a_c", to će pronaći sve vrednosti koje počinju sa "a", zatim sledi bilo koji jedan znak, a zatim sledi "c". Ovi simboli mogu se kombinovati za složenije pretrage. Operator NOT LIKE se koristi za pronalaženje vrednosti koje ne odgovaraju zadatom šablonu.

Napomena: MySQL kod tekstualnih podataka ne pravi razliku između malih i velikih slova.

Primer: Prikazati prezime, ime i studijski program za studente koji su se upisali na studijski program RT ili IS.

```
SELECT PREZIME, IME, SMER
FROM student
WHERE SMER IN ('RT','IS');
```

Operator IN je skupovni operator. Omogućava navođenje skupa vrednosti koje kolona može da ima kako bi uslov naveden u WHERE klauzuli bio ispunjen.

Upotreba IN umesto OR operatora preporučuje se u slučaju kombinovanja više uslova koji se odnose na istu kolonu. Na primer, u prethodno navedenom SQL upitu mogli

smo koristiti OR operator za kombinovanje dva uslova, ali je efikasnije koristiti IN operator.

```
SELECT PREZIME, IME, SMER
FROM student
WHERE SMER ='RT' OR SMER ='IS';
```

Primer: Prikazati prezime, ime i godinu upisa za studente koji su se upisali između 2017. i 2019. uključujući i te godine.

```
SELECT PREZIME,IME, GODINA_UPISA
FROM student
WHERE GODINA_UPISA BETWEEN 2017 AND 2019;
```

	PREZIME	IME	GODINA_UPISA
▶	Bakić	Marko	2019
	Petrović	Miloš	2019
	Milić	Lena	2018
	Nikolić	Stefan	2019
	Marković	Aleksandar	2019
	Grujić	Violeta	2019
	Todorović	Aleksa	2019
	Tadić	Igor	2018
	Vuletić	Branislav	2019
	Branković	Nenad	2018
	Stojković	Mateja	2018
	Minić	Jovana	2019
	Jevtić	Filip	2018
	Popović	Dragan	2019
	Marković	Željko	2018
	Lekić	Janko	2019
	Pantić	Todor	2019

Slika 6.5.2.1 Rezultat primene BETWEEN AND operatora

Rezultujuća tabela je prikazana na slici 6.5.2.1.

Prethodni primer je prikazao upotrebu BETWEEN operatora. Ovaj operator menja upotrebu operatora AND, >= i <=.

Omogućava ispitivanje da li je vrednost navedene kolone unutar postavljenog opsega, uključujući i granice tog opsega.

SQL upit se može napisati upotrebom AND operatora. Vraća isti rezultat kao i sa upotrebom BETWEEN operatora i ima sledeći oblik:

```
SELECT PREZIME, IME, GODINA_UPISA
FROM student
WHERE GODINA_UPISA >= 2017 AND GODINA_UPISA <=2019;
```

Primer: Prikazati prezime, ime studenata za koje nije unet podataka o JMBG-u.

```
SELECT PREZIME, IME
FROM student
WHERE JMBG IS NULL;
```

Upotreba IS NOT NULL operatora, suprotno od IS NULL, omogućava izdvajanje skupa redova zapisa koji u ispitivanoj koloni imaju upisanu vrednost.

```
SELECT PREZIME, IME
FROM student
WHERE JMBG IS NOT NULL;
```

Provera NULL vrednosti se odnosi na ispitivanje da li kolona sadrži ili ne NULL vrednost. S obzirom da ne podržavaju rad sa relacionim operatorima, NULL vrednosti se ne mogu navoditi u uslovima navedenim u WHERE klauzuli, a u kojima se vrši njihovo poređenje sa vrednostima neke specificiranim kolone.

SQL upit naveden u nastavku neće vratiti nijedan zapis u rezultujućoj tabeli, a MySQL DBMS neće prijaviti grešku u ovom slučaju.

```
SELECT PREZIME, IME
FROM student
WHERE JMBG = NULL;
```

6.5.3 ORDER BY

Klauzula ORDER BY omogućava prilagođavanje izgleda rezultata upita odnosno specificiranje prikaza redova rezultujuće tabele sortiranjem po vrednostima jedne ili više kolone u rastućem (engl. *ascending, ASC*, podrazumevana vrednost) ili opadajućem redosledu (engl.*descending, DESC*).

Ukoliko se klauzula ORDER BY ne navede u SELECT rečenici, rezultujući redovi se prikazuju po slučajnom principu i ne postoji nikakva garancije da će isti upit uvek generisati isto sortiran prikaz.

Primer: Prikazati prezime, ime studenta i studijski program na koji upisan. Prikaz sortirati po prezimenu studenta u rastućem redosledu.

```
SELECT PREZIME, IME, SMER
FROM student
ORDER BY PREZIME ASC;
```

Rastući redosled je podrazumevana vrednost pa se ne mora navoditi u ORDER BY klauzuli. U slučaju prikaza rezultata sortiranih po prezimenu u opadajućem redosledu u SQL upitu se eksplicitno mora navesti parametar DESC.

```
SELECT PREZIME, IME, SMER
FROM student
ORDER BY PREZIME DESC;
```

U ORDER BY klauzuli može se navesti i naziv kolone koja nije eksplicitno navedena u SELECT rečenici. Međutim, ako se u SQL upitu koriste GROUP BY ili DISTINCT klauzule, ne može se sortirati po kolonama koje nisu u SELECT listi. U tom slučaju,

sve kolone prema kojima se rezultat sortira navode se u ORDER BY klauzuli zajedno sa smerom sortiranja.

Primer: Prikazati prezime, ime studenta i studijski program na koji upisan. Prikaz sortirati po godini upisa studenta u opadajućem redosledu.

```
SELECT PREZIME, IME, SMER
FROM student
ORDER BY GODINA_UPISA DESC;
```

Za sortiranje rezultata prikaza po jednoj od kolona navedenih u SELECT listi, u ORDER BY klauzuli može se umesto imena kolone navesti broj koji označava njen položaj u listi SELECT rečenice.

```
SELECT PREZIME,IME, SMER
FROM student
ORDER BY 1 DESC;
```

Alternativni SQL upit za sortiranje prikaza podataka o studentima po prezimenima može se napisati i na sledeći način:

U prethodnim primerima, opisan je postupak sortiranja rezultujućeg prikaza po vrednostima jedne kolone. ORDER BY klauzula dozvoljava navođenje više od jedne kolone. S obzirom da redosled prioriteta u ORDER BY klauzuli ide sleva nadesno, kolone po kojima se vrši sortiranje se navode u skladu sa tim. To znači da redosled navođenja kolona određuje redosled prema kojem će se njihove vrednosti uzimati u obzir prilikom sortiranja rezultujuće tabele. Ako se za sortiranje koristi numerički položaj kolone u SELECT listi, taj broj ne sme biti veći od broja stavki u SELECT listi.

Primer: Prikazati prezime, ime studenta i studijski program na koji upisan. Prikaz sortirati po studijskom programu u opadajućem i po prezimenu u rastućem redosledu.

```
SELECT PREZIME, IME, SMER
FROM student
ORDER BY 3 DESC, PREZIME;
```

6.5.4 LIMIT

U *SQL* je moguće ograničiti broja redova koji se vraća u rezultatu upita. Ova opcija se često koristi kada se želi prikazati samo prvih nekoliko redova ili u slučaju optimizacije izvršavanja upita koji se procesiraju nad velikim količinama podataka. U *MySQL*-u, korišćenjem klauzule *LIMIT*, može se ograničiti broj redova koji se vraćaju u rezultujućoj tabeli. Ova klauzula se dodaje na kraj upita i ima dva parametra i to: red od kojeg počinje prikazivanje rezultata i maksimalan broj redova koji treba da se prikaže. Ako je potrebno prikazati samo određeni broj redova, navodi se jedan parametar koji ukazuje na maksimalan broj redova rezultujuće tabele. LIMIT

klauzula se najčešće koristi u kombinaciji sa ORDER BY kako bi redosled redova u rezultatima upita imao određeni smisao.

Primer: SQL kod u nastavku ilustruje primenu LIMIT klauzule za prikaz prvih 5 redova tabele student.

```
SELECT *
FROM student
LIMIT 5;
```

Primer: SQL kod u nastavku ilustruje primenu LIMIT klauzule za prikaz tri reda tabele student počevši od trećeg reda.

```
SELECT *
FROM student
LIMIT 2,3;
```

SQL upit obezneđuje da prva dva reda budu preskočena i da rezultujuća tabela sadrži redove tri reda počevši od trećeg reda tabele student. Ovaj pristup je koristan za prikaz samo određenog segment rezultata, kao što su, na primer, stranice u paginaciji, ili kada je poželjno preskočiti nekoliko početnih redova pre nego što se prikažu rezultati.

Primer: SQL kod u nastavku ilustruje primenu LIMIT i ORDER BY klauzule za prikaz najstarije godine upisa.

```
SELECT GODINA_UPISA
FROM student
ORDER BY 1
LIMIT 1;
```

Ovaj upit će sortirati redove u opadajućem redosledu, po vrednostima podataka u koloni GODINA_UPISA i prikazati samo prvi red rezultujuće tabele. Korišćenjem klauzule LIMIT 1, rezultat se ograničava samo na prvi red koji zadovoljava kriterijum sortiranja.

6.6 FUNKCIJE

Većina sistema za upravljanje bazama podataka (DBMS) pruža podršku za širok spektar ugrađenih funkcija, kao i mogućnost definisanja korisničkih funkcija. Funkcije u SQL-u mogu se klasifikovati kao determinističke ili nedeterminističke.

Determinističke SQL funkcije su one koje daju isti rezultat za iste ulazne vrednosti u svakom pozivu. To znači da izvršavanje funkcije sa istim ulaznim parametrima uvek generiše identičan rezultat. Na primer, funkcija koja vraća kvadrat broja ili funkcija koja vraća dužinu stringa su determinističke ako se izvršavaju nad istim ulazom. Nedeterminističke SQL funkcije su one čiji rezultat može varirati čak i kada se pozivaju sa istim ulaznim vrednostima. Ovo se dešava zbog toga što ove funkcije mogu zavisiti od spoljnih faktora koji se menjaju, kao što su vreme, stanje baze

podataka ili korisnički input. Na primer, funkcija koja vraća trenutno vreme ili funkcija koja generiše slučajan broj su nedeterminističke jer će njihov rezultat zavisiti od trenutnog vremena ili drugih faktora koji mogu varirati između različitih poziva funkcije.

Ugrađene funkcije u SQL-u se grupišu u različite kategorije, a među njima su posebno značajne sledeće:

- Aritmetičke funkcije.
- Funkcije za manipulaciju znakovnim podacima.
- Funkcije za rad sa datumima.
- Agregatne funkcije.

Ove funkcije omogućavaju izvršavanje raznih operacija nad podacima u bazi, pružajući korisnicima moćne alate za obradu i analizu informacija.

6.6.1 IFNULL()

Proces pretvaranja NULL u stvarne vrednosti može biti ključan za upravljanje podacima u bazi. Podrazumeva dodeljivanje prosleđene vrednosti poljima koja u datoj koloni imaju vrednost NULL.

Postupak pretvaranja NULL u stvarne vrednosti se može kategorizovati na sledeći način:

- Dodeljivanje prosleđene vrednosti poljima podrazumeva postavljanje prosleđene vrednosti na sva polja koja imaju NULL vrednost u određenoj koloni. Na primer, ako želimo da sve NULL vrednosti u koloni STATUS postavimo na 'Nepoznato', koristimo ovaj pristup.
- Privremena dodela vrednosti se odnosi na dodeljivanje vrednosti poljima koja imaju NULL vrednost. U slučajevima izvršavanja matematičkih operacija, da bi se dobili precizni rezultati, NULL vrednosti mogu da budu tretirane kao određena vrednost. Prilikom izvršavanja operacija sabiranja ili oduzimanja, ako želimo da NULL vrednosti budu tretirane, postavljamo ih na nulu. Slično kao i kod sabiranja i oduzimanja, NULL vrednosti mogu biti tretirane kao jedinica kada se koristi množenje ili deljenje. Kada želimo da spajamo znakovne podatke, NULL vrednosti obično ne doprinose rezultatu. Zbog toga se NULL vrednosti često zamenjuju praznom znakovnom konstantom ili nekom drugom vrednošću koja odgovara potrebama upita.

U MySQL DBMS okruženju, za zamenu NULL vrednosti određenim vrednostima, koristi se funkcija IFNULL. Ova funkcija ima dva argumenta: prvi argument predstavlja naziv kolone za koju se proverava da li je NULL, dok je drugi argument vrednost koja će biti korišćena kao zamena ukoliko je prvi argument NULL. Ako vrednost nije NULL, vraća se prvobitna vrednost.

Primer: U nastavku je naveden SQL upit koji prikazuje prezime, ime i matični broj studenta. Za studente koji nemaju upisan JMBG, u rezultujućoj tabeli, treba da bude prikazana vrednost 'Nepoznata'.

```
SELECT PREZIME, IME, IFNULL(JMBG,'Nepoznato')
FROM student;
```

Primer: Za ilustraciju primene IFNULL funkcije modifikovana je tabela *profesor*. Dodata je kolona u kojoj se beleže podaci o iznosu nagrade za određenog profesora. Potrebno je po profesoru prikazati mesečni iznos za isplatu kada se uračuna i iznos nagrade. Jednog meseca neki profesori dobiju nagradu, a neki ne.

```
ALTER TABLE profesor
ADD COLUMN NAGRADA decimal(10, 2);
```

IFNULL(NAGRADA, 0) funkcija se koristi za proveru da li postoji vrednost u koloni NAGRADA za svakog profesora. Ako je vrednost NULL (profesor nije dobio nagradu), funkcija će vratiti 0, inače će vratiti vrednost iz kolone NAGRADA.

```
SELECT BROJ_CASOVA*CENA_PO_CASU+IFNULL(NAGRADA,0)
FROM profesor;
```

Ako se prethodni upit napiše bez primene IFNULL funkcije, redovi koje nemaju vrednost u koloni NAGRADA neće biti uključeni u izraza za izračunavanje.

6.6.2 ARITMETIČKE FUNKCIJE

SQL omogućava korišćenje matematičkih funkcija u SELECT i WHERE klauzulama. Na ovaj način, u SELECT listi se može navesti i prikazati rezultat izračunavanja matematičkih izraza. Takođe, uslov naveden u WHERE klauzuli može biti matematički izraz.

Neke od aritmetičkih funkcija koje podržava MySQL DBMS su:

- ABS: Vraća apsolutnu vrednost datog broja.
- CEIL: Zaokružuje broj na višu celobrojnu vrednost.
- FLOOR: Zaokružuje broj na nižu celobrojnu vrednost.
- MOD: Vraća ostatak pri deljenju dva broja.
- ROUND: Zaokružuje realnu vrednost na određeni broj decimala.
- TRUNCATE: Odseca decimale iza određene pozicije, bez zaokruživanja.
- SIGN: Vraća znak datog broja.
- DIV: Koristi se za celobrojno deljenje.
- EXP: Koristi se za eksponencijalno stepenovanje.
- POW: Vraća vrednost broja stepenovanog na određeni broj.
- SQRT: Vraća kvadratni koren datog broja.

Primer: Neka se plata profesora računa na osnovu broju realizovanih časova na nivou meseca i cene jednog časa. Prikazati ostvarenu zaradu na godišnjem nivou. Za potrebe ovog primera modifikovana je tabela profesor dodavanjem dve kolone BROJ_CASOVA i CENA_PO_CASU. S obzirom da su kolone dodate naknadno (nakon popunjavanja tabele profesor) vrednost ograničenja NULL je podrazumevana (dozvoljava se da u poljima ovih kolona nije upisana vrednost).

```
ALTER TABLE profesor
ADD COLUMN BROJ_CASOVA int,
ADD COLUMN CENA_PO_CASU decimal(10, 2);
```

SQL upit kojim se prikazuje obračunata zarada profesora na godišnjem nivou je prikazan u nastavku. Korišćena je ROUND funkcija za postavljanje broja decimalnih mesta na nula.

```
SELECT PREZIME, IME, ROUND(BROJ_CASOVA*CENA_PO_CASU*12,0)
FROM profesor;
```

Primer: Naredni SQL upit prikazuje podatke o profesorima kod kojih iznos mesečnih primanja uvećan za 5000 ima vrednost manju od 90000.

```
SELECT PREZIME,IME,ROUND(BROJ_CASOVA*CENA_PO_CASU,0)
FROM profesor
WHERE BROJ CASOVA*CENA PO CASU+5000<90000;
```

Umesto funkcije ROUND sa postavljenim brojem decimala na nula može se koristiti FLOOR funkcija.

```
SELECT PREZIME,IME,FLOOR(BROJ_CASOVA*CENA_PO_CASU)
FROM profesor
WHERE BROJ CASOVA*CENA PO CASU+5000<90000;
```

6.6.3 ALIJAS KOLONE

Rezultatu matematičke funkcije BROJ_CASOVA*CENA_PO_CASU može da se dodeli dodeli ime odnosno alijas. Alijas kolone je način preimenovanja imena kolona na izlazu. Bez alijasa, kolone prikazane kao rezultat SELECT rečenice imaće ista imena kao kolone u tabeli ili ime koje pokazuje matematička operacija, kao BROJ_CASOVA*CENA_PO_CASU*12. Nekada je poželjno da na izlazu bude prikazano intuitivnije ime kolone koje je lakše razumeti.

Postoji nekoliko pravila kod korišćenja alijasa kolona i to:

- menja naslov kolone samo u trenutku prikaza rezultujuće tabele (stvarni naziv kolone se menja upotrebom ALTER TABELE klauzule),
- koristan je kod proračuna,
- navodi se odmah posle imena kolone,
- može imati opcionalnu ključnu reč AS između imena kolone i alijasa,
- zahteva znake navoda ako alijas sadrži blanko-znak, specijalne znake ili je osetljiv na velika i mala slova,
- ne može se navoditi u WHERE klauzuli umesto matematičkog izraza.

Primer: U nastavku je dat SQL upit koji demonstrira način primene mehanizma alijasa za imenovanje kolona u rezultujućoj tabeli.

```
SELECT PREZIME, IME, BROJ_CASOVA 'Broj časova',
CENA_PO_CASU+1000 'Uvećana cena časa'
FROM profesor
WHERE CENA_PO_CASU<2500;
```

U SQL upitu u nastavku je prikazana upotreba alijasa i u WHERE klauzuli što će rezultirati prijavu greške od strane DBMS –a.

```
SELECT PREZIME, IME,FLOOR(BROJ_CASOVA*CENA_PO_CASU) AS
'zarada'
FROM profesor
WHERE zarada<90000;
```

Ukoliko se u WHERE klauzuli navede alijas kolone umesto matematičkog izraza MySQL DBMS prijavljuje grešku *Error Code: 1054. Unknown column 'zarada' in 'where clause'*.

6.6.4 FUNKCIJE ZA MANIPULACIJU ZNAKOVNIM PODACIMA

Funkcije za rad sa stringovima omogućavaju manipulaciju znakovnim podacima. U nastavku je dat deo funkcija za rad sa stringovma koji je podržan od strane MySQL DBMS-a:

- TRIM: Koristi za uklanjanje belina (praznih prostora) sa početka i kraja stringa.
- LTRIM: Koristi se za uklanjanje belina (praznih prostora) sa leve strane stringa.
- RTRIM: Koristi se za uklanjanje belina (praznih prostora) sa desne strane stringa.
- LENGTH: Koristi za dobijanje dužine stringa, tj. broja karaktera u datom stringu. Ova funkcija može biti korisna u slučaju filtriranja i prikazivanja podataka na osnovu dužine stringa.
- LOWER: Pretvara velika slova u mala u datom stringu.
- UPPER : Pretvara mala slova u velika u datom stringu.

- SUBSTRING: Koristi za izdvajanje podstringa iz datog stringa, sa određene pozicije i određenog broja karaktera.
- REVERSE: Koristi za obrtanje redosleda karaktera u stringu.
- CONCAT: Koristi za spajanje (konkatenaciju) dva ili više stringova. Može se koristiti za spajanje konstantnih stringova, ali i za spajanje vrednosti iz različitih kolona ili izraza.

Primer: Prikazati podatke o studentima tako da prezime i ime bude navedeno u jednoj koloni rezultujuće tabele. Koristiti alijas za kolonu nastalu spajanjem prezimena i imena. Prikaz sortirati po prezimenu u rastućem redosledu.

```
SELECT CONCAT(PREZIME,' ', IME) as 'Prezime i ime studenta'
FROM student
ORDER BY 1;
```

Primer: U nastavku je dat SQL upit koji demonstrira način primene funkcije CONCAT i UPPER.

```
SELECT CONCAT(UPPER(PREZIME),' ',UPPER(IME)) as 'Prezime i
ime studenta'
FROM student
ORDER BY 1;
```

6.6.5 FUNKCIJE ZA RAD SA DATUMIMA

Funkcije za rad sa datumima omogućavaju manipulaciju, formatiranje i izvlačenje informacija iz datuma.

U nastavku je dat deo funkcija za rad sa datumima koji je podržan od strane MySQL DBMS-a:

- NOW: Vraća trenutni datum i vreme.
- CURDATE: Vraća trenutni datum.
- CURTIME: Vraća trenutno vreme.
- DATE_ADD: Dodaje određeni vremenski interval na vrednost datuma.
- DATE_SUB: Oduzima određeni vremenski interval od vrednosti datuma.
- DATEDIFF: Računa razliku između dva datuma u danima.
- DATE_FORMAT: Prikazuje datum u definisanom formatu.
- DAYOFWEEK: Vraća dan u nedelji kao broj (1 za nedelju, 2 za ponedeljak itd.).
- MONTH(date): Vraća mesec iz datuma kao broj (1 za januar, 2 za februar itd.).
- YEAR(date): Vraća godinu iz datuma.

Primer: Prikazati podatke o profesorima koji su se zaposlili posle 2002. godine.

```
SELECT PREZIME, IME, YEAR(DATUM_ZAP)
FROM profesor
WHERE YEAR(DATUM ZAP)>2002;
```

6.6.6 AGREGATNE FUNKCIJE

Agregatne funkcije igraju ključnu ulogu u dobijanju grupisanih i sumarnih informacija u rezultatima upita. Smatraju se višerednim ili grupnim funkcijama jer se primenjuju nad grupom redova da bi dale jedan rezultat po grupi. Ove funkcije uzimaju više redova kao ulaz i vraćaju jedan rezultat kao izlaz.

Navode se u listi kolona *SELECT* klauzule, gde omogućavaju agregiranje vrednosti za određene podatke. Grupa podataka može obuhvatiti celu tabelu, redove koji zadovoljavaju određeni uslov, grupisane redove po jednoj ili više kolona, kao i redove unutar tih grupa koji zadovoljavaju postavljeni uslov. Prilikom primene na podatke, *NULL* vrednosti se zanemaruju.

Agregatne funkcije ne mogu se koristiti direktno u *WHERE* klauzuli. Važno je napomenuti da su funkcije agregacije deterministčke funkcije, što znači da daju konzistentan rezultat za iste ulazne vrednosti u svakom pozivu.

Neki od čestih primera agregatnih funkcija u *MySQL DBMS* okruženju uključuju:

- AVG(kolona): Izračunava srednju vrednost kolone.
- SUM(kolona): Izračunava sumu svih vrednosti atributa.
- MIN(kolona): Nalazi minimalnu vrednost atributa.
- MAX(kolona): Nalazi najveću vrednost atributa.
- COUNT(*): Nalazi broj redova u tabeli ili grupi.
- COUNT(kolona): Nalazi broj redova sa ne-NULL vrednostima u određenoj koloni.
- COUNT(DISTINCT kolona): Nalazi broj redova sa različitim vrednostima u određenoj koloni.

Funkcije *MIN*, *MAX* i *COUNT* mogu raditi sa bilo kojim tipom podataka, a *SUM*, *AVG*, *STDDEV* i *VARIANCE* zahtevaju numeričke vrednosti radi pravilnog izračunavanja.

Primer: Funkcija *COUNT* prikazuje broj redova u rezultujućoj tabeli. U nastavku je naveden *SQL* upit koji se određuje broj studenata o kojima se čuvaju podaci u bazi podataka.

```
SELECT COUNT(*)'Broj studenata'
FROM student;
```

Primer: Sledeći *SQL* upit određuje maksimalnu, minimalnu, prosečnu i ukupnu cenu po času svih profesora.

```
SELECT MAX(CENA_PO_CASU), MIN(CENA_PO_CASU),
AVG(CENA_PO_CASU), SUM(CENA_PO_CASU)
FROM profesor;
```

Funkcije agregacije nije moguće koristiti direktno u *WHERE* klauzuli. Razlog tome leži u činjenici da se rezultat funkcija agregacija izračunava nakon što se odrede redovi koje ulaze u sastav rezultujuće tabele, odnosno nakon obrade uslova postavljenog u *WHERE* klauzuli. Ova ograničenja proizlaze iz redosleda obrade *SQL* upita.

Primer: Sledeći SQL ne može biti izvršen. Generiše se greška *Error Code: 1111. Invalid use of group function.*

```
SELECT PREZIME,IME,CENA_PO_CASU
FROM profesor
WHERE CENA_PO_CASU>AVG(CENA_PO_CASU);
```

6.6.7 COUNT()

Često se javlja potreba za određivanjem ukupnog broja zapisa ili redova u tabeli. Funkcija **COUNT**(kolona_za_brojanje) broji redove u kojima *kolona_za_brojanje* ima vrednost različitu od NULL i koji zadovoljavaju, ukoliko je naveden, uslov u WHERE klauzuli. Funkcija COUNT(*) se koristi za brojanje svih redova u tabeli ili svih redova koji zadovoljavaju uslov (ukoliko je naveden) u WHERE klauzuli.

U nastavku je dat primer koji ilustruje rezultat COUNT funkcije za različite slučajeve primene.

Kolona1	Kolona2	Kolona3	Kolona4
2		Lll	
3	L	Bbb	
	L		
4	L		

COUNT(Kolona1) = 3
COUNT(Kolona2) = 3
COUNT(Kolona3) = 2
COUNT(Kolona4) = null
COUNT(*) = 4

Ukoliko je potrebno izbrojati samo različite vrednosti u specificiranoj koloni koristi se ključna reč DISTINCT, odnosno COUNT(DISTINCT kolona_za_brojanje) oblik funkcije.

Za slučaj primene funkcije COUNT(DISTINCT kolona_za_brojanje) na podatke iz prethodnog primera, rezultujuće vrednosti će biti nešto drugačije.

Kolona1	Kolona2	Kolona3	Kolona4
2		Lll	
3	L	Bbb	
	L		
4	L		

COUNT(DISTINCT Kolona1) = 3
COUNT(DISTINCT Kolona2) = 1
COUNT(DISTINCT Kolona3) = 2

6.6.8 GROUP BY, HAVING

U prethodnom delu ovog poglavlja, objašnjena je primena funkcija agregacije, koja omogućava generisanje sumarnih informacija na osnovu podataka koji se nalaze u bazi podataka. Kombinacija funkcija agregacije sa GROUP BY i HAVING klauzulom naglašava njihovu svrhu i praktičnu primenu.

Klauzula **GROUP BY** ima za cilj grupisanje redova tabele na osnovu zajedničkih vrednosti u specificiranim kolonama. Omogućava primenu funkcija agregacije ne samo na celu rezultujuću tabelu, već i na posebne grupe redova čime se povećava korisnost ovih funkcija. Kreiranje grupa redova se postiže SELECT naredbom. Grupisanje se vrši tako što se redovi koji imaju jednake vrednosti u kolonama po kojima se grupisanje smeštaju u istu grupu. Nakon GROUP BY klauzule, sve operacije se izvršavaju nad kreiranim grupama. Kada se grupisanje vrši po više kolona, redovi sa istom vrednošću u prvoj koloni formiraju jednu grupu, a unutar tih grupa se redovi ponovo reorganizuju prema vrednostima druge kolone, i tako dalje. Redosled po kome su kolone za grupisanje navedene utiče na način formiranja grupa sa podgrupama u okviru njih.

Klauzula GROUP BY ne zamenjuje ORDER BY. GROUP BY se isključivo primenjuje za grupisanje redova po određenim kolonama, a ORDER BY za sortiranje prikaza rezultujuće tabele. Sortiranje može biti rastuće (ASC) i opadajuće (DESC), dok grupisanje podržava samo rastući redosled. Takođe, sortiranje ne mora biti primenjeno na sve kolone navedene u SELECT klauzuli, dok je za grupisanje neophodno da se navedu sve kolone koje se koriste za grupisanje.

Klauzula GROUP BY se izvršava nakon WHERE klauzule. Uslov naveden u klauzuli WHERE se primenjuje na originalni skup podataka pre nego što se izvrši grupisanje. To znači da redovi koji ne zadovoljavaju uslov naveden u WHERE klauzuli neće biti uključeni u rezultujuće grupe, jer su isključeni pre izvršenog grupisanja.

Važno: Ako nije navedena GROUP BY klauzula, nije moguće koristiti funkcije agregacije u kombinaciji sa imenima kolona u SELECT klauzuli.

Primer: Za ilustraciju primene GROUP BY klauzule i COUNT funkcije agregacije u nastavku je naveden SQL upit koji prikazuje upisan broj studenata po studijskim programima.

```
SELECT SMER, COUNT(*)
FROM student
GROUP BY SMER;
```

Ukoliko se ne navede GROUP BY klauzula, u SELECT klauzuli nije moguće kombinovati funkcije agregacije sa imenima kolona. MySQL DBMS generiše grešku *Error Code: 1140. In aggregated query without GROUP BY, expression #1 of SELECT list contains nonaggregated column 'studentski_servis.student.SMER'; this is incompatible with sql_mode=only_full_group_*by

Primer: Primena GROUP BY klauzule i AVG funkcije agregacije navedena je u SQL upitu koji prikazuje prosečnu cenu časa po zvanjima profesora.

```
SELECT ZVANJE,ROUND(AVG(CENA_PO_CASU), 2)
FROM profesor
GROUP BY ZVANJE;
```

Primer: Grupisanje po dve kolone ilustrovano je SQL upitom koji prikazuje upisan broj studenata po studijskom programu za određenu školsku godinu.

```
SELECT SMER, GODINA_UPISA, COUNT(*)
FROM student
GROUP BY SMER, GODINA_UPISA
ORDER BY 1;
```

U prethodnom primeru, ukoliko se izostavi kolonu GODINA_UPISA iz GROUP BY klauzule, a navedena je u SELECT listi MySQl DBMS prijavljuje grešku *Error Code: 1055. Expression #2 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'studentski_servis.student.GODINA_UPISA' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by*

Važno: Sve kolone navedene u SELECT listi, na koje nije primenjena neka funkcija agregacije, moraju da budu specificirane u GROUP BY klauzuli. U suprotnom, upit se neće izvršiti i MySQL DBMS prijavljuje grešku.

Napomena: Starije verzije MySQL DBMS neće prijaviti grešku ako kolone navedene u SELECT listi, na koje nije primenjena neka funkcija agregacije, nisu specificirane u GROUP BY klauzuli. Međutim, SQL upit generisan na ovaj način neće dati ispravan rezultat.

Takođe treba napomenuti da, i ako u SELECT klauzuli nije navedeno ništa osim agregatne funkcije, grupisanje se može izvršiti navođenjem drugih kolona tabele.

Primer: SQL upit prikazuje primenu grupisanja kada je u SELECT listi navedena samo agregatna funkcija.

```
SELECT COUNT(*)
FROM student
GROUP BY SMER;
```

Klauzula **HAVING** ("koji imaju") se koristi za filtriranje rezultata koji su grupisani korišćenjem GROUP BY klauzule. Kombinuje se sa funkcijama agregacije poput COUNT, SUM, AVG, itd. Ova klauzula omogućava primenu uslova filtriranja na formirane grupe, što je korisno kada želimo da izvršimo filtriranje na rezultate agregatnih funkcija po grupama.

Ukoliko se u SQL upitu koriste klauzule WHERE, GROUP BY i HAVING neophodno je ispoštovati redosled njihovog navođenja i to:

- WHERE: Filtriranje redova pre grupisanja.
- GROUP BY: Grupisanje redova.
- HAVING: Filtriranje grupa nakon grupisanja.

Važno: HAVING klauzula ne može biti navedena ako prethodno nije izvršeno grupisanje u GROUP BY klauzuli. Međutim, GROUP BY može biti navedena bez HAVING klauzule. U GROUP BY klauzuli ne može se koristiti alijas umesto naziva kolone, dok u HAVING klauzuli to jest moguće. HAVING klauzula ne treba da se koristi umesto WHERE klauzule jer WHERE vrši filtriranje redova, dok HAVING filtrira grupe.

***Primer*:** Prikazati broj upisanih studenata po studijskim programima za studijske programe na kojima je upisano više od pet studenta.

```
SELECT SMER, COUNT(*)
FROM student
GROUP BY SMER
HAVING COUNT(*)>5;
```

***Primer*:** Po godinama zaposlenja i zvanjima prikazati broj zaposlenih profesora, samo za godine u kojime je zaposleno dva i više profesora.

```
SELECT YEAR(DATUM_ZAP), ZVANJE, COUNT(*)
FROM profesor
GROUP BY YEAR(DATUM_ZAP), ZVANJE
HAVING COUNT(*)>=2
ORDER BY 1;
```

Napomena: U nastavku je dat prikaz anatomije SELECT rečenice sa značajnim karakteristikama i ograničenjima

```
SELECT kolona 1, kolona 2, agregatna funkcija 1, agregatna funkcija 2, …
FROM tabela
WHERE uslov na nivou reda – NE MOŽE AGEGATNA FUNKCIJA I ALIJAS KOLONE
GROUP BY kolona 1, kolona 2, … NE MOŽE ALIJAS KOLONE
HAVING uslov na nivou grupe redova – može agregatna funkcija – može alijas kolone
ORDER BY naziv kolone ili numerička pozicija u SELECT listi, funkcija ili izraz [ASC/DESC] – može alijas kolone
LIMIT ofset ili broj redova koji se prikazuje – samo kod MySQL DBMS;
```

6.7 PITANJA I ZADACI

1. Objasniti podelu SQL- a na podjezike DDL, DML, DCL, TCL.
2. Navesti i objasniti osnovne kategorije tipova podataka koje podržava većina relacionih baza podataka.
3. Primenom odgovarajuće naredbe kreirati bazu podataka ORDINACIJA.
4. Primenom odgovarajuće naredbe kreirati tabele koje sadrže podatke o pacijentima, lekarima, pregledima i uslugama.
5. Napisati SQL upit za prikaza podataka o pacijentima koji su punoletni. Prikaz sortirati u opadajućem redosledu po godini rođenja i prezimenu pacijenta.
6. Napisati SQL upit za prikaza podataka o lekarima čije ime počinje sa slovom P.
7. Napisati SQL upit za prikaza usluga čija je cena između 2000 i 5000 uključujući i granične vrednosti.
8. Napisati SQL upit za prikaza podataka o pregledima pacijenta sa identifikacionim brojem 100.
9. Napisati SQL upit za prikaz podataka o pregledima lekara sa identifikacionim broj 2 koji su obavljeni u mesecu maju godine 2022.
10. Napisati SQL upit za prikaz broja pregleda po identifikacionim brojevima lekara, ali samo za lekare koji su imali više od 2 pregleda.

7 PRIKAZ PODATAKA IZ VIŠE TABELA

Cilj ovog poglavlja se odnosi na objašnjenje prikaza podataka izdvojenih iz dve i više tabela. Obrađeni su načini povezivanja tabela i to: bezuslovno spajanje, spajanje po jednakosti, unutrašnje spajanje, prirodno spajanje, spoljašnje spajanje, spajanje tabele sa samom sobom kao i različite kategorije podupita.

7.1 TIPOVI SPOJEVA

SQL upiti, objašnjeni u prethodnom poglavlja, su prikazivali podatke samo iz jedne tabele. Imajući u vidu da su relacione baze podataka u pitanju, očekuje se da relevantne informacije često proizilaze iz podataka smeštenih u različitim tabelama. U takvim slučajevima, potrebno je povezati redove iz različitih tabela kako bi se generisala rezultujuća tabela.

Ukoliko je u FROM klauzuli navedeno više od jedne tabele neophodno je primeniti mehanizma spajanja. Navođenje uslova spajanja oodrazumeva specifikaciju kolona na osnovu kojih se vrši povezivanje redova iz različitih tabela. Spajanje tabela se najčešće ostvaruje tako što se strani ključ iz jedne povezuje sa primarnim ključem u drugoj tabeli. Uslov spajanja se može definisati u okviru WHERE klauzule ili korišćenjem ključne reči JOIN direktno u FROM klauzuli.

SQL standardom definisane su sledeće kategorije povezivanja tabela:

- Dekartov proizvod ili bezuslovno spajanje (engl.*cross join*)
- Spajanje po jednakosti (engl. *equi join*)
- Unutrašnje spajanje (engl. *inner join*)
- Prirodno spajanje (engl. *natural join*)
- Spoljašnje spajanje (engl. *outer join*)
- Spajanje tabele sa samom sobom (engl. *self join*)
- Relacijsko deljenje (engl. *subquery*)

7.1.1 DEKARTOV (CROSS–JOIN) PROIZVOD

Dekartov ili Kartezijanski proizvod dve tabele predstavlja rezultat operacije u kojoj se svaki red jedne tabele kombinuje sa svakim redom druge tabele. S obzirom da nema uslova spajanja, rezultujuća tabela sadrži sve moguće kombinacija redova iz obe tabele.

Ovakvo spajanje retko pruža korisnu informaciju jer može proizvesti ogroman broj redova u rezultatu, što često dovodi do dugog vremena izvršavanja upita. Dekartov proizvod se obično koristi oprezno i samo kada je to zaista potrebno, jer može opteretiti bazu podataka i pružiti neefikasne rezultate.

7.1.2 SPAJANJE PO JEDNAKOSTI (EQUI-JOIN)

Spajanje po jednakosti predstavlja povezivanje podataka iz dve ili više tabela na osnovu jednakosti odgovarajućih kolona, najčešće primarnih i stranih ključeva. Kombinovanje podataka iz dve tabele obično zahteva navođenje uslova spajanja kako bi se ograničile kombinacije redova jedne tabele sa redovima druge tabele. Uslov spoja po jednakosti se može navesti u WHERE klauzuli (stariji način spajanja) ili ON klauzuli kada se u FROM koristi klauzula INNER JOIN između tabela koje se povezuju. Ova veza između tabela se definiše preko njihovih ključeva, koristeći primarne i strane ključeve kako bi se uspostavila korelacija između odgovarajućih redova.

Opšta sintaksa spajanja po jednakosti u WHERE klauzuli je data u nastavku:

```
SELECT kolona1, kolona2,…
FROM tabela1, tabela2
WHERE tabela1.kolona= tabela2.kolona;
```

U FROM klauzuli se navode tabele koje se spajaju razdvojene zarezom. Uslov spajanja se specificira u WHERE klauzuli i najčešće predstavlja uslov jednakosti između stranog ključa iz tabele 1 i primarnog ključa iz tabele 2.

Primer: Ilustracija spajanja po jednakosti u WHERE klauzuli je prikazana u SQL upitu koji generiše rezultujuću tabelu sa podacima o profesorima i predmetima koji predaju.

```
SELECT PREZIME,IME,NAZIV
FROM profesor, predmet
WHERE predmet.ID_PROFESORA = profesor.ID_PROFESORA
ORDER BY 1;
```

7.1.3 UNUTRAŠNJE SPAJANJE (INNER JOIN)

Unutrašnje spajanje predstavlja najčešći korišćen tip povezivanja tabela. Generalno je ekvivalento spajanju po jednakosti koje je realizovano u WHERE klauzuli. Koristi se za kombinovanje redova iz dve ili više tabela na osnovu jednakosti vrednosti u određenim kolonama. Ključno je da JOIN klauzula eksplicitno navodi vrstu spajanja (u ovom slučaju INNER JOIN), dok se uslov spajanja (u ovom slučaju, uslov jednakosti) definiše unutar ON klauzule.

Opšta sintaksa unutrašnjeg spajanja je data u nastavku:

```
SELECT kolona1, kolona2
FROM tabela1 INNER JOIN tabela2
ON tabela1.kolona_t1= tabela2. kolona_t2;
```

U FROM klauzuli, između tabela koje se spajaju, navodi se INNER JOIN s tim da se reč INNER podrazumeva tako da se može navesti samo JOIN. U ON klauzuli se definiše uslov spajanja *tabela1.kolona= tabela2.kolona*. Na ovaj način se obezbeđuje da redovi iz *tabele1* i *tabele2* se spajaju samo ako imaju iste vrednosti u kolonama *kolona_t1, kolona_t2*.

Važno: Broj uslova spojeva koji se navodi u ON ili WHERE klauzuli zavisi od broja tabela navedenih u FROM klauzuli i može se definisati na sledeći način. Neka je ***n*** broj tabela koje su navedene u FROM klauzuli. Broj uslova spojeva koji treba da se navede u klauzuli spajanja se izračunava kao ***n-1***. Ovo je zato što svaka tabela (osim poslednje) mora biti spojena sa nekom drugom tabelom. Kada je i poslednja tabela spojena sa prethodnom, nema potrebe za dodatnim uslovima spoja. Ovaj princip olakšava pisanje SQL upita i pomaže u održavanju preglednosti i logike spojeva.

Primer: Ilustracija unutrašnjeg spajanja je prikazana u SQL upitu koji generiše rezultujuću tabelu sa podacima o profesorima i predmetima koji predaju.

```
SELECT PREZIME, IME, NAZIV
FROM professor JOIN predmet
ON predmet.ID_PROFESORA = profesor.ID_PROFESORA
ORDER BY 1;
```

Postoji nekoliko razloga zbog kojih se preporučuje korišćenje unutrašnjeg spajanja umesto spajanja u WHERE klauzuli:

- **Čitljivost.** Upiti koji koriste JOIN klauzule su često čitljiviji i lakši za razumevanje, naročito kada je potrebno spojiti više tabela. U slučaju spajanja u WHERE klauzuli i postavljanja dodatnih uslova za filtriranje redova, upit postaje manje pregledan i teže čitljiv.
- **Performanse.** SQL optimizator baze podataka ostvaruje bolju optimizaciju upita koji koriste JOIN klauzule, naročito kada je potrebno odrediti redosled izvršavanja spojeva. Spajanje u WHERE klauzuli može otežati optimizaciju.
- **Fleksibilnost.** JOIN klauzule omogućavaju različite tipove spojeva kao što su INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, dok spajanje u WHERE klauzuli obično podrazumeva INNER JOIN. Ovo omogućava veću fleksibilnost u oblikovanju upita i olakšava prilagođavanje zahtevima konkretne situacije.

7.1.4 KVALIFIKOVANJE KOLONA

Kada se u SELECT listi navede kolona koja postoji u više tabela koje su specificirane u FROM klauzuli, neophodno je izvršiti kvalifikovanje te kolone dodavanjem ispred njenog naziva tabele kojoj pripada. Ovo je bitno kako bi SQL engine mogao jednoznačno da tačno identifikuje kolona koja se koristi.

Na primer, za kolonu ID_PROFESORA u tabelama predmet i profesor, mora se napisati profesor. ID_PROFESORA ili predmet. ID_PROFESORA kako bi se jasno odredila kolona koja se koristi. Na ovaj način se osigurava tačnost i čitljivost upita, posebno u složenim SQL upitima koji uključuju više tabela.

***Primer*:** Ilustracija kvalifikovanja kolona je prikazana navođenjem u SELECT listi kolone ID_PROFESORA.

```
SELECT profesor.ID_PROFESORA, PREZIME, IME, NAZIV
FROM profesor JOIN predmet
ON predmet.ID_PROFESORA=profesor.ID_PROFESORA;
```

Ukoliko se navede kolona ID_PROFESORA bez naziva tabele kojoj pripada, MySQL DBMS prijavljuje grešku: *Error Code: 1052. Column 'ID_PROFESORA' in field list is ambiguous.*

7.1.5 UPOTREBA ALIJASA TABELA

Rad sa više tabela u SQL-u može postati složeniji kada se u različitim tabelama nalaze kolone istog imena, a barem jedna od njih se koristi u upitu. Da bi se jasno identifikovale ove kolone, osim njihovih naziva, treba navesti i naziv tabele u kojoj se svaka kolona nalazi. Ovo se postiže kvalifikovanjem imena kolone sa imenom tabele ili njenim alijasom. Osim toga, kako bi se smanjila redundantnost i olakšalo korišćenje tabela u upitu, preporučuje se korišćenje alijasa za tabele, što omogućava korišćenje kraćih oznaka umesto punih imena tabela.

Alijasi tabela se definišu u FROM klauzuli prilikom specificiranja naziva tabela koje učestvuju u spoju bilo da se radi o prirodnom ili spoljašnjem spajanju. Na primer, umesto da pišemo student.ime, možemo koristiti s.ime, gde je s alijas za tabelu student. Ovakav pristup čini SQL upite čitljivijim i olakšava održavanje koda.

Primer: Ilustracija upotrebe alijasa tabela je prikazana u SQL upitu koji generiše rezultujuću tabelu sa podacima o profesorima i predmetima koji predaju.

```
SELECT prof.PREZIME, prof.IME, pred.NAZIV
FROM profesor prof JOIN predmet pred
ON pred.ID_PROFESORA=prof.ID_PROFESORA;
```

Važno: Prilikom upotrebe alijasa tabela potrebno je održati doslednost. U *FROM* klauzuli navodi se puno ime tabele i dodeli skraćenica za tabelu koja se koristi u tom

upitu. Ako su uvedeni alijasi tabela, moraju se koristiti u svim kvalifikovanim nazivima kolona u klauzulama *SELECT*, *WHERE* i druge.

7.1.6 PRIRODNO SPAJANJE (NATURAL JOIN)

Prirodno spajanje podrazumeva spajanje dve tabele u kojoj se kolone koje se koriste za spajanje automatski određuju na osnovu njihovih imena, umesto da se eksplicitno navode u ON klauzuli. Kada se koristi NATURAL JOIN u klauzuli FROM, SQL sistem će automatski tražiti kolone sa istim nazivima u obe tabele i koristiti ih za spajanje po jednakosti tih kolona. U tom slučaju, dodavanje dodatne klauzule za uslov spajanja, kao što je ON, postaje suvišno jer sam sistem samostalno određuje kolone za spajanje.

Prednosti upotrebe prirodnog spajanja su:

- **Jednostavnost**. Automatski spaja tabele na osnovu kolona sa istim imenom, eliminišući potrebu za eksplicitnim navođenjem kolona koje se koriste za spajanje. Ovo čini upite jednostavnijim i lakšim za pregled.
- **Smanjenje redundantnosti koda.** Upiti postaju jednostavniji i manje skloni greškama jer se izbegava ponavljanje koda i lakše je održavati ih.

Mane upotrebe prirodnog spajanja su:

- **Nedostatak kontrole nad spajanjem.** Gubi se kontrola nad postupkom spajanja jer se zasniva isključivo na imenima kolona, a ne na eksplicitno definisanim uslovima. To može dovesti do neočekivanih i nepravilnih rezultata.
- **Nedostatak fleksibilnosti.** Manje je fleksibilan jer se ne može eksplicitno definisati uslov spajanja. Naknadna promena strukture tabela ili dodavanje nove kolone s istim imenom kao neka od postojećih može uticati na rezultate upita, što može biti problematično u složenijim sistemima.

NATURAL JOIN može biti koristan za jednostavne upite sa jasnim i stabilnim strukturama podataka. Međutim, ipak je preporučljivo koristiti eksplicitno postavljanje uslova spajanja (*ON*) kako bi se očuvala veća kontrola i smanjila mogućnost neočekivanih problema.

Primer: SQL upit iz prethodnog primera je modifikovan upotrebom prirodnog spajanja bez postavljanja eksplicitnog uslova u ON klauzuli.

```
SELECT prof.PREZIME, prof.IME, pred.NAZIV
FROM profesor prof NATURAL JOIN predmet pred;
```

Klauzula *NATURAL JOIN* automatski spaja tabele *profesor* i *predmet* na osnovu kolona sa istim nazivom, a to je u ovom slučaju *ID_PROFESORA*.

7.1.7 PRIRODNO SPAJANJE UPOTREBOM USING KLAUZULE

Iako prirodno spajanje značajno doprinosi čitljivosti SQL koda, treba imati na umu neke od problema koji mogu nastati:

- Nepotrebno uparivanje svih kolona. Ponekad priroda upita ne zahteva spajanje svih kolona sa istim imenima. U takvim slučajevima, korišćenje NATURAL JOIN-a može dovesti do nepotrebnog filtriranja i rezultirati netačnim ili nepotpunim rezultatima.
- Različiti tipovi podataka. Ako kolone s istim imenima iz različitih tabela imaju različite tipove podataka, nemoguće je izvršiti operaciju poređenja po jednakosti. Ovo može izazvati greške pri izvršavanju upita.
- Gubitak performansi. U slučajevima kada postoje velike tabele ili kompleksni upiti, NATURAL JOIN može biti manje efikasan u odnosu na spajanje sa uslovom eksplicitno navedenim u ON klauzuli jer SQL engine mora izvršiti dodatne korake kako bi odredio kolone za spajanje.

Korišćenjem klauzule USING u običnom JOIN-u, mogu se izbeći neki od potencijalnih problema koji mogu nastati prilikom korišćenja NATURAL JOIN-a. Klauzula USING omogućava eksplicitno specificiranje kolone za spajanje, umesto njihovog automatskog određivanja od strane sistema.

```
SELECT prof.PREZIME, prof.IME, pred.NAZIV
FROM profesor prof JOIN predmet pred
USING(ID_PROFESORA);
```

Kada se koristi klauzula USING u SQL upitu, kolone koje se navode u toj klauzuli ne smeju imati kvalifikator (ime tabele ili alijas tabele) u SQL rečenici. Ovo je zato što se kolone automatski povezuju sa kolonama istih imena u drugoj tabeli koja je navedena u klauzuli JOIN.

7.2 SPOLJAŠNJE SPAJANJE (OUTER JOIN)

Spoljašnje spajanje podrazumeva spajanje dve tabele po vrednosti određene kolone i uključivanje redova iz jedne ili druge tabele (ili iz obe) koji ne zadovoljavaju uslov spajanja. To znači da se u rezultatu spajanja prikazuju i zapisi iz tabele za koje u drugoj tabeli, koja je deo spoja, ne postoje odgovarajući zapisi. U rezultat spajanja se takođe uključuju redovi koji ne zadovoljavaju uslov spajanja, tj. nemaju parnjaka u obe tabel, a istovremeno zadovoljavaju uslove navedene u WHERE klauzuli.

Standard SQL-a definiše sledeće vrste spoljašnjeg spajanja:

- **Left outer join.** Uključuje sve redove iz leve (prve) tabele, i redove iz desne (druge) tabele koji se podudaraju prema uslovu spajanja, ali i one redove iz desne tabele koji nemaju odgovarajuće redove u levoj tabeli.

- **Right outer join.** Suprotno od left join-a, uključuje sve redove iz desne (druge) tabele i redove iz leve (prve) tabele koji se podudaraju prema uslovu spajanja, ali i one redove iz leve tabele koji nemaju odgovarajuće redove u desnoj tabeli.
- **Full outer join.** Uključuje sve redove iz obe tabele, i redove iz leve i desne tabele koji se podudaraju prema uslovu spajanja, ali i one redove koji nemaju odgovarajuće redove u drugoj tabeli.

7.2.1 LEFT JOIN

Levi spoljašnji spoj (*T1 LEFT OUTER JOIN T2*) uključuje redove koji su rezultat unutrašnjeg spoja, ali takođe uključuje i redove iz tabele *T1* (leve tabele) koji nemaju odgovarajući par u tabeli *T2* (desnoj tabeli). Za redove iz tabele *T1* koji se ne mogu spojiti ni sa jednim redom iz tabele *T2*, vrednosti kolona iz tabele *T2* će biti NULL.

Primer: Radi ilustracije primene levog spoljašnjeg spoja, tabela *predmet* je modifikovana dodavanjem ograničenja na kolonu ID_PROFESORA tako da može sadržati vrednosti NULL. To znači da može postojati predmet koji nije dodeljen ni jednom profesoru.

SQL upit demonstrira implementaciju levog spoljašnjeg spoja kako bi se prikazali predmeti koji su dodeljeni profesoru, uključujući i one predmete koji nemaju dodeljenog profesora.

```
SELECT pred.NAZIV, prof.PREZIME, prof.IME
FROM predmet pred LEFT JOIN profesor prof
ON pred.ID_PROFESORA = prof.ID_PROFESORA;
```

U rezultujućoj tabeli za predmete koji imaju dodeljenog profesora biće prikazani nazivi predmeta, njegovo prezime i ime. Za predmete koji nemaju dodeljenog predmetnog profesora kolone *prof.PREZIME* i *prof.IME* sadržaće *NULL* vrednost.

7.2.2 RIGHT JOIN

Desni spoljašnji spoj funkcioniše na isti načina kao i levi spoj. Razlika je što se u rezultujućoj tabeli prikazuju svi redovi iz tabele koja se navodi sa desne strane spoja. *T1 RIGHT OUTER JOIN T2* uključuje redove koji su rezultat unutrašnjeg spoja, ali takođe uključuje i redove iz tabele *T2* (desne tabele) koji nemaju odgovarajući par u tabeli *T1* (levoj tabeli).

Za redove iz tabele *T2* koji se ne mogu spojiti ni sa jednim redom iz tabele *T1*, vrednosti kolona iz tabele *T1* će biti NULL.

Primer: SQL upit iz prethodnog primera primenom RIGHT JOIN može da generiše rezultujuću tabelu sa istim rezultatima samo što je u ovom slučaju *predmet* sa desne strane spoja.

```
SELECT pred.NAZIV, prof.PREZIME, prof.IME
FROM profesor prof RIGHT JOIN predmet pred
ON pred.ID_PROFESORA=prof.ID_PROFESORA;
```

Levo i desno spoljašnje spajanje se često koristi u održavanju baze podataka za nekoliko ključnih zadataka:

- Uklanjanje zapisa "siročića". LEFT ili RIGHT JOIN omogućava identifikaciju zapisa koji nisu u relaciji sa drugim entitetima u bazi podataka. Koristeći jedno od navedenih spoljašnje spajanje, mogu se identifikovati zapisi koji nemaju odgovarajuće veze ili reference sa drugim tabelama, a zatim i ukloniti iz baze podataka.
- Pronalaženje duplikata podataka. Levo ili desno spoljašnje spajanje se može koristi za pronalaženje duplikata unutar tabela. Kroz ovaj proces, mogu se identifikovati redovi koji imaju iste vrednosti u određenim kolonama. Nakon identifikacije, mogu se izvršiti brisanje ili ažuriranje duplikata.
- Kopiranje u novu tabelu. Ponekad se podaci iz jedne tabele kopiraju u novu tabelu radi dalje analize ili obrade. Levo spoljašnje spajanje može biti korisno u ovom procesu za filtriranje podataka pre kopiranja u novu tabelu, čime se osigurava da se kopiraju samo relevantni podaci.Desno spoljašnje spajanje se može koristiti u nekim od sledećih situacija:
- Popunjavanje nedostajućih informacija. Kada je potrebno popuniti nedostajuće podatke u jednoj tabeli podacima iz druge tabele LEFT ili RIGHT JOIN omogućava spajanje dve tabele tako da svaki red iz leve ili desne tabele bude prikazan, čak i ako nema odgovarajućeg podudaranja u desnoj ili levoj tabeli, što može biti korisno za popunjavanje nedostajućih vrednosti.

7.2.3 FULL JOIN

Potpuno spoljašnje spajanje, odnosno FULL OUTER JOIN vraća rezultat koji sadrži sve redove iz obe tabele, bez obzira na to da li postoji podudaranje ili ne. Može se reći da generiše rezultujuću tabelu koja predstavlja kombinaciju rezultata levog i desnog spoljašnjeg spajanja. Potpuni spoljašnji spoj (*T1 FULL OUTER JOIN T2*) pored redova koje uključuje unutrašnji spoj u rezultat uključuje i redove iz obe tabele (T1 i T2) koje nemaju odgovarajuće parove u drugoj tabeli. U vrstama koje ne mogu da se upare, nedostajuće kolone imaju vrednost NULL.

FULL JOIN se koristi kada je potrebno prikazati sve podatke iz obe tabele, bez obzira na to da li postoji podudaranje između njih. To je korisno u analizi podataka kada želite da uključite sve dostupne informacije iz obe tabele, čak i ako one nemaju potpuno podudaranje.

U MySQL-u, FULL JOIN se ne podržava eksplicitno kao ključna reč kao što je to slučaj u drugim sistemima upravljanja bazama podataka, poput PostgreSQL-a ili SQL Server-a. Međutim, isti rezultat se može postići koristeći kombinaciju LEFT JOIN i RIGHT JOIN uz operator UNION. UNION je skupovni operator koji se koristi za spajanje rezultata više SELECT upita u jedan rezultat. Kada se koristi UNION, MySQL kombinuje rezultate iz svih SELECT upita i uklanja duplikate.

Primer: Radi ilustracije primene potpunog spoljašnjeg spoja, u tabeli *profesor* su dodata dva zapisa o profesorima kojima nemaju dodeljene predmete. *SQL* upit demonstrira implementaciju potpunog spoljašnjeg spoja kako bi se prikazali predmeti koji su dodeljeni profesoru, uključujući predmete koji nemaju dodeljenog profesora i profesore koji nemaju dodeljene predmete.

```
SELECT pred.NAZIV, prof.PREZIME, prof.IME
FROM  profesor prof RIGHT JOIN predmet pred
ON pred.ID_PROFESORA=prof.ID_PROFESORA
UNION
SELECT pred.NAZIV, prof.PREZIME, prof.IME
FROM profesor prof LEFT JOIN predmet pred
ON pred.ID PROFESORA = prof.ID PROFESORA;
```

7.2.4 SELF JOIN

SELF JOIN predstavlja operaciju spajanja tabele sa samom sobom. To znači da se koristi jedna tabela, ali se prave dve virtuelne instance te iste tabele kako bi se podaci iz njih uporedili i kombinovali. Osnovni koncept se odnosi na poređenje redova u istoj tabeli, kada ti redovi imaju neku vrstu odnosa ili hijerarhije između sebe. SELF JOIN se često koristi kada tabela sadrži informacije koje se međusobno odnose, kao što su nadređeni i podređeni entiteti, ili kada postoji potreba za upoređivanjem ili analizom podataka unutar iste tabele, posebno kada postoji relacija 1:1 ili 1:N unutar objekata te tabele.

Primena *SELF JOIN* -a može biti različita:

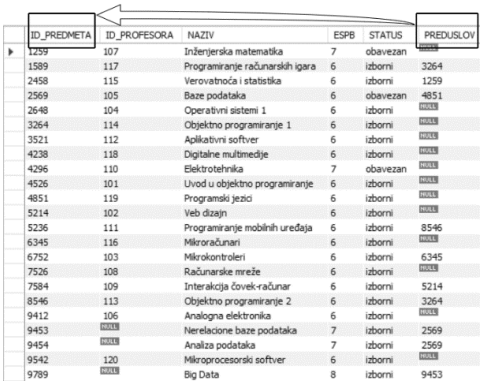
- Hijerarhijska struktura. Tabela koja sadrži podatke o zaposlenima, gde svaki zaposleni ima ID i ID svog nadređenog rukovodioca. Self join se može koristiti da bi se pronašli podaci o rukovodiocima i njihovim radnicima, ili da bi se za svakog radnika prikazali podaci o njegovom rukovodiocu.
- Samopovezane tabele. Ponekad tabela može imati samopovezane ili rekurentne odnose. Na primer, tabela koja sadrži podatke o korisnicima i njihovim referencama na druge korisnike unutar iste tabele. Self join se može koristiti da bi se identifikovali korisnici koji imaju zajedničke reference.
- Rekurzivne strukture. U nekim slučajevima, tabele mogu sadržati rekurzivne strukture, kao što su liste ili stabla. Self join se može koristiti za pretragu ili

analizu ovih rekurzivnih struktura, kao što je pretraga stabla porodice ili organizacione strukture.

U bazi podataka *studentski_servis*, ilustracija primene povezivanja tabele sa samom sobom je prikazana na primeru tabele *predmet*. Tabela predmet je modifikovana dodavanjem kolone *preduslov* sa podatkom o predmetu (ID) koji je preduslov određenom predmetu. U cilju jednostavnosti prikaza operacije SELF-JOIN, za opisani slučaj posmatranja, podrazumeva se da predmet može imati samo jedan predmet kao preduslov. Predmet može i ne mora imati preduslov.

Primer: Potrebno je prikazati nazive predmeta i nazive predmeta koji su njihovi preduslovi. SQL upit koji prikazuje postavljen zahtev koristi dve virtuelne instance tabele predmete: tabelu A koja sadrži podatke o predmetima i tabelu B koja sadrži podatke o predmetima koji su preduslovi nekim predmetima.

Na slici 7.2.4.1 je prikazan sadržaj tabele predmet. Strelica označava da kolona *A.PREDUSLOV* ukazuje na kolonu *B.ID_PREDMETA* odnosno može se posmatrati kao strani ključ tabele *A* koji ukazuje na primarni ključ *B.ID_PREDMETA* tabele *B*.



ID_PREDMETA	ID_PROFESORA	NAZIV	ESPB	STATUS	PREDUSLOV
1259	107	Inženjerska matematika	7	obavezan	NULL
1589	117	Programiranje računarskih igara	6	izborni	3264
2458	115	Verovatnoća i statistika	6	izborni	1259
2569	105	Baze podataka	6	obavezan	4851
2648	104	Operativni sistemi 1	6	izborni	NULL
3264	114	Objektno programiranje 1	6	izborni	NULL
3521	112	Aplikativni softver	6	izborni	NULL
4238	118	Digitalne multimedije	6	izborni	NULL
4296	110	Elektrotehnika	7	obavezan	NULL
4526	101	Uvod u objektno programiranje	6	izborni	NULL
4851	119	Programski jezici	6	izborni	NULL
5214	102	Veb dizajn	6	izborni	NULL
5236	111	Programiranje mobilnih uređaja	6	izborni	8546
6345	116	Mikroračunari	6	izborni	NULL
6752	103	Mikrokontroleri	6	izborni	6345
7526	108	Računarske mreže	6	izborni	NULL
7584	109	Interakcija čovek-računar	6	izborni	5214
8546	113	Objektno programiranje 2	6	izborni	3264
9412	106	Analogna elektronika	6	izborni	NULL
9453	NULL	Nerelacione baze podataka	7	izborni	2569
9454	NULL	Analiza podataka	7	izborni	2569
9542	120	Mikroprocesorski softver	6	izborni	NULL
9789	NULL	Big Data	8	izborni	9453

Slika 7.2.4.1 Prikaz povezivanja tabele predmet sa samom sobom

```
SELECT B.NAZIV as preduslov, A.NAZIV as predmet
from predmet A, predmet B
WHERE A.PREDUSLOV=B.ID_PREDMETA;
```

Rezulata SQL upita je prikazan na slici 7.2.4.2:

Preduslov	Predmet
Objektno programiranje 1	Programiranje računarskih igara
Inženjerska matematika	Verovatnoća i statistika
Programski jezici	Baze podataka
Objektno programiranje 2	Programiranje mobilnih uređaja
Mikroračunari	Mikrokontroleri
Veb dizajn	Interakcija čovek-računar
Objektno programiranje 1	Objektno programiranje 2
Baze podataka	Nerelacione baze podataka
Baze podataka	Analiza podataka
Nerelacione baze podataka	Big Data

Slika 7.2.4.2 Rezultujuća tabela nakon SELF JOIN-a nad tabelom predmet

Na osnovu SQL upita iz prethodnog primera može se videti da SELF-JOIN operacija podrazumeva spajanje tabele same sa sobom primenom spajanja po jednakosti u WHERE klauzuli odnosno da ne postoji klauzula SELF_JOIN.

7.3 UPOTREBA SPOJEVA NAD VIŠE OD DVE TABELE

U prethodnom delu teksta ovog poglavlja, operacija spajanja tabela je primenjan sao na dve tabele kako bi se dobile potrebne informacije iz baze podataka. Realni sistemi podrazumevaju izdvajanje podataka koji su smešteni i u više od sve tabele. Sve vrste spajanja mogu se primeniti u tim slučajevima.

Iz baza podataka *studentski_servisi,* koja se koristi kao reprezentativan primer, mogu se izdvojiti podaci koji su smešteni u više od dve tabele.

Na primer, podaci o predmetima koji su studenti izabrali da slušaju se čuvaju u tabeli student_predmet. Ova tabela predstavlja realizaciju veze više prema više između entiteta student i entiteta predmet.

Tabela zapisnik pored ocene i poena, sadrži podatke o identifikacionom broju studenta i identifikacionom broju ispita koji je polagao. U tabeli ispiti, za svaki identifikacioni broj ispita, zapis sadrži podatake od identifikacionom broju predmeta, identifikacionom broju ispitnog roka i datumu ispita.

Na osnovu prethodno navedenog, može se zaključiti da za prikaz zapisnika za predmet Baze podataka u januraskom ispitnom roku u spoju učestvuje više od dve tabele.

Primer: SQL upit ilustruje prikaz izabranih predmeta za studenta Lazara Markovića u školskoj 2018/19 godini.

```
SELECT NAZIV as 'Izabrani predmet'
FROM student S JOIN student_predmet SP
USING(ID_STUDENTA)
JOIN predmet p
USING(ID_PREDMETA)
WHERE S.IME='Lazar' AND S.PREZIME='Marković'
      AND SP.SKOLSKA GODINA='2018/19';
```

Za realizaciju SQL upita iz ovog primera izvršeno je spajanje po jednakosti primenom JOIN i USING klauzule. Spajanje tabele student sa tabelom student_predmet za posledicu ima dostupnost redova zapisa iz obe tabela, što uključuje i podatke o identifikacionom broju predmeta.

Na osnovu toga je moguće uključiti u spoj i tabelu predmet kako bi se prikazali podaci o nazivu predmeta. WHERE klauzulom izdvojeni su redovi zapisa koji odgovaraju zadatim uslovima u primeru.

Primer: Realizacija SQL upit koji ilustruje prikaz zapisnika predmeta Baze podataka za januarski ispitni rok školske 2019/20. godine uključuje spajanje tabela *zapisnik*, *ispit*, *ispitni_rok*, *predmet* i *student*. Spajanja tabela je realizovano preko jednakosti odgovarajućih ključeva upotrebom klauzule JOIN i USING. Za intuitivni prikaz podataka o prezimenu i imenu studenta kao i indeksu studenta, implementirana je funkcija CONCAT. U cilju bolje preglednosti i jasnoće upita korišćeni su alijasi za tabele.

```
SELECT concat(PREZIME,' ',IME) as 'Prezime i ime studenta ',
concat(SMER,'-',BROJ,'/',GODINA_UPISA) as indeks,
       OCENA, BODOVI
FROM zapisnik Z join ispit I
USING(ID_ISPITA)
JOIN predmet P
USING(ID_PREDMETA)
JOIN ispitni_rok IR
USING (ID_ROKA)
JOIN student S
USING (ID_STUDENTA)
WHERE P.NAZIV='Baze podataka' AND IR.NAZIV='Januar' AND
IR.SKOLSKA GOD='2019/20';
```

Rezultat SQL upita je prikazan na slici 7.3.1.

Prezime i ime studenta	indeks	OCENA	BODOVI
Milić Lena	IS-10/2018	9	87
Todorović Aleksa	RT-55/2019	10	97
Jevtić Filip	RT-29/2018	9	87
Lekić Janko	NRT-85/2019	9	87

Slika 7.3.1 Rezultujuća tabela

Primer: Realizacija SQL upita koji ilustruje prikaz podataka o studentima koji su izabrali određeni predmet, uključujući i one predmete koji nisu izabrani, obuhvata spajanje tabela *student*, *student_predmet* i *predmet*. Spajanje tabela *student* i *student_predmet* se ostvaruje korišćenjem klauzule JOIN i USING na odgovarajućim ključevima. Dodatno, spajanje uključuje tabelu *predmet* korišćenjem spoljašnjeg spoja RIGHT JOIN, kako bi se prikazali i predmeti koji nemaju dodeljene studente.

```
SELECT P.NAZIV as 'Predmet', CONCAT(PREZIME,' ',IME) as
'Prezime i ime studenta'
FROM student S JOIN student_predmet SP
USING(ID_STUDENTA)
RIGHT JOIN predmet P
USING(ID_PREDMETA)
ORDER BY 1;
```

Rezultat SQL upita je prikazan na slici 7.3.2.:

Predmet	Prezime i ime studenta
Analiza podataka	NULL
Analogna elektronika	Milutinović Nikola
Analogna elektronika	Mirković Vuk
Aplikativni softver	Miljković Sara
Aplikativni softver	Lazić Andrej
Aplikativni softver	Marković Lazar
Aplikativni softver	Markić Aleksandar
Aplikativni softver	Milutinović Nikola
Aplikativni softver	Grujić Violeta
Aplikativni softver	Todorović Aleksa
Aplikativni softver	Kostić Petra
Aplikativni softver	Stojković Mateja
Aplikativni softver	Marković Željko
Aplikativni softver	Ilić Nada
Aplikativni softver	Ivanović Lara
Baze podataka	Petrović Miloš
Baze podataka	Milić Lena
Baze podataka	Todorović Aleksa
Baze podataka	Jevtić Filip
Baze podataka	Lekić Janko
Big Data	NULL
Digitalne multimedije	Miljković Sara

Slika 7.3.2 Rezultujuća tabela

7.4 PODUPITI (UGNJEŽDENI UPITI)

Ugnježdeni upiti ili podupiti su SQL upiti koji se izvršavaju unutar glavnog SQL upita kako bi se dobio određeni rezultat. Podupiti se mogu koristiti u različitim klauzulama SQL upita i to: SELECT, WHERE, HAVING, FROM, INSERT, UPDATE, DELETE.

U većini primena, spoljašnji (glavni) upit koristi rezultat podupita kako bi obavio dalje operacije ili analize. Podupit se izvršava jednom pre spoljašnjeg upita.

Podupiti se najčešće koriste za:

- Upoređivanje podataka iz jedne tabele sa podacima iz druge tabele.
- Računanje agregatnih funkcija kako bi se moglo izvršiti postavljanje odgovarajućih uslova u WHERE klauzuli.
- Filtriranje rezultata spoljašnjeg upita na osnovu uslova iz drugih tabela.
- Izvršavanje složenih operacija dodavanja, ažuriranja ili brisanja podataka u okviru spoljašnjeg upita zasnovanih na rezultatima i uslovima postavljenim u podupitu.

Spoljašnji upit i podupit mogu biti povezani po vrednostima više kolona. Kada se upoređuju argumenti koji se sastoje od više kolona, oba argumenta moraju imati jednak broj kolona. Kolone koje se upoređuju moraju biti istog ili kompatibilnog tipa podataka. Prva kolona se upoređuje sa prvom, druga sa drugom, i tako dalje.

Rezultat podupita može biti konkretna vrednost, kada podupit vraća jedan red (poznato kao *single row subquery*), ili skup vrednosti. Kada se vraća skup vrednosti, koristi se operator IN za poređenje sa više vrednosti.

U zavisnosti od vrednosti koje vraćaju podupiti se dele na:

- **Skalarni (jednoredni) podupiti**. Vraćaju samo jednu vrednost. Spoljašnji upit obrađuje jednu vrednost iz unutrašnjeg podupita. Spoljašnji upit i unutrašnji podupit se povezuju isključivo upotrebom jednorednih operatora: =, >=, <, <>, <=.
- **Viševrednosni podupit**. Vraćaju rezultat sličan tabeli sa jednom kolonom. Spoljašnji upit obrađuje više vrednosti. Spoljašnji upit i unutrašnji podupit se povezuju pomoću višerednih operatora IN, ALL, ANY.
- **Tabelarni podupit**. Vraćaju više vrednosti u formi tabele sa kolonama i redovima. Pišu se iza ključne reči FROM.

Prilikom pisanja podupita neophodno je ispoštovati sledeća pravila:

- Podupiti se navode u zagradi.
- U uslovu poređenja, podupiti su na desnoj strani.
- Spoljašnji (glavni) i unutrašnji upit (podupit) mogu uzeti podatke iz različitih tabela.
- Tip kolone navedene u podupitu mora biti isti kao tip izraza na levoj strani operatora.
- Samo jedna klauzula ORDER BY može biti uključena u jednu SELECTrečenicu i ona mora biti poslednja klauzula spoljašnjeg upita.
- Podupit ne može imati sopstvenu ORDER BY klauzulu.
- Jedino ograničenje broja podupita je veličina bafera odnosno memorije koja se koristi za čuvanje SQL upita pre nego što se oni izvrše.

7.4.1 PODUPITI U WHERE KLAUZULI

Podupiti u WHERE klauzuli su podupiti koji se koriste za generisanje vrednosti koje se zatim koriste za postavljanje uslova. U većini slučajeva, vrednosti dobijene izvršavanjem podupita nije moguće direktno postaviti kao uslov spoljašnjeg upita.

Primer: SQL upit ilustruje upotrebu podupita u WHERE klauzuli na primeru tabele *student*. Ovaj primer prikazuje upotrebu agregatnih funkcija za postavljanje uslova.

```
SELECT IME, PREZIME
FROM student
WHERE GODINA_UPISA = (SELECT MAX(GODINA_UPISA)
                      FROM student);
```

U ovom primeru, podupit *SELECT MAX(GODINA_UPISA) FROM student* se izvršava unutar WHERE klauzule i vraća maksimalnu godinu upisa studenata.

Spoljašnji upit zatim filtrira rezultate tako da prikaže samo studente kod kojih godina upisa odgovara vrednosti maksimalne godine.

Primer: SQL upit ilustruje upotrebu podupita u WHERE klauzuli za generisanje skupa vrednosti koje se zatim koriste u uslovu specificiranom u spoljašnjem upitu.

```
SELECT IME, PREZIME,SMER
FROM student
WHERE SMER = (SELECT SMER
               FROM student
               WHERE IME='Lazar' AND PREZIME='Marković')
AND IME<>'Lazar' AND PREZIME<>'Marković';
```

U ovom primeru, podupit vraća vrednost koja predstavlja naziv smera koji studira student Lazar Marković. Vrednost dobijena iz podupita se koristi u spoljašnjem upitu u WHERE klauzuli specificiranjem uslova na osnovu koga se izdvajaju redovi zapisa o studenatima koji studiraju isti smer kao i student Lazar Marković. Iz prikaza se isključuje Lazar Marković što je postignuto postavljanjem dodatnog uslova *IME<>'Lazar' AND PREZIME<>'Marković'* u spoljašnjem upitu.

Primer: Upotrebom podupita u WHERE klauzuli, SQL upit prikazuje podatke o profesorima čija je cena po času manja od prosečne cene časa.

```
SELECT PREZIME, IME
FROM profesor
WHERE CENA_PO_CASU<(SELECT AVG(CENA_PO_CASU)
                    FROM profesor);
```

Podupit unutar WHERE klauzule se koristi kako bi se uporedila cenu po času svakog profesora sa prosečnom cenom po času, koja se dobija izračunavanjem AVG funkcije na koloni CENA_PO_CASU u tabeli profesor.

Primer: SQL upit koristi podupit koji vraća skup vrednosti i u okviru svoje WHERE klauzuli primenjuje mehanizam postavljanja novog podupita.

```
SELECT PREZIME,IME
FROM student
WHERE ID_STUDENTA IN
                   (SELECT ID_STUDENTA
                    FROM student_predmet
                    WHERE ID_PREDMETA =
                       (SELECT ID_PREDMETA
                        FROM predmet
                        WHERE NAZIV="Inženjerska matematika")
                    AND SKOLSKA_GODINA="2018/19")
```

U ovom primeru spoljašnji upit koristi podupit unutar WHERE klauzule sa IN operatorom. Podupit vraća identifikacione brojeve studenata koji su upisani na predmet "Inženjerska matematika" tokom školske godine 2018/19.

Operatori ANY i ALL se koriste u podupitima za poređenje vrednosti sa skupom rezultata koji se vraćaju iz podupita. Operator ANY vraća vrednost TRUE ako bar jedna od vrednosti iz podupita zadovoljava uslov. Na primer, > ANY znači "veće od bilo koje vrednosti u podupitu". Operator ALL vraća TRUE samo ako sve vrednosti iz podupita zadovoljavaju uslov. Na primer, > ALL znači "veće od svih vrednosti u podupitu".

Upotreba =ANY može se zameniti sa IN operatorom, što može olakšati čitanje i razumevanje upita. Međutim, upotreba =ALL nema baš puno smisla, jer jedna vrednost ne može biti identična sa svakom vrednošću iz skupa vrednosti.

Primer: SQL upit koristi podupit sa ANY operatorom za prikaz podataka o profesorima čija je godina zaposlenja manja od bilo koje godine zaposlenja profesora koji primaju nagradu.

```
SELECT PREZIME, IME
FROM profesor
WHERE YEAR(DATUM_ZAP)<ANY(SELECT DISTINCT YEAR(DATUM_ZAP)
                          FROM profesor
                          WHERE NAGRADA IS NOT NULL)
```

U ovom primeru koristi se podupit unutar WHERE klauzule sa ANY operatorom. Ovaj podupit izdvaja sve jedinstvene godine zaposlenja profesora koji su dobili nagradu, a zatim upoređuje godinu zaposlenja svakog profesora sa ovim godinama. SQL upit prikazuje podatke o profesorima čija je datum zaposlenja stariji od bilo kog datuma zaposlenja profesora koji su dobili nagradu.

7.4.2 MULTIPLE-COLUMN PODUPITI

Multiple-Column podupiti se koriste kada želimo da uporedimo više od jedne kolone u podupitu. To znači da uslov u WHERE klauzuli sadrži više od jedne kolone za poređenje sa rezultatima podupita. Spoljašnji i unutrašnji podupit su povezani argumentima koji sadrže vrednosti više kolona, što omogućava *pair-wise* upoređivanje. Prilikom *pair-wise* upoređivanja, prva kolona argumenta iz spoljašnjeg upita upoređuje se sa prvom kolonom argumenta u podupitu, druga sa drugom, i tako dalje.

Takođe, kolone koje se upoređuju moraju biti istog ili kompatibilnog tipa podataka. *Non-pair-wise multiple-column* podupiti takođe koriste više od jedne kolone u podupitu, ali ih porede ih jednu po jednu. Poređenja se izvode u različitim podupitima, obično po jedan podupit po koloni koja se treba uporediti.

Primer: SQL upit koristi *pair-wise* upoređivanje sa podupitom za prikaz naziva predmeta koji imaju isti status i broj ESPB bodova kao i predmeti koji imaju definisane preduslove.

```
SELECT NAZIV
FROM predmet
WHERE (STATUS, ESPB) IN
            (SELECT STATUS, ESPB
             FROM predmet
             WHERE PREDUSLOV IS NOT NULL);
```

U ovom primeru koristi se *multiple-column* podupit unutar WHERE klauzule sa IN operatorom. Podupit unutar IN operatora vraća status i broj ESPB bodova za predmete koji imaju definisane preduslove. Spoljašnji upit zatim filtrira predmete tako da prikaže samo one čiji su status i broj ESPB bodova upareni sa rezultatima podupita.

Primer: SQL upit koristi *non-pair-wise* upoređivanje za prikaz podataka o profesorima koji su se zaposlili u godini kada su se zaposlili profesori koji imaju između 25 i 30 časova i cena po času im je manja od prosečne cene časa.

```
SELECT PREZIME, IME, CENA_PO_CASU
FROM profesor
WHERE YEAR(DATUM_ZAP)IN
                        (SELECT YEAR(DATUM_ZAP)
                         FROM profesor
                         WHERE BROJ_CASOVA between 25 and 30)

AND CENA_PO_CASU  <   (SELECT AVG(CENA_PO_CASU)
                       FROM profesor);
```

U ovom primeru primene *non-pair-wise multiple-column* podupita, prvi podupit *SELECT YEAR(DATUM_ZAP) FROM profesor WHERE BROJ_CASOVA BETWEEN 25 AND 30* vraća godine zaposlenja profesora koji imaju između 25 i 30 časova, a spoljašnji upit filtrira profesore čija je godina zaposlenja uključena u rezultat ovog podupita.

Drugi podupit *SELECT AVG(CENA_PO_CASU) FROM profesor* računa prosečnu cenu po času svih profesora, a spoljašnji upit filtrira profesore čija je cena po času manja od ove prosečne vrednosti.

SQL upit će vratiti prezime, ime i cenu po času profesora koji ispunjavaju oba ova uslova: imaju godinu zaposlenja kao i profesori koji imaju broj časova u rasponu od 25 do 30 časova i cena po času je manja od prosečne cene po času za sve profesore.

7.4.3 KORELISANI PODUPITI

Korelisani podupiti su vrsta podupita u SQL-u u kojima se rezultat unutrašnjeg podupita koristi u spoljašnjem upitu, pri čemu se ta vrednost dinamički menja za svaki red rezultata spoljašnjeg upita. Vrednosti iz spoljašnjeg upita se koriste kao parametri za unutrašnji (korelisani) podupit, omogućavajući vezu između dve tabele. Obrada korelisanog podupita se izvršava za svaki red rezultujuće tabele iz spoljašnjeg upita. Korelisani podupiti su korisni kada je potrebno izvršiti upit koji zavisi od vrednosti redova koji se trenutno obrađuju.

***Primer**:* SQL upit koristi korelisani podupit za prikaz naziva predmeta kod kojih predmetni nastavnici imaju zvanje magistra.

```
SELECT PR.NAZIV
FROM predmet PR
WHERE "mr" IN
          (SELECT PF.ZVANJE
           FROM PROFESOR PF
           WHERE PR.ID_PROFESORA=PF.ID_PROFESORA);
```

U ovom primeru koristi se korelirani podupit za izdvajanje naziva predmeta koji su predavani od strane profesora sa zvanjem "mr". Unutrašnji podupit se ne izvršava pre spoljašnjeg i zavisi od promenljive *PR.ID_PROFESORA*. Za prvi red tabele *predmet* izdvaja se vrednost *PR.ID_PROFESORA* i to je 107. Generiše se podupit *SELECT PF.ZVANJE FROM PROFESOR PF WHERE PR.ID_PROFESORA=107* koji vraća zvanje profesora čiji je identifikacioni broj 107. Rezultat je vrednost *"dr"* koja ne zadovoljava postavljen uslov u WHERE klauzuli spoljašnjeg upita. Prelazi se na sledeći red tabele *predmet*. Prikazuju se samo redovi zapisa tabele *predmet* koji zadovoljavaju postavljen uslov za zvanje profesora. Postupak se ponavlja sve do poslednjeg reda tabele *predmet*.

U korelisanim podupitima koristi se i unarni logički operator EXISTS. Za ispravno funkcionisanje operatora EXISTS bitno je u WHERE klauzuli spoljašnjeg upita pravilno povezati (korelisati) ugnježđeni upit sa spoljašnjim upitom kako bi se proverilo postojanje redova u podupitu. Operator EXISTS u SQL-u se koristi za proveru postojanja rezultata podupita. Ako podupit vraća bilo koji red, EXISTS će vratiti vrednost TRUE, u suprotnom vratit će FALSE.

Primer: SQL upit koristi *EXISTS* operator za povezivanje spoljašnjeg upita i korelisanog podupita. Rezultat izvršavanja ovog upita predstavlja prikaz podataka o profesorima koji imaju dodeljen bar jedan predmet.

```
SELECT *
FROM profesor
WHERE EXISTS
       (SELECT ID_PROFESORA
        FROM predmet
        WHERE predmet.ID_PROFESORA=profesor.ID_PROFESORA);
```

U ovom primeru SQL upit koristi operator EXISTS kako bi proverio postojanje rezultata u unutrašnjem podupitu. U podupitu se postavlja uslov korelisanja koji je u ovom slučaju *predmet.ID_PROFESORA=profesor.ID_PROFESORA*. Na osnovu postavljenog uslova korelisanja upit proverava da li postoji bar jedan red u tabeli *predmet* za svakog profesora iz tabele *profesor*. Ako postoji takav red, EXISTS vraća TRUE, u suprotnom će vratiti FALSE. Ako EXISTS vrati TRUE, tada se prikazuju svi podaci iz tabele *profesor* za redove gde postoji odgovarajući red u tabeli *predmet*.

Ukoliko se izostavi uslov korelisanja, upit ne vraća tražene podatke.

```
SELECT *
FROM profesor
WHERE EXISTS
       (SELECT ID_PROFESORA
        FROM predmet);
```

Nekorelisani upita vraća podatke o svim profesorima bez obzira na to da li imaju dodeljen predmet ili ne. Problem je nepostojanje korelacije između spoljašnjeg upita i ugnježdenog upita. U ovom slučaju, kad god se proverava da li profesor ima dodeljen predmet ili ne, ugnježdeni upit uvek vraća isti rezultat (sve predmete).

Primer: Modifikacijom SQL upita iz prethodnog primera može se dobiti upit koji prikazuje podatke o profesorima koji nemaju dodeljene predmete.

```
SELECT *
FROM profesor
WHERE NOT EXISTS
       (SELECT ID_PROFESORA
        FROM predmet
        WHERE predmet.ID_PROFESORA=profesor.ID_PROFESORA);
```

SQL upit iz ovog primera može biti modifikovan upotrebom konstante vrednosti 0 u SELECT listi ugnježdenog upita. Ovaj način se može primeniti kada nisu neophodni podaci iz ugnježdenog upita već je samo potrebno proveriti da li oni postoje ili ne.

```
SELECT *
FROM profesor
WHERE NOT EXISTS
       (SELECT 0
        FROM predmet
        WHERE predmet.ID_PROFESORA=profesor.ID_PROFESORA);
```

7.4.4 PODUPITI U FROM KLAUZULI

Mehanizam virtuelnih pogleda omogućava ugnježđavanje jednog SQL upita u FROM klauzuli drugog SQL upita. Ugnježđeni SQL upit se u tom slučaju tretira kao i bilo koja druga tabela iz baze podataka. Rezultat podupita se smatra virtuelnom tabelom iz koje se izdvajaju n-torke (redovi zapisa). Važno je napomenuti da nazivi kolona u spoljašnjem upitu moraju biti podskup skupa naziva kolona u unutrašnjem podupitu.

Primer: Za formiranje spiska predmeta koji imaju više od 10 prijavljenih studenata moguće je iskoristiti mehanizam virtuelnih pogleda.

```
SELECT predmet.NAZIV, predmet.STATUS, broj.BROJ_STUDENATA
FROM predmet,
     (select id_predmeta, count(*) BROJ_STUDENATA
      from student_predmet group by ID_PREDMETA) as broj
WHERE  predmet.ID_PREDMETA=broj.ID_PREDMETA
AND broj.BROJ_STUDENATA>10;
```

U ovom primeru ugnježđeni upit generiše virtuelnu tabelu za prikaz broj studenata po predmetu na osnovu podataka iz tabele *student_predmet*. Spoljašnji upita kombinuje rezultat unutrašnjeg upita sa tabelom *predmet*. U FROM klauzuli, virtuelnoj tabeli je dodeljen alijas *broj*. U WHERE klauzuli se vrši povezivanje redova iz tabele *predmet* sa odgovarajućim redovima virtuelne tabele *broj*. Uslov spajanja *predmet.ID_PREDMETA = broj.ID_PREDMETA* obezbeđuje povezivanje odgovarajućeg predmeta sa brojem prijavljenih studenata na njemu. Uslov u WHERE klauzuli spoljašnjeg upita, *broj.BROJ_STUDENATA > 10*, obezbeđuje prikaz podataka samo o predmetima koji imaju više od 10 prijavljenih studenata.

SQL upit iz ovog primera može se rešiti i spajanjem po jednakosti ključeva tabela *predmet* i *student_predmet* primenom JOIN i USING klauzula.

```
SELECT NAZIV, STATUS, COUNT(*) BROJ_STUDENATA
FROM predmet JOIN student_predmet
USING (ID_PREDMETA)
GROUP BY NAZIV, STATUS
HAVING COUNT(*)>10;
```

7.4.5 PODUPITI U SELECT KLAUZULI

U SELECT klauzuli, podupiti se koriste kako bi se dobio dodatni podatak za svaki red u rezultujućoj tabeli. Podupiti u SELECT klauzuli često rade nad podacima iz iste ili srodne tabele kao i spoljšanji upit, jer moraju biti vezani za trenutni red u rezultujućoj tabeli. Korisni su za prikaz agregiranih podataka ili podataka iz drugih redova u rezultatu upita.

Primer: SQL upit ilustruje primenu podupita u SELECT klauzuli za izračunavanje broja prijavljenih studenata za svaki predmet. U ovom primeru je prikazana upotreba podupita u SELECT klauzuli za rad sa podacima iz tabela povezanih preko jednakosti ključeva.

```
SELECT NAZIV, STATUS,
(SELECT COUNT(*) FROM student_predmet WHERE
ID_PREDMETA=p.ID_PREDMETA) as BROJ_STUDENATA
FROM predmet p
ORDER BY 3 DESC;
```

7.4.6 PODUPITI U HAVING KLAUZULI

Podupiti u HAVING klauzuli se koriste za filtriranje grupisanih podataka na osnovu agregiranih vrednosti. Ova tehnika omogućava dodatno filtriranje grupa podataka na osnovu uslova koji se primenjuju na rezultate agregacije. HAVING klauzula omogućava izdvajanje grupa redova prema postavljenim uslovima koji se odnose na rezultate agregacije, a koji nisu direktno podržani u WHERE klauzuli.

Primer: SQL upit ilustruje primenu podupita u HAVING klauzuli za prikaz podataka o broju profesora po zvanjima i prosečnoj ceni časa samo za zvanja na kojima je prosečna cena časa veća od prosečne cene časa na nivou fakulteta.

```
SELECT ZVANJE,COUNT(*),AVG(CENA_PO_CASU)
FROM profesor
GROUP BY ZVANJE
HAVING AVG(CENA_PO_CASU)>
                  (SELECT AVG(CENA_PO_CASU) FROM profesor)
```

7.5 AGREGACIJA PODATAKA IZ VIŠE TABELA

Ukoliko SQL upit uključuje više tabela i potrebno je primeniti neku od agregatnih funkcija, važno je da spojevi ne naruše agregaciju. To se može postići korišćenjem odgovarajućih JOIN operacija ili primenom podupita i pažljivim postavljanjem uslova spoja kako bi se osiguralo da se agregatne funkcije primenjuju na ispravne skupove podataka.

Primer: Prikazati podatke o studentima koji su na ispitu iz predmeta Baze podataka u januarskom ispitnom roku školske 2019/20 osvojili najveći broj poena.

```
SELECT PREZIME, IME, OCENA, BODOVI
FROM student s join  zapisnik z
USING (ID_STUDENTA)
JOIN ispit i
USING (ID_ISPITA)
JOIN ispitni_rok ir
USING (ID_ROKA)
JOIN predmet p
USING (ID_PREDMETA)
WHERE p.NAZIV="Baze podataka" AND ir.NAZIV="Januar" AND
SKOLSKA_GOD="2019/20"
AND BODOVI = (SELECT max(BODOVI)
             FROM ZAPISNIK z JOIN ISPIT i
             USING (ID_ISPITA)
             JOIN ispitni_rok ir
                       USING (ID_ROKA)
                       JOIN predmet p
                       USING (ID_PREDMETA)
             WHERE p.NAZIV="Baze podataka" AND
ir.NAZIV="Januar" AND SKOLSKA_GOD="2019/20");
```

Primer: SQL upit ilustruje prikaz prosečne ocene po predmetima, ispitnom roku, školskoj godini samo za predmete na kojima je prosečna ocena veća od 8.

```
SELECT p.NAZIV,ir.NAZIV,ir.SKOLSKA_GOD, AVG(OCENA)
FROM zapisnik z JOIN ispit i
USING (ID_ISPITA)
JOIN ispitni_rok ir
USING (ID_ROKA)
JOIN predmet p
USING (ID_PREDMETA)
WHERE OCENA>5
GROUP BY p.ID_PREDMETA, ir.ID_ROKA, ir.SKOLSKA_GOD
HAVING AVG(OCENA)>8
ORDER BY 1;
```

Primer: SQL upit ilustruje prikaz ukupnih broja ESPB poena za svakog studenta u školskoj 2018/19. godini.

```
SELECT PREZIME, IME, SUM(ESPB)
FROM student s JOIN student_predmet sp
USING (ID_STUDENTA)
JOIN predmet p
USING(ID_PREDMETA)
WHERE SKOLSKA_GODINA="2018/19"
GROUP BY PREZIME,IME;
```

U primeru SQL upita, u GROUP BY klauzuli su navedene kolone *PREZIME* i *IME* koje se nalaze i u SELECT listi, a nisu u funkciji agregacije. Međutim, umesto imena i prezimena, grupisanje se može vršiti po identifikacionom broju studenta. Budući da u spoju učestvuju tabele *student* i *student_predmet*, potrebno je koristiti alijase tabela prilikom navođenja kolo*ne ID_STUDENTA* kako bi se izbegla ambiguitetna referenca.

Primer: SQL upit ilustruje prikaz podataka o profesorima, predmetima koji predaju, broju prijavljenih studenta u školskoj 2019/20. godini, ali samo ako je na predmetu prijavljeno više od četri studenta.

```
SELECT PREZIME,IME, NAZIV, COUNT(*)
FROM profesor pf JOIN predmet pd
USING(ID_PROFESORA)
JOIN student_predmet sp
USING (ID_PREDMETA)
WHERE SKOLSKA_GODINA="2019/20"
GROUP BY pf.ID_PROFESORA, sp.ID_PREDMETA
HAVING COUNT(*)>4;
```

7.6 KOJI PRISTUP KORISTITI ZA POVEZIVANJE TABELA?

Odluka o tome koji pristup je "bolji" zavisi od različitih faktora, uključujući strukturu baze podataka, efikasnost izvršavanja upita, čitljivost koda i preference programera.

Prilikom donošenja odluke treba uzeti u obzir sledeće:

– **Performanse**. U većini slučajeva, upit sa JOIN klauzulom je brži jer omogućava bazi podataka da optimizuje izvršavanje upita. Međutim, to nije uvek slučaj. U nekim situacijama ugnježđeni upiti mogu biti efikasniji, posebno ako je baza podataka manja ili ako postoje određene optimizacije koje se mogu primeniti na taj pristup.

- **Čitljivost koda**. SQL kod sa JOIN klauzulom je u većini slučajeva lakši za čitanje i razumevanje, posebno ako imate više tabela koje se spajaju. Ugnježđeni upiti mogu postati složeniji i teže ih je pratiti ako se koriste u većim i kompleksnijim upitima.
- **Prilagodljivost**. Upit sa JOIN klauzulom može biti fleksibilniji i omogućava dodavanje dodatnih uslova za spajanje ili filtriranje podataka. Sa druge strane, ugnježđeni upiti mogu biti korisni ako je potrebno dodatno filtriranje ili manipulacija podacima u podupitu.
- **Standardi**. U nekim slučajevima, organizacija može imati standarde ili praksu koja preferira jedan pristup nad drugim. Važno je razmotriti takve smernice prilikom izbora pristupa.

U suštini, nema univerzalnog odgovora na pitanje koji pristup je "bolji". Preporaka je eksperimentisanje sa oba pristupa u vašem okruženju kako biste utvrdili koji pristup najbolje odgovara potrebama i zahtevima određenog projekta.

7.7 PITANJA I ZADACI

1. Napisati SQL upit za prikaz datuma i usluge specijalističkog pregleda fizijatra dr Mirka Mirkovića za pacijenta Marka Markovića.
2. Napisati SQL upit za prikaz obavljenih pregleda kod doktora Milice Jovanović.
3. Napisati SQL upit za prikaz podataka o lekaru sa najviše obavljenh pregleda u 2022. godini.
4. Napisati SQL upit za prikaz obavljenih pregleda kod doktora Milice Jovanović.
5. Napisati SQL upit za prikaz podataka o pacijentima doktora Ane Milojčević.
6. Napisati SQL upit za prikaz iznosa za plaćanje obavljenih usluga vađenja krvi i očnog specijalističkog pregleda pacijenta Gordane Smiljanić koji su obavljeni 22.03.2023.
7. Napisati SQL upit za prikaz najskuplje vrste usluge.
8. Napisati SQL upit za prikaz podataka o pacijentu sa najvećim brojem obavljenih pregleda.
9. Napisati SQL upit za prikaz podataka o lekarima i o pacijentima koje su pregledali, ali samo za lekare koji su imali više od pet obavljenih pregleda.
10. Napisati SQL upit za prikaz podataka o broju lekara po oblastima specijalizacije.

8 NAREDBE ZA MODIFIKACIJU PODATAKA

Cilj ovog poglavlja je fokusiran na objašnjenje SQL DML naredbi koje omogućavaju osnovne operacije ažuriranja podataka u relacionim bazama podataka. Definisane su naredbe INSERT, UPDATE i DELETE. Razmatrani su principi implementacije akcionih SQL upita zasnovanih na prethodno navedenim naredbama.

8.1 UVOD

Modifikacija podataka u relacionoj bazi podataka se realizuje primenom DML naredbi, koje omogućavaju osnovne operacije ažuriranja podataka:

- **Dodavanje novih podataka**. Ova operacija omogućava dodavanje novih redova zapisa u tabelu kao i kopiranje postojećih.
- **Ažuriranje podataka.** Ova operacija omogućava izmenu vrednosti kolona u postojećim redovima tabele.
- **Brisanje podataka.** Ova operacija omogućava brisanje postojećih redova iz tabele.

Promene na nivou redova tabele obuhvataju ažuriranje podataka, a to se obično vrši putem aplikacija. Procedura zaštite integriteta podataka uključuje niz pravila i ograničenja koji se primenjuju na bazi podataka kako bi se osigurala konzistentnost i tačnost podataka. Za implementaciju naredbi INSERT, DELETE i UPDATE mogu se koristiti procedure.

Ažuriranje na nivou jedne tabele može se izvršiti direktno na osnovu promena u toj tabeli ili na osnovu vrednosti iz druge tabele. Pri ažuriranju tabele na osnovu vrednosti iz druge tabele, može se primeniti WHERE klauzula za tabelu koja se ažurira, zajedno sa korelisanim podupitom.

8.2 DODAVANJE NOVIH PODATAKA

SQL naredba INSERT INTO se koristi za dodavanje novih redova u tabelu relacione baze podataka. Vrednosti kolona se definišu navođenjem konkretnih vrednosti ili korišćenjem rezultata SQL upita.

Postoje različiti oblici INSERT INTO naredbe, zavisno od načina zadavanja vrednosti kolona:

- **Navođenje svih vrednosti za jedan red.** Navode se sve vrednosti za sve kolone jednog reda.
- **Navođenje vrednosti za određene kolone jednog reda.** Navode se vrednosti samo za određene kolone reda.
- **Kopiranje podataka iz druge tabele.** Kreira se nova tabela sa istim kolonama kao i tabela iz koje se kopiraju podaci, a zatim se podaci kopiraju u novu tabelu.

Osnovni oblik naredbe INSERT INTO podrzumeva navođenje svih vrednosti za jedan red

```
INSERT INTO ime_tabele (kolona1, kolona2, kolona3)
VALUES (vrednost1, vrednost2, vrednost3);
```

Redosled vrednosti kolona u listi VALUES mora da bude isti kao što je u listi INSERT INTO. Ukoliko se u INSERT...INTO naredbi zadaje lista kolona, moguće je promeniti redosled zadavanja kolona u odnosu na onaj koji je specificiran prilikom kreiranja tabele.

Primer: SQL upit ilustruje dodavanje podataka o novo profesoru.

```
INSERT INTO profesor(ID_PROFESORA,IME,PREZIME,ZVANJE,
DATUM_ZAP,BROJ_CASOVA,CENA_PO_CASU)
VALUES (123, 'Aleksandra', 'Marnojlović', 'docent', '2024-03-
04', 12, 3890);
```

U ovom primeru naredbom INSERT INTO dodati su podaci o prefesoru Aleksandri Manojlović koja ima zvanje docenta, zaposlena je 04.03.2024. godine, ima 12 časova i cena po času je 3890.

Primer: Potrebno je dodati podatke o tri studenta u tabelu student.

```
INSERT INTO student (ID_STUDENTA, IME, PREZIME, SMER, BROJ,
GODINA_UPISA, JMBG)
VALUES
('31', 'Mila', 'Jonić', 'RT', 12, 2024, NULL),
('32', 'Ana', 'Simić', 'IS', 23, 2024, 1304041345678),
('33', 'Stefan', 'Stefanović', 'NRT', 44, 2024, NULL);
```

U ovom primeru je prikazan postupak dodavanja više redova zapisa podataka u okviru jedne INSERT INTO naredbe. Kolone su navedene istim redosledom koji je i specificiran prilikom kreiranja tabele *student*. Kolona JMBG dozvoljava NULL vrednosti što omogućava upis samo poznate vrednosti u jednom od dodatih redova zapisa.

Lista kolona u pojedinim slučajevima ne mora biti kompletna. Iz liste se mogu izostaviti kolone kod kojih nije definisano NOT NULL ograničenje i kolone koje imaju definisano DEFAULT ograničenje. Za nedostajuće kolone, podrazumevana vrednost će biti ili vrednost definisana DEFAULT ograničenjem ili NULL vrednost ukoliko DEFAULT ograničenje ne postoji.

Primer: SQL kod ilustruje dodavanje zapisa o profesoru samo sa podacima za kolone za koje je postavljeno ograničenje NOT NULL.

```
INSERT INTO profesor (ID_PROFESORA,IME,PREZIME,ZVANJE,
DATUM_ZAP)VALUES
(124,'Jelena', 'Simić', 'redovni','2024-01-02');
```

U SQL upitu, redosled vrednosti kolona u listi VALUES mora da odgovara redosledu navedenom u listi INSERT INTO. Ovo je važno jer SQL engine upita očekuje da svaka vrednost ide u odgovarajuću kolonu na osnovu redosleda navedenog u INSERT INTO delu upita. Ako redosled nije isti, to može dovesti do grešaka ili neželjenog ponašanja prilikom umetanja podataka.

Primer: SQL kod ilustruje izmenu redosleda kolona u INSERT INTO listi, a u skladu sa tim i redosleda odgovarajućih vrednosti u listi VALUES.

```
INSERT INTO profesor (ID_PROFESORA,PREZIME,IME,ZVANJE,
DATUM_ZAP)VALUES
(125,'Milić','Milica','mr','2024-01-15');
```

Ako u INSERT INTO delu upita nisu navedeni nazivi kolona, vrednosti u listi VALUES moraju biti navedene redosledom specificiranim prilikom kreiranja tabele. Za kolone koje dozvoljavaju NULL vrednost i trenutno nemaju poznatu vrednost, na odgovarajućoj poziciji u listi VALUES mora biti navedena NULL vrednost.

Primer: SQL kod ilustruje naredbu primer primene naredbe INSERT INTO bez navedenih naziva kolona.

```
INSERT INTO profesor VALUES
(126,'Janko','Ilić','mr','2024-02-19',NULL,NULL,NULL);
```

U SQL-u, mogu se dodavati redovi u tabelu i na osnovu rezultata SELECT upita korišćenjem INSERT INTO naredbe zajedno sa SELECT naredbom. Ovo se naziva INSERT INTO SELECT sintaksa:

```
INSERT INTO tabela (kolona1, kolona2,...)
<SELECT upit>;
```

Primer: Za potrebe ilustracije primene INSERT INTO SELECT naredbe kreirana je tabela *profesori_stari* sa kolonoma ID, PREZIME, IME, ZVANJE, DATUM_Z.

```
CREATE TABLE profesori_stari (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    PREZIME VARCHAR(50) NOT NULL,
    IME VARCHAR(50) NOT NULL,
    ZVANJE VARCHAR(10),
    DATUM_Z DATE
);
```

SQL upit ilustruje dodavanja podataka o profesorima, koji su zaposleni pre 2000. godine, u tabelu *profesori_stari* korišćenjem INSERT INTO SELECT upita.

```
INSERT INTO profesori_stari
SELECT ID_PROFESORA,PREZIME, IME, ZVANJE, DATUM_ZAP
FROM profesor
WHERE YEAR(DATUM ZAP)<2000;
```

Za automatsko kreiranje nove tabele i kopiranje podataka iz postojeće u novokreiranu tabelu, u MySQL se koristi SELECT naredba u kombinaciji sa CREATE TABLE.

Primer: SQL kod ilustruje postupak kreiranja nove tabele na osnovu podataka koji vraća naredba SELECT. Bitno je uočiti da kreirana tabela nema nijedno ograničenja, uključujući i primarni ključ.

```
CREATE TABLE studenti_rt
SELECT * FROM student
WHERE SMER="RT";
```

8.3 AŽURIRANJE PODATAKA

SQL naredba UPDATE ..SET se koristi za ažuriranje podataka u tabelama relacionih baza podataka. Može se promeniti vrednost jedne ili više kolona unutar jednog reda zapisa. Takođe je moguće promeniti i vrednost jedne kolone unutar više redova zapisa.

Osnovni oblik naredba UPDATE ..SET je dat u nastavku:

```
UPDATE ime_tabele
SET kolona1=vrednost1[,kolona2=vrednost2, ...]
WHERE [lista uslova];
```

Klauzulom SET se definiše kolona koja menja vrednost na osnovu određenog izraza. Izraz može biti konstantna vrednost, vrednost nekog izraza ili vrednost koju vraća SQL upit. Ako se koristi WHERE klauzula, ažuriranje se vrši samo za redove koji ispunjavaju navedeni uslov.

Primer: SQL upit ilustruje primenu UPDATE SET naredbe za upis podatka koji nedostaje u odgovarajućem redu zapisa koji je specificiran uslovom u WHERE klauzuli.

```
UPDATE student
SET JMBG="1304051678123"
WHERE IME="Lazar" AND PREZIME="Marković";
```

Napomena: Prilikom korišćenja UPDATE...SET naredbe ukoliko se u izostavi ili nije dobro postavljen uslov u WHERE klauzuli izmene će biti primenjene na sve redove tabele navedene u UPDATE delu naredbe.

Primer: SQL upit ilustruje primenu UPDATE SET naredbe za istovremenu izmenu vrednosti većeg broja kolona koje su navedene u SET klauzuli.

```
UPDATE profesor
SET BROJ_CASOVA=15, CENA_PO_CASU=2500
WHERE DATUM_ZAP>"2024-01-14" AND ZVANJE="mr";
```

Primer: SQL upit ilustruje primenu UPDATE SET naredbe za izmenu izborne liste predmeta studenta Nenada Brankovića u školskoj godini 2018/19.

```
UPDATE student_predmet
SET ID_PREDMETA=(SELECT ID_PREDMETA FROM predmet
                 WHERE NAZIV="Baze podataka")
WHERE ID_STUDENTA=(SELECT ID_STUDENTA FROM student
                   WHERE IME="Nenad" AND PREZIME="Branković")
AND ID_PREDMETA=(SELECT ID_PREDMETA FROM predmet
                 WHERE naziv="Programski jezici")
AND SKOLSKA_GODINA="2018/19";
```

U ovom primeru, klauzulom SET je navedena kolona ID_PREDMETA koja se ažurira na osnovu vrednosti koju vraća SQL podupit *SELECT ID_PREDMETA FROM predmet WHERE NAZIV="Baze podataka"*. U WHERE klauzuli, ažuriranje se vrši na osnovu postavljenog uslova u WHERE klauzuli za zapis koji se odnosi na studenta Nenada Brankovića i prethodno izabran predmet Programski jezici u školskoj 2018/19 godini. Postavljeni uslov je definisan korišćenjem dva podupita. Jedan podupit vraća identifikacioni broj studenta Nenada Brankovića iz tabele

student, dok se drugi podupit koristi za izdvajanje identifikacionog broja predmeta Programski jezici iz tabele *predmet*, koji se menja sa predmetom Baze podataka.

Od verzije 8.0 MySQL podržava upotrebu JOIN klauzule u UPDATE SET naredbama.

Primer: Modifikovan SQL upit iz prethodnog primera koristi JOIN klauzulu za spajanje sa tabelama predmet i student što omogućava postavljanje uslova u WHERE klauzuli bez primene podupita.

```
UPDATE student_predmet
JOIN predmet
USING(ID_PREDMETA)
JOIN student
USING (ID_STUDENTA)
SET ID_PREDMETA=(SELECT ID_PREDMETA FROM predmet
                 WHERE NAZIV="Baze podataka")
WHERE IME="Nenad"AND PREZIME="Branković"
AND NAZIV="Programski jezici"
AND SKOLSKA_GODINA="2018/19";
```

8.4 BRISANJE PODATAKA

SQL naredba DELETE se koristi za brisanje podataka u tabeli i može se izvesti pojedinačno ili grupno. DELETE naredba briše celu n-torku (red zapisa), a ne samo vrednost u jednoj koloni.

Osnovni oblik sintakse naredbe DELETE je dat u nastavku:

```
DELETE
FROM ime_tabele
[WHERE lista uslova];
```

Primer: SQL upit ilustruje brisanje zapisa o profesoru Janku Iliću koji je dao otkaz i ne radi više na fakultetu.

```
DELETE
FROM profesor
WHERE IME="Janko" AND PREZIME="Ilić";
```

U ovom primeru, SQL upit sa DELETE naredbom briše red zapisa o profesoru Janku Iliću iz tabele *profesor*. Podaci o predmetnom profesoru određenog predmeta čuvaju se u tabeli *predmet* u koloni ID_PROFESORA, koja predstavlja strani ključ i ukazuje na primarni ključ tabele *profesor*.

Prilikom korišćenja naredbe DELETE, treba voditi računa o efektima koji mogu da se jave kao posledica postojanja ograničenja stranog ključa. Izvršavanjem ove naredbe moguće je da se javi jedna od naredne tri situacije:

Ukoliko u tabeli *predmet* ne postoje predmeti dodeljeni profesoru Janku Iliću iz tabele *profesor*, biće obrisani podaci o navedenom profesoru.

Ukoliko u tabeli *predmet* postoje predmeti dodeljeni profesoru Janku Iliću i strani ključ (kolona ID_PROFESORA) je definisan korišćenjem opcije ON DELETE NO ACTION, DBMS će prijaviti grešku i neće izvršiti brisanje. Brisanje u ovom slučaju nije moguće jer bi u tabeli *predmet* dobili redove koji referenciraju nepostojećeg profesora, čime bi bio narušen referencijalni integritet.

Ukoliko u tabeli *predmet* postoje predmeti dodeljeni profesoru Janku Iliću i strani ključ (kolona ID_PROFESORA) je definisan korišćenjem opcije ON DELETE CASCADE, iz tabele *profesor* biće obrisani podaci o navedenom profesoru, ali će takođe biti obrisani podaci o njegovim predmetima iz tabele *predmet*. Za razliku od prethodnog slučaja, gde će DBMS sprečiti brisanje da ne bi došlo do narušavanja referencijalnog integriteta, u ovom slučaju DBMS automatski briše i podatke iz tabele u kojoj je strani ključ.

Postavlja se sledeće pitanje: Kako obezbediti da prilikom brisanja podataka o Janku Iliću iz tabele *profesor* ne dođe do narušavanja referencijalnog integriteta, a da pri tome sačuvamo zapise o predmetima koji su dodeljeni pomenutom profesoru?

Rešenje ovog problema obuhvata sledeće korake:

- Ažurirati vrednosti kolone ID_PROFESORA u tabeli "predmet".
- Obrisati zapis o profesoru Janku Iliću.

Pre brisanja zapisa o profesoru, potrebno je izvršiti dodelu njegovih predmeta drugim profesorima. Za izmenu vrednosti kolone ID_PROFESORA u tabeli *predmet* koristi se UPDATE SET naredba. Nakon dodele predmeta drugom profesoru, moguće je izvršiti brisanje zapisa o profesoru Janku Iliću uz postavljena ograničenja referencijalnog integriteta.

```
UPDATE predmet
SET ID_PROFESORA = (SELECT ID_PROFESORA FROM profesor
                    WHERE IME="Milica" AND PREZIME ="Milić")
WHERE ID_PROFESORA =(SELECT ID_PROFESORA FROM profesor
                     WHERE IME="Janko" AND PREZIME="Ilić");
```

Primer: Za brisanje podataka iz tabele *student_predmet* o izabranim predmetima studenta Andreja Lazića u školskoj 2018/19. godini korišćen je podupit za izdvajanje identifikacionog broja navedenog studenta koji je specificiran kao jedan od postavljenih uslova u WHERE klauzuli.

```
DELETE
FROM student_predmet
WHERE ID_STUDENTA=(SELECT ID_STUDENTA
                   FROM student
                   WHERE IME="Andrej" AND PREZIME="Lazić")
AND SKOLSKA GODINA="2018/19";
```

Preporuka: Prilikom upotrebe naredbi za ažuriranje neophodno je biti veoma oprezan jer one menjaju stanje baze podataka. Nakon brisanja nije moguće povratiti podatake sem ukoliko se ne radi svakodnevni *backup* baze podataka. Pristup za postizanje efekta pod nazivom *"soft delete"*, gde podaci ostaju u bazi podataka, ali se označavaju kao nedostupni je jedna od preporuka.

Ovaj pristup podrazumeva dodavanje kolone OBRISAN sa podrazumevanom vrednošću "NE". Kada je potrebno obrisati određene podatke, umesto DELETE upita, koristi se UPDATE upit za ažuriranje vrednosti u koloni OBRISAN na "DA".

Primer: Za potrebe ilustracije efekta *"soft delete"* modikovana je tabela profesor. Dodata je kolona OBRISAN sa postavljenom podrazumevanom vrednošću "NE".

```
ALTER TABLE profesor
ADD COLUMN OBRISAN VARCHAR(2) DEFAULT 'NE';
```

Za "brisanje" zapisa o profesoru Janku Iliću je korišćena UPDATE SET naredba kojom je vrednost kolone OBRISAN, za ovaj red zapisa, postavljena na "DA".

```
UPDATE profesor
SET Obrisan = 'DA'
WHERE WHERE IME="Janko" AND PREZIME="Ilić";
```

8.5 PITANJA I ZADACI

1. Napisati SQL upit za dodavanje podataka o pacijentu Ivanović Miroslavu, rođenom 10.01.1995., sa adresom Ustanička 15 i brojem telefon 061123456.
2. Napisati SQL upit za dodavanje podataka za uslugu specijalističkog nutricionističkog pregleda čija je cena 3450.
3. Napisati SQL upit za ažuriranje podataka o e-maila lekara Milice Jovanović.
4. Napisati SQL upit za promenu cene usluge vađenja krvi. Postaviti da je cena uvećana za 10%.
5. Za preglede iz oblasti pedijatrije postaviti lekara Jelenu Lukić.
6. Obrisati podatke o uslugama za koje nije obavljen ni jedan pregled.
7. U ordinaciji se više ne obavlja usluga specijalističkog ORL pregleda. Napisati SQL upit koji će da obriše navedene podatke.
8. Podatke o pregledima iz 2000. godine iskopirati u tabelu Arhiva.
9. Lekar Ana Milojčević je dala otkaz. Prebaciti njene pacijente kod lekara Milice Jovanović.
10. Podatke o lekarima koji imaju više od 10 pregleda u mesecu maju iskopirati u tabelu Stimulacija.

9 NAPREDNI SQL

Cilj ovog poglavlja je da se detaljno objasne ključni elementi relacione baze podataka koji su od suštinskog značaja za uspešnu implementaciju aplikacija u različitim programskim okruženjima. U tom kontekstu, definisani su koncepti pogleda, uskladištenih rutina i okidača. Objašnjene su razlike između pogleda i tabele, naglašavajući kako funkcionalnost pogleda pruža podršku održavanju integriteta podataka u bazi podataka. Istaknute su razlike između koncepta funkcije i procedure, pružajući dublje razumevanje njihove svrhe i primene. Obrađene su kategorije i podkategorije okidača i objašnjena njihova uloga u procesu modifikacije podataka. Naglašene su prednosti i značaj primene opisanih objekata u stvarnim aplikativnim okruženjima, čime se osigurava efikasnost i integritet sistema.

9.1 POGLEDI (VIEW)

U relacionim bazama podataka, pogled predstavlja virtualnu tabelu koja se sastoji od rezultata upita nad jednom ili više tabela u bazi podataka ili drugih pogleda. Osnovna ideja je da se korisnicima omogući pristup podacima na jednostavan i kontrolisan način, prikrivajući kompleksnost strukture podataka i smanjujući potrebu za pisanjem složenih upita. Na primer, jedan korisnik može imati prava prikaza i štampanja samo osnovnih podataka o studentima (ime, prezime, broj indeksa), dok drugi korisnik može u isto vreme imati pravo prikaza ličnih osetljivih podataka o studentima (adresa, matični broj,..).

Osnovna razlika između tabele i pogleda je u tome što tabela fizički postoji na memorijskom medijumu računara (disku), dok je pogled fiktivna virtuelna tabela koja ne zauzima memoriju, ali se može formirati uvek kada je potrebno, odnosno kada se referencira. Sa korisničke tačke gledišta, nema razlike u pretraživanju virtuelne i osnovne tabele u relacionoj bazi podataka.

Definicija pogleda, koja obuhvata naziv pogleda, nazive kolona i upit, se čuva u bazi podataka u prevedenom obliku, što omogućava brzu obradu upita uključenog u pogled. Kada korisnik zahteva pristup pogledu na bazu podataka, DBMS ga poziva i kreira virtuelnu tabelu. Prilikom izvršavanja upita nad pogledom, DBMS kombinuje taj upit sa definicijom pogleda, stvarajući novi upit koji se izvršava nad osnovnim tabelama koje čuvaju podatke na disku. Imena kolona u pogledu ne moraju biti identična imenima kolona u tabelama iz kojih se pogled izvodi.

Pri kreiranju pogleda obavlja se optimizacija njegovog izvršavanja. Ponekad je preporučljivo povremeno ukloniti poglede iz šeme baze podataka i odmah ih ponovo kreirati. Optimalno izvršavanje pogleda zavisi od trenutnih vrednosti u tabelama. Ako

se podaci u tabelama često ažuriraju, promene mogu uticati na performanse pogleda. Stoga je poželjno povremeno ukloniti pogled iz šeme baze podataka i ponovo ga kreirati. Prilikom ponovnog kreiranja, novodobijeni binarni kod će biti optimalan za novo stanje podataka u bazi.

Rad sa pogledima pruža podršku očuvanju integriteta podataka u bazi podataka. Omogućava uprošćavanje upita, kontrolu pristupa podacima i prikaz samo određenih podataka određenim korisnicima. Koristeći poglede, postižu se bolje performanse jer se oni čuvaju u prevedenom obliku, što ubrzava rad. Takođe, pogledi pružaju nezavisnost podataka jer se može menjati definicija pogleda, a ne aplikativni programi koji koriste podatke iz baze preko tih pogleda. Prilikom kreiranja pogleda, kreator pogleda mora imati odgovarajuća prava nad tabelama nad kojima se pogled formira. Korisniku nije neophodno da ima SELECT prava nad izvornim tabelama, ukoliko ima SELECT pravo nad samim pogledom.

Pogledi u bazi podataka se mogu koristiti za ažuriranje podataka u osnovnim tabelama koje se koriste u njihovom formiranju (navedene u FROM klauzuli upita). Da bi ažuriranje podataka bilo moguće, baza podataka mora biti u stanju da mapira strukturu pogleda na strukturu osnovnih tabela (njihove kolone i redove). Za izvršavanje ažuriranja podataka putem pogleda, koriste se standardne SQL naredbe: INSERT...INTO, UPDATE...SET, DELETE FROM. Međutim, važno je imati na umu da neke operacije ažuriranja mogu biti ograničene. Na primer, nije moguće direktno ažurirati vrednosti kolona koje su rezultat primene SQL funkcija, kao što su aritmetičke, funkcije za rad sa stringovima ili agregatne funkcije. U većini sistema za upravljanje bazama podataka, korisniku nije dozvoljeno da vrši ažuriranje tabele preko pogleda ako ima samo SELECT privilegije nad tim pogledom. Ažuriranje podataka obično zahteva odgovarajuće privilegije za izmene u izvornoj tabeli. Dakle, korisnik bi trebao da ima odgovarajuće privilegije za ažuriranje izvorne tabele ako želi da vrši izmene preko pogleda.

Mogu se izdvojiti ključne karakteristika i prednosti pogleda u relacionim bazama podataka:

- **Apstrakcija podataka**. Pogled omogućava korisnicima da vide samo određeni deo podataka, filtrirane ili grupisane na određeni način, bez potrebe za pristupanjem direktno tabelama u bazi.
- **Jednostavnost korišćenja**. Pogledi mogu sakriti složene upite iza jednostavne virtualne tabele, što olakšava korišćenje podataka bez potrebe za pisanjem ponovljivih upita.
- **Bezbednost**. Pogledi omogućavaju kontrolisan pristup podacima. Administratori baze podataka mogu definisati prava pristupa za svaki pogled, ograničavajući korisnicima pristup samo određenim delovima podataka.

- **Optimizacija performansi**. Pogledi mogu pomoći u optimizaciji performansi upita. Na primer, mogu sadržati predefinisane agregatne funkcije ili složene operacije koje mogu biti optimizovane za česte upite.
- **Reorganizacija podataka**. Ukoliko se struktura podataka u bazi promeni, pogledi mogu ostati nepromenjeni, pružajući tako korisnicima stabilan pristup podacima bez potrebe za modifikacijama u aplikacijama koje koriste te podatke.

Međutim, važno je imati na umu i postojanje određenih ograničenja pogleda, kao što su:

- **Performanse**. Složeni pogledi mogu usporiti performanse upita, posebno ako se izvršavaju nad velikim skupovima podataka.
- **Aktuelnost podataka**. Pogledi prikazuju podatke u realnom vremenu samo ako su definisani kao "pogledi sa pretraživanjem" (materialized views), u suprotnom mogu prikazivati samo trenutno stanje podataka u osnovnim tabelama u trenutku izvršavanja upita.
- **Složenost održavanja**. Ukoliko se struktura osnovnih tabela promeni, to može zahtevati i ažuriranje ili redefinisanje pogleda, što može biti kompleksno i zahtevati dodatno održavanje.

Uprkos ovim ograničenjima, pogledi su važan alat u razvoju baza podataka jer omogućavaju efikasniji pristup u korišćenje i prikazu podataka, a i olakšavaju administraciju i kontrolu pristupa.

Osnovni oblik naredbe za kreiranje pogleda je dat u nastavku:

```
CREATE VIEW <ime_pogleda> ([<lista_kolona>]) AS
<SELECT_naredba>;
```

Ova naredba kreira virtuelnu tabelu sa specificiranom listom atributa. Ako lista kolona nije navedena, nazivi i tipovi podataka se preuzimaju iz tabela navedenih u SELECT naredbi.

Osnovni oblik naredbe za brisanje pogleda je dat u nastavku:

```
DROP VIEW <ime_pogleda>;
```

Osnovni oblik naredbe za izmenu pogleda je dat u nastavku:

```
ALTER VIEW <ime_pogleda> ([<lista_kolona>]) AS
<SELECT_naredba>;
```

Pretpostavlja se da je specificirani pogled već kreiran naredbom CREATE VIEW. Sukcesivno izvršavanje naredbi DROP VIEW i CREATE VIEW ima isto dejstvo kao naredba ALTER VIEW, ali zahteva ponovno uspostavljanje prava pristupa.

Primer: SQL upit ilustruje kreiranje pogleda VW_ PREDMET_STUDENT koji sadrži nazive predmeta, školsku godinu, imena, prezimena, broj indeksa studenata koji su prijavljeni za određeni predmet.

```
CREATE VIEW VW_PREDMET_STUDENT AS
   SELECT NAZIV as 'Predmet',
   SKOLSKA_GODINA as 'Školska godina',
   CONCAT(PREZIME,' ',IME) as 'Prezime i ime studenta',
   CONCAT(SMER,'-',BROJ,'/',SUBSTRING(GODINA_UPISA,2,2)) as
   'Broj indeksa'
   FROM student S JOIN student_predmet SP
   USING(ID_STUDENTA)
   JOIN predmet p
   USING(ID_PREDMETA)
   ORDER BY 1,2;
```

Nakon kreiranja, pogled se može koristiti u SQL upitima kao i bilo koja druga tabela. Naredna SQL naredba prikazuje podatke iz kreiranog pogleda VW_PREDMET_STUDENT.

```
SELECT * FROM VW_PREDMET_STUDENT;
```

Primer: SQL upit ilustruje kreiranje pogleda VW_ BROJ_STUDENT_PO_PREDMETU koji za svaki predmet sadrži broj prijavljenih studenata u školskoj godini.

```
CREATE VIEW VW_BROJ_STUDENT_PO_PREDMETU AS
   SELECT NAZIV 'Predmet',
   SKOLSKA_GODINA 'Školska godina',
   COUNT(*) 'Broj prijavljenih studenata'
   FROM student_predmet sp JOIN predmet p
   ON sp.ID_PREDMETA = p.ID_PREDMETA
   GROUP BY p.NAZIV, sp.SKOLSKA_GODINA;
```

SQL upit u ovom primeru koristi alijase za kolone navedene u SELECT listi uključujući i agregatnu funkciju COUNT(*). Kreiranjem pogleda sa ovako navedenom SELECT naredbom, kolone pogleda će imati nazive koji odgovaraju navedenim alijasima.

Alterantivni način imenovanja kolona pogleda podrazumeva navođenje naziva u CREATE VIEW listi. U tom slučaju, za prethodni primer, alternativna CREATE VIEW naredba je data u nastavku.

```
CREATE VIEW VW_BROJ_STUDENT_PO_PREDMETU
(Predmet, Skolska_godina, Broj_prijavljenih_studenata) AS
  SELECT NAZIV, SKOLSKA_GODINA, COUNT(*)
  FROM student_predmet sp JOIN predmet p
  ON sp.ID_PREDMETA = p.ID_PREDMETA
  GROUP BY  p.NAZIV, sp.SKOLSKA_GODINA;
```

9.2 USKLADIŠTENE RUTINE

Uskladištene rutine predstavljaju procedure i funkcije koje sadrže skupove SQL naredbi smeštenih na serveru baze podataka. Predstavljaju ključni koncept u upravljanju bazama podataka i razvoju aplikacija. Omogućavaju korisnicima da izvršavaju određene operacije bez potrebe za ponovnim pisanjem istih naredbi više puta. Umesto toga, jednostavno se pozivaju ove rutine.

Ključne prednosti korišćenja uskladištenih rutina su:

- **Korišćenje programskog koda.** Uskladištene rutine omogućavaju programerima da napišu kompleksne skupove SQL naredbi samo jednom, a zatim ih koriste više puta pozivajući odgovarajuću rutinu. Ovo smanjuje dupliranje koda i olakšava održavanje, jer ako dođe do promena u logici ili zahtevima, potrebno je izmeniti samo jednu rutinu umesto svakog pojedinačnog poziva.
- **Integrisanje.** U okruženjima gde postoji više klijentskih aplikacija napisanih u različitim programskim jezicima, uskladištene rutine omogućavaju da se zajedničke operacije izvršavaju bez obzira na korišćeno programsko okruženje. Ovo olakšava integraciju sistema i smanjuje složenost komunikacije između aplikacija i baze podataka.
- **Bezbednost**. U bankarskim sistemima i drugim okruženjima gde je bezbednost od ključnog značaja, uskladištene procedure i funkcije se koriste za sve operacije nad bazom podataka. Na ovaj način obezbeđena je centralizovana kontrola nad pristupom podacima i operacijama nad njima, čime se smanjuje rizik od neovlašćenog pristupa i zloupotrebe.
- **Performanse**. Uskladištene rutine mogu poboljšati performanse sistema tako što minimizuju količinu podataka koja se prenosi između baze podataka i klijenta. Umesto slanja kompleksnih upita, korisnici jednostavno pozivaju rutinu koja je već optimizovana i izvršava se na serverskoj strani.
- **Monitoring**. Korišćenje uskladištenih rutina omogućava precizno praćenje svake operacije nad bazom podataka. Ovo je posebno korisno u okruženjima gde je neophodno pratiti sve promene podataka iz bezbednosnih ili regulatornih razloga.

- Uskladištene rutine predstavljaju moćan alat koji omogućava efikasnije upravljanje bazama podataka, olakšava razvoj aplikacija i poboljšava bezbednost sistema. Njihova primena se često preporučuje u složenim okruženjima gde je potrebno obezbediti stabilnost, sigurnost i efikasnost.

9.2.1 KORISNIČKI DEFINISANE FUNKCIJE

U kontekstu baze podataka, funkcija predstavlja grupu naredbi odnosno potprogram koji obavlja određenu funkcionalnost i vraća rezultat. Ta funkcionalnost može biti izvršavanje određenih operacija ili izračunavanje vrednosti na osnovu ulaznih parametara ili sadržaja drugih resursa, kao što su tabele u bazi podataka. Može biti pozvana kako u drugim funkcijama i procedurama, tako i u samostalnim upitima nad bazom podataka.

Funkcija može imati različite karakteristike:

- **Izlazni rezultat bez ulaznih parametara.** Funkcija može generisati izlazni rezultat bez potrebe za ulaznim parametrima. Na primer, funkcija može izračunati trenutni datum i vreme i vratiti ih kao rezultat.
- **Ulazni parametri i pristup tabelama.** Ukoliko funkcija ima definisane ulazne parametre, može pristupiti određenim tabelama u bazi podataka i generisati rezultat na osnovu kombinacije ulaznih parametara i sadržaja tih tabela. Na primer, funkcija može primiti datum kao ulazni parametar i vratiti sve transakcije koje su se dogodile tog datuma.

U oba slučaja, funkcija omogućava modularnost i ponovno korišćenje koda, olakšavajući razvoj i održavanje aplikacija i informacionih sistema.

Definicija funkcije podrazumeva navođenje jedinstvenog imena, liste ulaznih parametara i tipa podataka vrednosti koje vraća kao izlaz. Ukoliko se navode ulazni parametri, svaki treba da je precizno definisan imenom i tipom podatka. Imena funkcija nisu strogo ograničena konvencijom imenovanja, mada se preporučuje pridržavanje opštih pravila za imenovanje elemenata baze podataka.

U MySQL-u postoje dve osnovne vrste funkcija:

- **Ugrađene funkcije**. Ove funkcije su već ugrađene u MySQL i omogućavaju različite operacije, kao što su manipulacija datuma, rad sa numeričkim vrednostima, karakterima, agregacijom podataka itd.
- **Korisnički definisane funkcije.** Ove funkcije su kreirane od strane *database developera* i mogu se prilagoditi specifičnim potrebama aplikacije ili baze podataka. U zavisnosti od izlaza, korisnički definisane funkcije mogu biti skalarne ili tabelarne.
    - **Skalarne funkcije**. Izlaz ovih funkcija je jedna vrednost, na primer, rezultat nekog izračunavanja ili manipulacije.

- **Tabelarne funkcije**. Izlaz ovih funkcija je tabela (*recordset*), što znači da funkcija vraća podatke u formi tabelarnog zapisa.

Važno je napomenuti da MySQL podržava samo skalarne funkcije, što znači da izlaz korisnički definisane funkcije u MySQL-u može biti samo jedna vrednost, a ne tabelarni skup podataka u formi *recordseta*.

Osnovna sintaksa za kreiranje funkcije u MySQL bazi podataka je prikazana u nastavku:

```
CREATE [DEFINER = {db_korisnik | CURRENT_USER }]
FUNCTION ime_funkcije ([ime_ulaznog parametra
tip_parametra[,...]])
RETURNS tip_izlazne vrednosti koja se vraća
telo_funkcije
characteristic:
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| {CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}
| SQL SECURITY {DEFINER|INVOKER}
```

Objašnjenje delova sintakse:

- DEFINER: Opcioni deo koji definiše ko je kreator funkcije. Može biti specificiran kao određeni korisnik (db_korisnik) ili trenutni korisnik (CURRENT_USER).
- ime_ulaznog parametra tip_parametra: Navođenje ulaznih parametara funkcije, zajedno sa njihovim tipovima podataka.
- RETURNS tip_izlazne vrednosti: Navođenje tipa podataka izlazne vrednosti koju funkcija vraća.
- characteristic: Opcioni deo koji sadrži različite karakteristike funkcije:
- COMMENT 'string': Opcioni komentar koji opisuje funkciju.
- LANGUAGE SQL: Specifikacija da je jezik funkcije SQL.
- [NOT] DETERMINISTIC: Opcioni deo koji ukazuje da li je funkcija deterministička ili nedeterministička. Deterministička funkcija je ona koja za isti skup ulaznih parametara uvek daje isti rezultat. Podrazumevana vrednost je NOT DETERMINISTIC, što znači da ako se ova karakteristika ne navede, funkcija će biti tretirana kao nedeterministička.

- {CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}: Definisanje tipa SQL operacija koje funkcija može obavljati.
- SQL SECURITY {DEFINER|INVOKER}: Definisanje sigurnosnih privilegija funkcije.

Primer: U primeru je prikazan postupak kreiranja funkcije bez ulaznih parametara. Naredba RETURNS se koristi za definisanje tipa podataka izlazne vrednosti koju vraća funkcija, a naredba RETURN za prikaz izlazne vrednosti funkcije.

```
DELIMITER $$
CREATE FUNCTION FUN_zdravo()
RETURNS varchar(20)
DETERMINISTIC
NO SQL
BEGIN
    RETURN 'Zdravo svete';
END $$
DELIMITER ;
```

U ovom primeru je prikazan SQL kôd za kreiranje jednostavne funkcije u MySQL DBMS sietmu bez ulaznih parametara.Naredba *DELIMITER $$* postavlja delimiter na *$$*. To omogućava da se definicija funkcije napiše na više linija, jer *MySQL* obično koristi tačku-zarez (;) kao delimiter za završetak naredbi. Završetak tela funkcije se označava navođenjem *END $$*. Nakon toga, sa *DELIMITER ;* obezbeđuje se vraćanje delimitera na tačku-zarez (;) što je standardni delimiter za završetak naredbi u MySQL-u.

Ako se koriste novije verzije MySQL-a (od verzije 8.x), DBMS prijavljuju grešku *Error Code: 1418. This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled* ukoliko u definiciji nisu navedene karakteristike funkcije koje su neophodne kada se koristi binarno logovanje. Binarno logovanje (*binlog*) je tehnika koja se koristi u *MySQL* bazi podataka za zapisivanje svih promena koje se dešavaju na nivou baze podataka. Ovo uključuje upite za izmenu podataka (INSERT, UPDATE, DELETE), kao i upite za izmene strukture baze podataka (ALTER, CREATE, DROP).

Dodavanje karakteristika funkcije uključuje odgovarajuće karakteristike kao što su DETERMINISTIC, NO SQL ili READS SQL DATA. Za opisani primer funkcija ne pristupa bazi podataka, žzo uključuje navođenje karaketristike NO SQL. Funkcija

nema ulaznih parametara i uvek vraća istu vrednost pa se može postaviti i DETERMINISTIC karakteristika.

Nakon što se kreira, funkcija se može pozvati korišćenjem SELECT klauzule u SQL upitu, što omogućava da njen rezultat bude direktno umetnut u sam upit. Ovo znači da se rezultat funkcije može koristiti kao deo SELECT naredbe, WHERE klauzule, ili kao argument u drugoj funkciji ili izrazu u okviru istog upita. Na primer, rezultat funkcije može biti direktno prikazan u rezultatu upita ili može biti korišćen za filtriranje podataka prema određenim kriterijumima.

U nastavku je naveden poziv funkcije u okviru SELECT klauzule:

```
SELECT FUN_zdravo();
```

Klauzula *ALTER FUNCTION* u *MySQL*-u može da menja samo karakteristike navedene u sintaksi *ALTER FUNCTION* upita. Ova klauzula omogućava navođenje više promena unutar jednog *ALTER FUNCTION* upita.

Međutim, važno je napomenuti da klauzula *ALTER FUNCTION* ne može da menja parametre ili telo same funkcije. U slučaju potrebe za promenom parametara ili tela funkcije, neophodno je prvo obrisati postojeću funkciju korišćenjem klauzule *DROP FUNCTION*, a zatim ponovo kreirati funkciju korišćenjem klauzule *CREATE FUNCTION* sa novim parametrima ili telom funkcije.

Primer: U nastavku je dat SQL kod za izmenu funkcije *FUN_zdravo()*. Za izmenu funkcije nije korišćena naredba *ALTER FUNCTION* zbog izmena u telu funkcije. Ovaj tip izmene je podrazumevo prvo da se postojeća funkcija obriše korišćenjem naredbe *DROP FUNCTION IF EXISTS*, a zatim ponovo kreira sa izmenjenim telom funkcije.

```
USE studentski_servis;
DROP FUNCTION IF EXISTS FUN_zdravo;
DELIMITER $$
CREATE FUNCTION FUN_zdravo()
RETURNS VARCHAR(20)
NO SQL
DETERMINISTIC
BEGIN
    RETURN 'Zdravo';
END $$
DELIMITER ;
```

Primer: SQL kod ilustruje postupak kreiranja funkcije *FUN_PREDMET_PROSEK* koja na osnovu prosleđenog identifikacionog broja predmeta i identifikacionog broja ispitnog roka u kom je održan ispit vraća prosečnu ocenu na tom predmetu.

```
DELIMITER $$
CREATE FUNCTION FUN_PREDMET_PROSEK (ID_PREDMETA_ul int,
ID_ROKA_ul int)
RETURNS float
BEGIN
DECLARE prosek FLOAT;
SET prosek=(SELECT AVG(ocena)
            FROM zapisnik JOIN ispit
            USING(ID_ISPITA)
            JOIN predmet
            USING(ID_PREDMETA)
            JOIN ispitni_rok
            USING(ID_ROKA)
            WHERE ID_PREDMETA=ID_PREDMETA_ul
            AND ID_ROKA=ID_ROKA_ul AND OCENA >5
            GROUP BY ID_ISPITA);
RETURN prosek;
END $$
```

Poziv funkcije sa ulaznim parametrima podarazumeva navođenje odgovarajućih vrednosti redosledom specificiranim u definiciji funkcije koja je kreirana pomoću CREATE FUNCTION naredbe.

Upotreba kreirane funkcije u SELECT klauzuli je prikazana u nastavku:

```
SELECT FUN_PREDMET_PROSEK(3521,2);
```

U ovom primeru prikazan je SQL kod za kreiranje funkcije sa ulaznim parametrom. Deklarisana je lokalna promenljiva prosek tipa FLOAT, u kojoj će biti smeštena prosečna ocena. Za izračunavanje prosečne ocene na ispitu iz predmeta, za koje su prosleđeni ulazni parametri identifikacioni broj predmeta i identifikacioni broj ispitnog roka u kojem se održao ispit, korišćen je podupit koji izračunava prosečnu ocenu primenom AVG agregatne funkcije. U podupitu je izvršeno spajanje po

jednakosti ključeva tabela zapisnik, ispit, ispitni_rok, i predmet primenom JOIN i USING klauzula. U WHERE klauzuli naveden je uslov OCENA > 5 kojim se izdvajaju redovi zapisa kada je predmet položen, i uslovi koji izdvajaju redove sa vrednostima u kolonama ID_PREDMETA i ID_ROKA koje odgovaraju prosleđenim ulaznim parametrima (ID_PREDMETA = ID_PREDMETA_ul AND ID_ROKA = ID_ROKA). Primena klauzule GROUP BY ID_ISPITA obezbeđuje izračunavanje prosečne ocene predmeta na osnovu realizovanih ispita iz navedenog predmeta.

Primer: SQL kod ilustruje primenu funkcije FUN_PREDMET_PROSEK za prikaz podataka o studentima koje su položili ispit sa ocenom većom od prosečne ocene ostvarene za određeni predmet.

```
SELECT PREZIME,IME,OCENA
FROM student s JOIN zapisnik z
USING (ID_STUDENTA)
JOIN ispit i
USING (ID_ISPITA)
JOIN ispitni_rok ir
USING (ID_ROKA)
JOIN predmet p
USING (ID_PREDMETA)
WHERE p.NAZIV="Aplikativni softver"
AND ir.NAZIV="Februar" AND SKOLSKA_GOD="2018/19"
AND OCENA > FUN_PREDMET_PROSEK(3521,2);
```

9.2.2 PROCEDURE

Procedure su moćan alat koji omogućava efikasnije upravljanje i izvršavanje operacija nad bazom podataka, čime se poboljšava performansa i održavanje sistema. Predstavljaju skupove SQL naredbi koji se čuvaju na serveru baze podataka i mogu se izvršavati po potrebi.

Sadrže logiku, kontrolne strukture, SQL naredbe i druge elemente koji omogućavaju izvršavanje kompleksnih operacija nad bazom podataka. Mogu biti pozvane unutar drugih procedura, ali i kao samostalna naredba. Svaka procedura ima svoje ime i listu parametara.

Osnovna razliku u odnosu na korisnički definisane funkcije je u tome što procedure ne vraćaju vrednosti. Parametri procedure mogu biti ulazni, izlazni ili kombinacija oba tipa odnosno ulazno-izlazni. Svaki parametar je definisan imenom i tipom

podataka. Imena procedura nisu ograničena konvencijama imenovanja, ali treba da budu u skladu sa opštim pravilima za imenovanje elemenata baze podataka.

Ključne prednosti procedura su:

- **Brzina izvršavanja.** Procedure su brže jer se izvršavaju direktno na serveru baze podataka, čime se smanjuje mrežni saobraćaj. Umesto da se šalju pojedinačni SQL upiti kroz mrežu, aplikacija jednostavno poziva proceduru i dobija rezultate.
- **Pogodne za ponavljajuće zadatke.** Idealne su za zadatke koji se ponavljaju i zahtevaju iteraciju ili kompleksnu logiku. Mogu biti automatizovane i izvršavane bez interakcije sa korisnikom, što olakšava upravljanje i održavanje sistema.
- **Nezavisnost od programskog jezika.** Budući da je logika uskladištena direktno u bazi podataka, promena programskog jezika za pristup bazi podataka ne bi trebalo da predstavlja problem. Aplikacija može pozivati iste procedure bez obzira na programski jezik koji se koristi.
- **Standardizacija sintakse**. Sintaksa procedura u MySQL-u prati SQL:2003 standard, što olakšava njihovu primenu u drugim relacionim bazama podataka. To znači da se većina veština i znanja koje se steknu u radu sa procedurama u MySQL-u lako može preneti na druge DBMS – ove.

Osnovna sintaksa za kreiranje procedure u MySQL bazi podataka je prikazana u nastavku:

```
CREATE [DEFINER = {db_user | CURRENT_USER }]
PROCEDURE naziv_procedure ([proc_parameter[,...]])
telo_procedure
characteristic:
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| {CONTAINS SQL|NO SQL|READS SQL DATA|MODIFIES SQL DATA}
| SQL SECURITY {DEFINER|INVOKER}
```

Procedura može biti definisana sa nekoliko strukturnih parametara (CONTAINS SQL, NO SQL, READS SQL DATA, MODIFIES SQL DATA) koji opisuju prirodu same procedure.

Ovi parametri nemaju direktnog uticaja na interakciju sa procedurom, već služe MySQL serveru kao smernice prilikom optimizacije i kreiranja statistika. Na taj način, MySQL može bolje razumeti ponašanje procedure i efikasnije upravljati njenim izvršavanjem. Još jedan parametar koji ima sličnu svrhu je DETERMINISTIC

(NON DETERMINISTIC). Ovaj parametar takođe ima značajan uticaj na rad optimizacionog Engine-a MySQL servera, ali ne nameće preterane sintaksne obaveze prilikom pisanja procedure. Kada procedura bude označena kao deterministic, to znači da se njen izlaz nikada ne menja.

Na primer, u proceduri koja prikazuje poruku 'zdravo', rezultat će uvek biti isti. Međutim, ako procedura nije deterministic (što je podrazumevano), MySQL ne očekuje da rezultat procedure uvek bude isti. Na primer, ako procedura sadrži funkciju now(), koja prikazuje tačno vreme, rezulata će svaki put biti drugačiji.

Naredba za brisanje procedure je prikazan u nastavku:

```
DROP PROCEDURE [IF EXISTS] ime_procedure;
```

Procedura može prihvatiti različite vrste parametara, a to su:

- **Ulazni parametri (IN).** Ovi parametri se koriste samo za čitanje unutar procedure i njihove vrednosti se prosleđuju prilikom poziva procedure. Oni omogućavaju unos podataka u proceduru.
- **Ulazno-izlazne parametre (INOUT).** Ovi parametri omogućavaju unos podataka u proceduru kao i povrat vrednosti iz procedure nazad u pozivajući kod. Oni se mogu čitati i menjati unutar procedure.
- **Izlazne parametre (OUT).** Ovi parametri se koriste za vraćanje vrednosti iz procedure nazad u pozivajući kod. Mogu biti prazni pri pozivu procedure, a zatim popunjeni sa vrednostima koje procedura dodeli.

Podrazumevana vrsta parametara procedure su ulazni parametri (IN), što znači da, ako nije drugačije navedeno, parametri se smatraju ulaznim parametrima. Ova vrsta parametara omogućava da se vrednosti prosleđene proceduri koriste unutar nje, ali se ne menja njihova vrednost van procedure.

Primer: SQL kod ilustruje postupak kreiranja procedure za prikaz zapisnika za ispit

```
DELIMITER $$
CREATE PROCEDURE SP_ZAPISNIK
(IN predmet VARCHAR(20), IN rok VARCHAR(20), IN godina
VARCHAR(10))
BEGIN
SELECT CONCAT(s.PREZIME, " ", s.IME) AS "Prezime i ime
studenta",CONCAT(s.SMER, "-", s.BROJ, "/",
SUBSTRING(s.GODINA_UPISA, 3, 2)) AS "BROJ INDEKSA",
z.BODOVI, z.OCENA
    FROM zapisnik z JOIN ispit i
    USING (ID_ISPITA)
    JOIN student s
    USING (ID_STUDENTA)
    JOIN ISPITNI_ROK ir
    USING (ID_ROKA)
    JOIN predmet p
    USING(ID_PREDMETA)
    WHERE p.NAZIV = predmet AND ir.NAZIV = rok
    AND ir.SKOLSKA_GOD = godina
    ORDER BY 2,1;
END $$
DELIMITER ;
```

9.2.3 RAZLIKA IZMEĐU PROCEDURA I FUNKCIJA

Procedura i funkcije spadaju u kategoriju uskladištenih rutina u SQL-u. Predstavljaju grupu naredbi koje se čuvaju kao objekti na serveru baze podataka, te imaju za cilj poboljšanje performansi i olakšavanje održavanja DBMS-a.

Iako se na prvi pogled mogu činiti sličnim, postoje značajne razlike između procedura i funkcija koje određuju kada je odgovarajuće koristiti koju od ovih rutina.

U tabeli 9.2.3.1 su prikazani ključni elementi koji razlikuju procedure od funkcija i važni su pri odabiru koja rutina je najprikladnija za određeni scenario.

Tabela 9.2.3.1 Razlike između između procedura i funkcija

Aspekt	Procedure	Funkcije
Povratne vrednosti	Mogu vratiti NULL ili više vrednosti.	Vraćaju samo jednu vrednost.
Parametri	Mogu imati ulazne, izlazne i ulazno-izlazne parametre.	Mogu imati samo ulazne parametre.
Korišćenje naredbi	Mogu koristiti SELECT i DML upite.	Mogu koristiti samo SELECT upite.
Pozivanje	Iz procedure mogu biti pozvane funkcije.	Iz funkcije ne može biti pozvana procedura.
Obrada izuzetaka	Moguće je korišćenje try-catch blokova.	Ne podržavaju korišćenje try-catch blokova.
Upotreba u SQL-u	Ne mogu se koristiti unutar SQL naredbi.	Mogu se koristiti unutar SQL naredbi.
Transakcije	Podržavaju transakcije.	Ne podržavaju transakcije.

9.3 OKIDAČI

Okidači ili trigeri su korisnički definisani blokovi koda koji se aktiviraju u trenutku neke od akcija ažuriranja podataka u tabelama (INSERT, UPDATE i DELETE). Omogućavaju manipulaciju podacima pre nego što budu zaista uneti, izmenjeni ili obrisani.

Klasifikuju se u tri osnovne kategorije: INSERT, UPDATE i DELETE, pri čemu se svaka kategorija dalje deli na dve podkategorije: BEFORE i AFTER. Različite kategorije okidača nude specifične opcije, prilagođavajući se potrebama korisnika.

Svaki okidač predstavlja objekat unutar određene baze podataka i važi isključivo za tu bazu. U slučaju brisanja baze podataka, svi njeni okidači takođe će biti uklonjeni.

Prilikom kreiranja okidača, neophodno je povezati ga sa određenom tabelom i akcijom koju želimo da pratimo. Važno je napomenuti da nije moguće imati više okidača istog imena u jednoj tabeli, niti postaviti iste tipove okidača na isti događaj u toj tabeli (na primer, dva BEFORE INSERT okidača na istoj tabeli). Ovo osigurava jasnoću i doslednost u radu sa okidačima.

Kada se postavi okidač na tabelu, on postaje deo te tabele u kontekstu njenih transakcija. To znači da se prilikom ažuriranja podataka, uzima u obzir i izvršavanje okidača, a ako dođe do greške prilikom aktivacije okidača, cela akcija smatra neuspešnom. Ukoliko dođe do greške prilikom aktivacije BEFORE okidača, to znači da se operacija nije uspešno izvršila i neće doći ni do aktivacije AFTER okidača (ukoliko se oba okidača nalaze na istoj naredbi iste tabele). Ovo je važno jer čini

celokupni proces transakcija pouzdanijim i omogućava sistematično rukovanje greškama.

9.3.1 BEFORE OKIDAČI

Karakteristika ovih okidača je da se aktiviraju pre nego što podatak stigne do same tabele. To omogućava da podatak bude uhvaćen i eventualne promene na njemu izvršene pre nego što je konačno zapisan. U tu svrhu, prilikom unosa, MySQL kreira privremenu tabelu u memoriji koja ima istu strukturu kao ciljna tabela, ali s jednom bitnom razlikom: podaci u ovoj privremenoj tabeli su zapravo podaci koje unosimo ili menjamo. Ovo omogućava okidačima da rade sa ažuriranim podacima pre nego što se oni stvarno zapišu u ciljnu tabelu.

Opšta sintaksa BEFORE okidača prikazana je u nastavku:

```
CREATE TRIGGER ime_okidača
BEFORE INSERT OR UPDATE OR DELETE ON ime_tabele
FOR EACH ROW
BEGIN
    -- Telo okidača: ovde se definiše
       šta okidač treba da uradi
END;
```

Objašnjenje delova sintakse:

- **BEFORE** označava da okidač treba da se aktivira pre nego što se izvrši INSERT, UPDATE ili DELETE operacija.
- **INSERT OR UPDATE OR DELETE** određuje koje operacije na tabeli će aktivirati okidač.
- **ime_tabele** je ime tabele na koju se odnosi okidač.
- **FOR EACH ROW** označava da će okidač biti izvršen za svaki zapis koji je zahvaćen operacijom.
- **BEGIN ... END** definiše telo okidača, u kojem se nalaze SQL naredbe koje se izvršavaju kada se okidač aktivira.

Izmene koda u telu okidača realizauju na isti način kao kod procedura i funkcija. Okidač se obriše, a zatim ponovo kreira.

Naredba za brisanje okidača je prikazana u nastavku:

```
DROP TRIGGER [IF EXISTS] ime_okidača;
```

Primer: SQL kod ilustruje postupak kreiranja BEFORE INSERT okidača na tabeli *predmet.*

```
DELIMITER //
CREATE TRIGGER before_insert_predmet
BEFORE INSERT ON predmet
FOR EACH ROW
BEGIN
    IF NEW.status IS NULL THEN
        SET NEW.status = 'izborni';
    END IF;
    IF NEW.espb IS NULL THEN
        SET NEW.espb = 6;
    END IF;
    IF NEW.id_profesora <=> NULL THEN
        SET NEW.id_profesora = NULL;
    END IF;
    IF NEW.preduslov <=> NULL THEN
        SET NEW.preduslov = NULL;
    END IF;
    IF NEW.naziv IS NULL OR NEW.naziv = '' THEN
        SET NEW.naziv = 'Nepoznato';
    END IF;
END;
//
DELIMITER ;
```

U MySQL-u, NEW je rezervisana ključna reč koja se koristi unutar okidača kako bi se referisalo na novi red koji se dodaje ili menja u tabeli za koju je okidač definisan. U ovom primeru upotreba NEW se odnosi na novi red koji se dodaje u tabelu *predmet*, jer je okidač definisan za ovu tabelu. *NEW.status*, *NEW.espb*, *NEW.id_profesora*, *NEW.preduslov*, i *NEW.naziv* se odnosi na kolone u tabeli *predmet* za koje se proverava ili postavlja vrednosti prilikom unosa novog reda.

9.3.2 AFTER OKIDAČI

AFTER okidači se mogu koristiti nakon INSERT, UPDATE ili DELETE naredbi. Ovi okidači se aktiviraju nakon što se izvrše prethodno pomenute naredbe nad odgovarajućom tabelom, omogućavajući izvršavanje određene akcije nakon što se operacija unosa, ažuriranja ili brisanja podataka završi. To znači da kada se nađemo u okviru koda okidača, imamo pristup već unetim podacima. Korisni su u slučaju ažuriranja dodatne tabele paralelno sa glavnom tabelom. Na primer, možemo koristiti After okidače da automatski ažuriramo arhivsku, log ili backup tabelu nakon što su podaci uspešno uneti ili izmenjeni u glavnoj tabeli. Ovo je korisno za održavanje dodatnih kopija podataka ili praćenje istorije promena.

Opšta sintaksa AFTER okidača prikazana je u nastavku:

```
CREATE TRIGGER ime_okidača
AFTER INSERT OR UPDATE OR DELETE ON ime_tabele
FOR EACH ROW
BEGIN
 Telo okidača: ovde se definišu akcije koje mogu da se izvrše
nakon unosa, izmene ili brisanja reda u tabeli
END;
```

9.4 PITANJA I ZADACI

1. Objasniti koncept i značaj pogleda.
2. Šta predstavljaju uskladištene rutine?
3. Navesti ključne karakteristike uskladištenih rutina.
4. Objasniti pojam korisnički definisanih funkcija.
5. Navesti karakteristike skalarnih funkcija.
6. Navesti karakteristike tabelarnih funkcija.
7. Objasniti pojam procedura.
8. Koje vrste parametara mogu da imaju procedure?
9. Navesti razlike između funkcija i procedura.
10. Objasniti pojam okidača i navesti podelu na kategorije i podkategorije.

10 INDEKSI

Cilj ovog poglavlja je usmeren na objašnjenje strategije indeksiranja u relacionim bazama podataka. Naglašena je ključna ulogu indeksa u efikasnom pretraživanju i pristupu podacima. Razmotrene su i posledice nepravilne ili prekomerne upotrebe indeksa. Ključni tipovi indeksa, poput klasterovanih, neklasterovanih i fulltext indeksa, jasno su definisani, uz objašnjenje strategija indeksiranja i metodologiju za definisanje različitih tipova indeksa. Uz to, razmatrani su i B-TREE i HASH mehanizmi za skladištenje indeksa.

10.1 ZNAČAJ PRIMENE INDEKSA

Ključni aspekt efikasnog upravljanja i rad sa bazama podataka usmeren je na precizno dobavljanje podataka. Kada su količine podataka u tabelama manje, sistem će uvek brzo funkcionisati. Međutim, kada baza počne da sadrži veće količine podataka, brzina postaje jedan od ključnih problema. Ipak, sama brzina dostavljanja podataka nije uzrok sporosti procesa. Glavni izazov leži u postupku pronalaženja i izdvajanja podataka. Nakon što se podatak jednom pronađe, brzina njegovog dostavljanja postaje irelevantna.

Mehanizam indeksa omogućava brzo pretraživanje i pristupanje podacima u tabelama. Indeksi se koriste za ubrzavanje upita tako što olakšavaju pronalaženje odgovarajućih redova u tabeli. Kada se kreira indeks za određenu kolonu ili skup kolona, DBMS generiše strukturu podataka koja omogućava efikasno traženje i pristupanje podacima na osnovu vrednosti indeksiranih kolona.

Većina podataka u MySQL bazi je organizovana po stranicama. Veličina jedne stranice je memorijski ograničena, obično 8 KB ili 16 KB, ali tačan broj redova koji može stati u jednu stranicu ne može se precizno odrediti, jer zavisi od različitih faktora, pre svega, od količine podataka u redovima.

Korišćenjem indeksa, SQL upiti mogu brže pronaći odgovarajuće redove u tabeli, što rezultira efikasnijim izvršavanjem upita i poboljšanom performansom baza podataka. Međutim, važno je voditi računa i o pravilnom korišćenju indeksa, jer neadekvatno korišćenje može dovesti do neefikasnosti ili nepotrebnog opterećenja baze podataka.

10.2 KLASTEROVANI I NEKLASTEROVANI INDEKSI

Pretraga indeksa funkcioniše po istom principu, ali se njena finalizacija može izvršiti na dva načina. Rezultat različite finalizacije pretrage indeksa ukazuje na postojanje

dve vrste standardnih indeksa: klasterovani i neklasterovani. Često se ova dva indeksa porede sa knjigom i telefonskim imenikom.

Ukoliko tražimo neku informaciju iz knjige, prvo ćemo pretražiti sadržaj i u njemu naći stranu za ono što nas zanima. Zatim ćemo prelistati knjigu i brzo doći do podatka, jer znamo stranu. Ako tražimo neki podatak u telefonskom imeniku, naći ćemo slovo koje nas zanima, a zatim direktno pristupiti strani na kojoj se nalaze svi podaci koji počinju tim slovom.

Klasterovani indeks je vrsta indeksa u bazi podataka gde redovi u tabeli fizički odgovaraju redovima u indeksu. Drugim rečima, redovi u tabeli su organizovani i skladišteni na isti način kao i redovi u indeksu. Ovo znači da kada se koristi klasterovani indeks za neku kolonu, redovi u tabeli su automatski sortirani prema vrednostima te kolone. Kao rezultat toga, ključevi indeksa direktno upućuju na odgovarajuće redove u tabeli.

Kada se kreira primarni ključ tabele, automatski se kreira i klasterovani indeks. Redovi u tabeli su organizovani prema vrednostima primarnog ključa, što omogućava brzo pretraživanje i pristupanje podacima. Klasterovani indeksi omogućavaju brže pretraživanje i pristupanje podacima, posebno kada se koriste u upitima koji koriste kolonu po kojoj je indeksiran klasterovani indeks. Međutim, važno je napomenuti da tabela može imati samo jedan klasterovani indeks, što znači da treba pažljivo odabrati kolonu za klasterovanje kako bi se postigla optimalna performansa.

Neklasterovani indeks je vrsta indeksa u bazi podataka gde redovi u indeksu ne odgovaraju direktno redovima u tabeli. Drugim rečima, redovi u indeksu nisu fizički organizovani na isti način kao redovi u tabeli, već sadrže reference ili pokazivače na odgovarajuće redove u tabeli. Ovo omogućava da redovi u tabeli budu organizovani nezavisno od strukture indeksa. Primer neklasterovanog indeksa može biti indeksiranje kolone koja nije primarni ključ ili ključ klastera.

Kada se koristi neklasterovani indeks, redovi u tabeli nisu automatski sortirani prema vrednostima indeksirane kolone, već se indeks koristi za brzo pronalaženje redova u tabeli na osnovu vrednosti indeksirane kolone. Neklasterovani indeksi omogućavaju efikasno pretraživanje i pristupanje podacima, posebno kada se koriste u upitima koji ne koriste kolonu po kojoj je indeksiran klasterovani indeks. Za razliku od klasterovanog, tabela može imati više neklasterovanih indeksa, što omogućava fleksibilnost u optimizaciji performansi baze podataka za različite vrste upita.

Može se zaključiti da klasterovani indeks, na svom najnižem nivou, sadrži same podatke, dok neklasterovani indeks sadrži reference na prave podatke. Zato su neklasterovani sporiji od klasterovanih indeksa.

10.3 FULLTEXT INDEKS

FULLTEXT je vrsta indeksa koja omogućava efikasno pretraživanje velikih količina teksta u bazi podataka. Ova vrsta indeksa je posebno korisna kada je potrebno vršiti pretragu unutar tekstualnih polja, kao što su opisi, članci, blogovi, komentari i slično.

Jedna od ključnih karakteristika FULLTEXT indeksa je sposobnost pretrage reči koje se nalaze u tekstu, a ne samo reči koje počinju određenim karakterom, kao što je to slučaj sa standardnim indeksima. Omogućava i rangiranje rezultata pretrage na osnovu relevantnosti, što je korisno kada se želi prikazati najrelevantniji rezultat na vrhu liste.

Kada se koristi FULLTEXT indeks na tekstualnom polju, podaci se parsiraju i indeksiraju na način koji omogućava brzo pronalaženje reči ili fraza unutar teksta. Proces započinje razbijanjem teksta na pojedinačne reči, uz eliminaciju stop reči, odnosno čestih reči koje se ne indeksiraju, poput "i", "ili", "ali", normalizaciju i druge transformacije koje olakšavaju pretragu. Svaka reč se smešta u indeks pod određenim brojem, dok se polje iz kojeg je tekst preuzet pamti pod određenim brojem. Na kraju se uspostavlja relacija između polja sa tekstom i reči koje tom polju pripadaju.

Prilikom pretrage, MySQL Engine prvo pronalazi tražene reči, zatim izlistava njihove relacije (u kojima se pamti i broj pojavljivanja reči u polju), a na osnovu tih relacija sortira rezultate pretrage po broju pojavljivanja reči. Ovo omogućava prikazivanje rezultata pretrage prema relevantnosti.

FULLTEXT indeksi su posebno korisni za pretragu teksta u aplikacijama kao što su pretraživači, sistemi za upravljanje sadržajem, forumi i druge aplikacije gde je pretraga teksta ključna funkcionalnost.

10.4 STRATEGIJA INDEKSIRANJA

Prilikom kreiranja tabele i definisanja ograničenja primarnog ključa nad nekom kolonom ili kolonama, DBMS automatski za te kolone kreira klasterovani indeks. Zbog toga je neophodno voditi računa na koje kolone se postavlja ograničenje primarnog ključa kako bi klasterovani indeks imao efekta u postupku pretraživanja i pristupanja podacima.

Neklasterovani indeks je malo pristupačniji i može se postaviti na bilo koju kolonu. Međutim, to ne znači da je pametno postaviti ga na sve kolone tabele. Indeks predstavlja dodatnu količinu podataka koja se čuva na serveru baze podataka. Istovremeno, zahteva dodatno procesiranje od strane DBMS-a.

Indeksi su od suštinskog značaja za ubrzavanje pretrage podataka, ali isto tako, mogu se koriste i prilikom drugih operacija nad tabelama. Veći broj indeksiranih kolona dovodi do usporavanja obrade indeksa prilikom maniplulacije sa podacima tabele koja se indeksira.

Opšta sintaksa naredbe za kreiranje indeksa u MySQL-u je prikazana u nastavku:

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX naziv_indeksa
USING {BTREE | HASH}
ON naziv_tabele (naziv_kolone,...);
```

Prilikom kreiranja potrebno je navesti tip indeksa koji se kreira i mehanizam čuvanja u bazi podataka. Navođenje tipa indeksa podrazumeva jednu od sledeće četri kategorije: INDEX, UNIQUE INDEX, FULLTEXT INDEX, SPATIAL INDEX. Što se tiče mehanizma čuvanja indeksa, MySQL podržava B-TREE i HASH.

INDEX i **UNIQUE INDEX** se odnosi na klasterovane i neklasterovane indekse.

- **INDEX** se koristi za kreiranje standardnog indeksa, bilo klasterovanog ili neklasterovanog. Ovo je indeks koji se koristi za optimizaciju performansi, bez garancije jedinstvenosti vrednosti.
- **UNIQUE INDEX** kreira indeks koji garantuje jedinstvenost vrednosti. Može se primeniti i na klasterovane i neklasterovane indekse. Ukoliko se koristi na klasterovanom indeksu, podaci će biti organizovani tako da se vrednosti ključa ne ponavljaju, što je karakteristično za primarni ključ. U slučaju neklasterovanog indeksa, osigurava se da nema dupliciranih vrednosti u indeksiranim kolonama.

B-TREE i **HASH** su dva različita mehanizma korišćena za čuvanje indeksa.

- **B-TREE** (engl.*Balanced Tree*) je podrazumevani mehanizam za indekse u MySQL-u. Koristi algoritam pretrage binarnog stabla. Indeksirani podaci se organizuju u balansirano binarno stablo, što omogućava efikasno pretraživanje. B-TREE indeksiranje je pogodno za različite tipove pretraga, uključujući opsege, tačkaste pretrage i pretrage sa prefiksom.
- **HASH** koristi pretragu tabele heš vrednosti. Ovo se često koristi za pretragu po primarnim ključevima ili za brze pretrage gde je ključ određen heš funkcijom. HASH indeksi obično imaju brži pristup od B-TREE indeksa, ali imaju ograničenja.

Izbor između B-TREE i HASH indeksa zavisi od specifičnih zahteva pretrage i tipova podataka koji se indeksiraju. B-TREE indeksi su opširniji i fleksibilniji, dok su HASH indeksi efikasniji za određene tipove pretrage.

Brisanje indeksa se vrši upotrebom naredbe DROP INDEX čija je sintaksa prikazana u nastavku:

```
DROP INDEX naziv_indeksa ON naziv_tabele;
```

Primer: SQL naredba ilustruje kreiranje jedinstvenog indeksa za kolonu NAZIV u tabeli predmet kojim se obezbeđuje jedinstvena vrednost naziva predmeta.

```
CREATE UNIQUE INDEX uq_predmet_naziv USING BTREE ON
predmet(NAZIV);
```

Za prikaz postojećih indeksa u tabeli *predmet* korišćena je naredba SHOW INDEX.

```
SHOW INDEX FROM predmet;
```

Rezultat izvršavanja prikazan je na slici 10.4.1.

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
predmet	0	PRIMARY	1	ID_PREDMETA	A	23	NULL	NULL		BTREE			YES	NULL
predmet	0	uq_predmet_naziv	1	NAZIV	A	24	NULL	NULL		BTREE			YES	NULL
predmet	1	FK_PREDMET_RELATIONS_PROFESOR	1	ID_PROFESORA	A	21	NULL	NULL	YES	BTREE			YES	NULL

Slika 10.4.1 Prikaz postavljenih indeksa na tabeli predmet

10.5 PITANJA I ZADACI

1. Objasniti značaj upotrebe indeksa u realcionim bazama podataka.
2. Koje su posledice neadekvatne primene indeksa?
3. Navesti tipove indeksa.
4. Navesti mehanizme za čuvanje indeksa.
5. Objasniti koncept klasterovanog indeksa.
6. Objasniti koncept neklasterovanog indeksa.
7. Objasniti koncept FULLTEXT indeksa.
8. Šta predstavlja INDEX a šta UNIQUE INDEX?
9. Objasniti B-TREE mehanizam.
10. Objasniti HASH mehanizam.

11 TRANSAKCIJE

Cilj ovog poglavlja je usmeren na objašnjenje koncepta transakcija kao ključnog mehanizma za upravljanje podacima. Definisani su principi na kojima je zasnovan ACID model i istaknut značaj u očuvanju integriteta podataka u relacionim bazama podataka. Razmatrani su problemi konkurencije transakcija kao što su fantomsko čitanje, izgubljeno ažuriranje ili čitanje prljavih podataka. Detaljno su objašnjeni načini zaključavanja transakcija u cilju rešavanja problema konkurentnosti. Razmatran je pojam deadlock-a i strategije sprečavanja istog. Objašnjen je koncept različitih tipova izolacionih nivoa.

11.1 ACID MODEL

Transakcije u relacionim bazama podataka predstavljaju ključni mehanizam za upravljanje podacima, omogućavajući grupisanje jedne ili više SQL naredbi u jednu celinu. To je ključna komponenta integriteta podataka u bazi.

Glavne karakteristike transakcija zasnovane su na ACID modelu, koji podrazumeva sledeće principe:

- **Atomičnost** (engl.*Atomicity*). Transakcija je nedeljiva, ili se izvrši ili ne izvrši. Delimična izvršavanja nisu moguća.Primenjuje se cela transakcija (operacija COMMIT) ili ni jedna od operacija obuhvaćenih transakcijom (operacija ROLLBACK).
- **Konzistentnost** (engl.*Consistency*). Ovaj princip osigurava da će baza podataka biti u konzistentnom stanju nakon izvršavanja transakcije. U slučaju bilo kakvih neispravnosti ili neadekvatnih ažuriranja podataka, sve će biti vraćeno na početno stanje.
- **Izolacija** (engl.*Isolation*). Omogućava izvršavanje više transakcija istovremeno, bez negativnih posledica ili međusobnog ometanja. Transakcija se izvršava tako da ne utiče na rad ostalih transakcija koje se izvršavaju ”paralelelno”.
- **Izdržljivost** (engl.*Durability*). Izvršena ažuriranja su trajna. Transakcije treba da budu sačuvane čak i u slučaju otkaza sistema ili prekida rada baze neposredno nakon njihovog završetka.

Podrazumevano, DBMS sistem relacionih baza podataka nalazi se u autocommit modu. To znači da se svaka SQL naredba izvršava u okviru jedne male transakcije, osiguravajući da naredbe budu izvršene u potpunosti, a nikada samo delimično. Na primer, brisanje zapisa iz baze podataka primenom DELETE naredbe dešava se

unutar jedne transakcije (za MySQL DBMS-u neophodno je da tabela iz koje se briše zapis bude tipa InnoDB).

11.2 COMMIT I ROLLBACK

Kontrola izvršavanja SQL naredbi, bilo da je u pitanju jedna ili grupa naredbi, postiže se primenom mehanizma transakcija, što podrazumeva prethodno isključivanje autocommit moda.

Isključivanje autocommit režima rada ostvaruje se primenom naredbe SET, kojom se sistemska promenljiva autocommit postavlja na vrednost nula (sistemske promenljive u SQL-u se označavaju sa @@).

```
SET @@autocommit=0;
ili
SET autocommit=0;
```

Kada je autocommit podešen na 0, ni osnovne naredbe neće biti izvršene ukoliko nije ekcplicitno navedena naredba kojom će se potvrditi izvršavanje. Drugim rečima, kada je autocommit mod isključen, sve SQL naredbe se smatraju jednom transakcijom sve dok se ista ne potvrdi ili otkaže

Postupak dodele vrednosti u sistemskoj promenljivoj u okviru SET naredbe je moguć i bez navođenja prefiksa @@. Međutim, za prikaz trenutne vrednosti autocommit promenljive u SELECT naredbi neophodno je navesti prefiks @@.

```
SELECT @@autocommit;
ne može
SELECT autocommit;
```

Za upravljanje transakcijama koriste se naredbe **COMMIT** i **ROLLBACK**.

Naredba **COMMIT** se koristi za potvrđivanje izvršavanja SQL naredbi unutar transakcije kojima se realizuju određene promene u bazi podataka. Kada se COMMIT naredba izvrši, sve prethodne promene (INSERT, UPDATE, DELETE) koje su bile privremeno skladištene tokom transakcije, postaju trajne i vidljive u bazi podataka. Nakon izvršavanja COMMIT naredbe, transakcija je završena i resursi koji su bili blokirani za tu transakciju se oslobađaju.

Naredba **ROLLBACK** se koristi za poništavanje ili opoziv promena izvršenih unutar transakcije. Ako se dogodi bilo kakva greška tokom izvršavanja transakcije ili ako korisnik odluči prekinuti transakcijukoristi se naredba ROLLBACK kako bi se vratio početni status baze podataka pre početka transakcije. Kada se ROLLBACK izvrši,

sve promene unutar transakcije se poništavaju, a baza podataka se vraća u stanje u kome je bile pre početka transakcije.

Pored naredbi za potvrdu odnosno otkazivanje promena u okviru transakcije, da bi se obezbedila eksplicitnost, neophodno je naglasiti početak i kraj transakcije.

Opšta sintaksa transakcije je navedena u nastavku:

```
START TRANSACTION
  [karakteristike_transakcije [,karakteristike_transakcije]
...] karakteristike_transakcije:
   {WITH CONSISTENT SNAPSHOT | READ WRITE | READ ONLY}
BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

11.3 KONTROLA TOKA TRANSAKCIJE

U cilju kontrole toka transakcije, korisno je primenjivati **kontrolne tačke**. One se obeležavaju ključnom rečju **SAVEPOINT** i omogućavaju definisanje određenih tačaka unutar transakcije na kojima se transakcija može vratiti u prethodno definisano stanje. Na ovaj način omogućava se veća kontrola nad tokom transakcije i efikasno upravljanje greškama ili promenama tokom izvršavanja. Kada se postavi kontrolna tačka unutar transakcije, stanje transakcije se "zamrzava" u tom trenutku.

Kontrolne tačke omogućavaju da, nakon izvršenja niza operacija, ukoliko je potrebno vratiti bazu podataka na prethodno stanje, koristi se kontrolna tačka kako bi se transakcija vratila u stanje pre izvršenih promena. Ovo omogućava očuvanje doslednosti podataka i minimiziranje gubitka informacija u slučaju problema.

Primena kontrolnih tačaka je korisna i u situacijama kada je potrebno podeliti transakciju na logičke segmente ili kada želimo obezbediti mogućnost vraćanja na prethodno stabilno stanje u slučaju greške.

Na primer, moguće je postaviti kontrolnu tačku na početku transakcije, zatim izvršiti nekoliko operacija, postaviti drugu kontrolnu tačku nakon nekog koraka, a zatim nastaviti sa izvršavanjem. Ako dođe do greške, proces se vraća na poslednju kontrolnu tačku i na taj način se izbegava nepotrebno poništavanje cele transakcije. Ovo može biti posebno korisno u složenim operacijama sa bazom podataka gde je važno održati dosledno stanje podataka i minimizovati gubitak podataka u slučaju problema.

Primer: SQL kod ilustruje primenu transakcije sa postavljenom kontrilnom tačkom T1.

```
START TRANSACTION;
INSERT INTO profesori_stari VALUES
            (121,"Mile","Milojković","dr","2001-12-12");
SAVEPOINT T1;
DELETE FROM profesori_stari WHERE YEAR(DATUM_ZAP) < 1980;
ROLLBACK TO T1;
COMMIT;
SELECT * FROM profesori_stari;
SET autocommit=0;
```

U ovom primeru prikazan je postupak dodavanja novog zapisa kao i brisanje zapisa koji zadovoljavaju postavljen uslov u tabeli *profesor_stari*. U cilju kontrole toka transkacije definisana je kontrolna tačka *T1* koja je postavljena posle INSERT INTO naredbe. Brisanje zapisa na osnovu postavljenog uslova se izvršava nakon kontrolne tačke T1.

Naredba ROLLBACK TO T1 poništava promene u tabeli nastale posle definisane kontrolne tačke. To znači da DELETE naredba za brisanje zapisa sa postavljenim uslovom će biti opozvana. U tabeli će biti zapaćeno stanje do trenutka kada je definisana kontrolna tačka što u ovom primeru podrazumeva da će dodavanje novog zapisa biti realizovano i zapaćeno kao novo stanje.

Naredba COMMIT potvrđuje sve promene napravljene tokom transakcije i trajno ih sprema u bazu podataka. S obzirom na prethodno izvršenu naredbu ROLLBACK operaciju, ova naredba neće imati nikakvih promena za potvrdu, ali se izvršava kao deo transakcijskog protokola.

11.4 PROBLEM KONKURENCIJE

Problem konkurencije u transakcijama baza podataka nastaje kada više transakcija istovremeno pristupa ili pokušava da promeni iste podatke u bazi. Ovaj problem može dovesti do različitih problema, uključujući gubitak integriteta podataka, anomalije u čitanju i pisanju, kao i mogućnost stvaranja deadlock-a. Jedan od glavnih uzroka problema konkurencije je nedovoljno upravljanje pristupom i zaključavanjem resursa u bazi podataka. Kada transakcije ne koriste odgovarajuće mehanizme za zaključavanje, moguće je da se dogode neželjene interakcije, kao što su fantomsko čitanje, izgubljeno ažuriranje ili čitanje prljavih podataka.

U nastavku je dat opis scenarija u kome može da dođe do problema konkurencije. Neka je upravo započela transakcija podizanja novca sa bankomata. Račun je umanjen za traženi iznos i čeka se da bude isplaćen korisniku bankomata kako bi transakcija bila zatvorena. U tom trenutku, ako bi se uradio upit stanja računa, videli

bismo da je stanje umanjeno za iznos koji se podiže sa bankomata. S obzirom da transakcija još uvek nije izvršena, stanje nije realno, jer novac još uvek nije u rukama korisnika. Pretpostavimo da službenik banke baš u tom trenutku odluči da proveri stanje računa navedenog korisnika. Postavlja se pitanje koje stanje će da vidi službenik banke.

Odgovor zavisi od načina na koji se transakcija izvršava, a moguća su sledeća četri slučaja:

- **Prljavo čitanje** (engl.*Dirty Read/Uncommited Read*) je situacija u kojoj jedna transakcija čita podatke koje je druga transakcija privremeno promenila, ali još nije potvrđena promena. Na primer, transakcija A menja određeni red zapisa u bazi podataka, ali još nije završila i potvrdila svoje promene, a transakcija B čita te promene pre nego što su potvrđene, tada dolazi do prljavog čitanja. To može dovesti do problema jer podaci koje je transakcija B pročitala možda nikada neće biti trajno zabeleženi u bazi podataka ako transakcija A na kraju bude poništena ili neuspešna.
- **Neponovljivo čitanje** (engl. *Non-Repeatable Read*) je situacija kada transakcija čita isti red više puta tokom svog izvršavanja, ali dobija različite vrednosti svaki put. Ovo se dešava kada druga transakcija ažurira red zapisa između čitanja od strane prve transakcije. Na primer, zamislimo situaciju gde Transakcija A čita red iz baze podataka, zatim Transakcija B ažurira isti red, i na kraju Transakcija A ponovo čita taj red. Ako Transakcija A dobije različite vrednosti u drugom čitanju u poređenju sa prvom, zbog ažuriranja od strane Transakcije B, došlo je do neponovljivog čitanja.
- **Fantomsko čitanje** (engl. *Phantom Read*) je situacija kada transakcija izvrši SQL upit sa SELECT klauzulom koji vraća skup redova, a zatim u okviru iste transakcije izvrši ponovo upit nad istim skupom podataka, ali dobije različit broj redova, pri čemu se prikazuju novi redovi ili ne prikazuju neki od postojećih redova. Ovo se može dogoditi ukoliko su realizovane INSERT ili DELETE naredbe za dodavanje ili redova od strane drugih transakcija između dva upita unutar prve transakcije.
- **Izgubljeno ažuriranje** (engl. *Lost Update*) je situacija koja se dešava kada više transakcija pokušava da ažurira isti zapis istovremeno ili u kratkom vremenskom intervalu. U ovoj situaciji, jedna od transakcija može nadjačati promene koje je napravila druga transakcija, što dovodi do gubitka ili "izgubljenog" ažuriranja. Na primer, pretpostavimo da Transakcija A čita zapis reda iz baze podataka. Neka postoji u tom redu neka vrednost 10. Transakcija A pokušava da ažurira ovu vrednost uvećavanjem za 10 (10+10). U isto vreme, Transakcija B takođe čita istu vrednost reda, i pokušava da ažurira navedenu vrednost uvećavanjem za 20 (10+20). Transakcija A očekuje krajnju vrednost od 20, a Transakcija B očekuje krajnju vrednost od 30. Ako se Transakcija B izvrši nakon Transakcije A pre nego što Transakcija A potvrdi svoje ažuriranje, tada će ažuriranje koje je izvršila

Transakcija A biti izgubljeno jer će biti nadjačano ažuriranjem koje je izvršila Transakcija B. Ukoliko se ipak izvrši Transakcija A, a zatim Transakcija B krajnja vrednost će biti 40 što nije željeni rezultat ažuriranja.

11.5 ZAKLJUČAVANJE

Zaključavanje (engl. *Locks*) je mehanizam koji se koristi za rešavanje problema konkuretnosti u transakcijama relacionih baza podataka. Osigurava doslednost podataka i sprečava neželjene interakcije između transakcija koje rade sa istim podacima istovremeno. Kada transakcija želi pristupiti određenim podacima, može postaviti zaključavanje na te podatke kako bi privremeno sprečila druge transakcije da ih čitaju ili ažuriraju.

Postoje različite vrste zaključavanja i to:

- **Čitanje-zaključavanje** (engl. *Read Locks*): Transakcija postavlja čitanje-zaključavanje na podatke koje čita, čime sprečava druge transakcije da ih ažuriraju dok ona čita. Ovo omogućava transakciji da sigurno pristupi podacima bez rizika od konflikta sa drugim transakcijama koje žele da ih menjaju. Međutim, drugim transakcijama je i dalje dozvoljeno čitanje istih podataka, što omogućava paralelni pristup podacima, ali sprečava njihovo ažuriranje sve dok se čitanje-zaključavanje ne ukloni.
- **Pisanje-zaključavanje** (engl. *Write Locks*): Transakcija postavlja pisanje-zaključavanje na podatke koje ažurira, čime sprečava druge transakcije da ih čitaju ili ažuriraju dok ona vrši ažuriranje. Ovo osigurava da nijedna druga transakcija neće imati pristup tim podacima dok se izvršava ažuriranje, što pomaže u očuvanju doslednosti i integriteta podataka. Nakon što se ažuriranje završi i zaključavanje ukloni, drugim transakcijama će ponovo biti dozvoljeno čitanje i ažuriranje podataka.
- **Ekskluzivno zaključavanje** (engl. *Exclusive Locks*): Ovo je zaključavanja koja potpuno blokira drugim transakcijama pristup podacima nad kojima je zaključavanje primenjeno, sve dok se to zaključavanje ne ukloni. Kada jedna transakcija postavi ekskluzivno zaključavanje na određene podatke, nijedna druga transakcija neće moći ni da čita ni da ažurira te podatke dok se zaključavanje ne oslobodi. Na ovaj način samo jedna transakcija ima ekskluzivan pristup tim podacima u određenom trenutku. Ova vrsta zaključavanje se često koristi kada je potrebno izvršiti operacije koje zahtevaju da se podaci ne menjaju tokom izvršavanja transakcije, kako bi se osigurala doslednost podataka ili izbegle neželjene posledice prilikom interakcije sa drugim transakcijama. Na primer, može se koristiti prilikom ažuriranja reda u tabeli gde je važno da nijedna druga transakcija ne vrši čitanje ili ažuriranje tog reda u isto vreme, kako bi se izbeglo gubljenje ažuriranja i pojave neusaglašenosti podataka.
- **Deljeno zaključavanje** (engl. *Shared Locks*): Ovo je vrsta zaključavanja koja omogućava da više transakcija čita iste podatke istovremeno, ali sprečava bilo

koju transakciju da ih ažurira dok je zaključavanje aktivno. Kada transakcija postavi deljeno zaključavanje na određene podatke, druge transakcije će moći da čitaju te podatke, ali neće moći da ih ažuriraju dok se zaključavanje ne ukloni. Deljeno zaključavanje je korisno kada je potrebno omogućiti paralelno čitanje podataka od strane više transakcija, ali istovremeno se želi sprečiti bilo kakva izmena tih podataka.

11.5.1 PESIMISTIČKI I OPTIMISTIČKI PRISTUP

Pesimističko i optimističko zaključavanje su dva pristupa upravljanju konkurentnim pristupom podacima u relacionim bazama podataka.

Pesimističko zaključavanje je pristup upravljanju konkurentnim pristupom podacima u relacionim bazama podataka u kojem se pretpostavlja da će doći do konflikta pristupa podacima. Ekskluzivno i deljeno zaključavanje se smatraju pesimističkim vrstama zaključavanja.

Pesimistički pristup se zasniva na tome da se resursi zaključavaju odmah nakon što se pristupi, sprečavajući tako bilo koju drugu transakciju da pristupi tim resursima dok ih trenutna transakcija koristi. Kada se koristi pesimističko zaključavanje, transakcije često dobijaju ekskluzivan pristup resursima koje koriste. To znači da druge transakcije moraju čekati dok se resursi ne oslobode. Ovaj pristup je koristan u situacijama kada su transakcije dugotrajne ili kada postoji visok nivo konkurencije za pristup određenim resursima.

Glavna prednost pesimističkog zaključavanja je što smanjuje mogućnost sukoba pristupa podacima i održava doslednost podataka. Međutim, jedan od nedostataka ovog pristupa je što može dovesti do situacije u kojoj transakcije čekaju na resurse, što može smanjiti performanse sistema, posebno u slučaju visokog opterećenja. U suštini, pesimističko zaključavanje se koristi kao mera za sprečavanje mogućih konflikata pristupa podacima, iako to može dovesti do neefikasnog korišćenja resursa u određenim scenarijima.

Optimističko zaključavanje je pristup upravljanju konkurentnim pristupom podacima u relacionim bazama podataka u kojem se pretpostavlja da će konflikti biti retki. Ovaj pristup se razlikuje od pesimističkog pristupa jer ne zaključava resurse odmah nakon pristupa, već dopušta transakcijama da se izvrše neometano. Provera konflikata se obično vrši tek pri kraju transakcije.

Kada koristimo optimističko zaključavanje, transakcije rade sa podacima kao da neće biti konflikta, iako postoji mogućnost da će se dogoditi. Tek nakon što transakcija završi svoje izvršavanje, sistem proverava da li je došlo do konflikta. Ako nema konflikta, transakcija se završava uspešno. Međutim, ako se otkrije konflikt, transakcija se poništava i može se ponovo izvršiti.

Glavna prednost optimističkog zaključavanja je to što omogućava veću paralelnost između transakcija i smanjuje potrebu za čekanjem na resurse. To može značajno poboljšati performanse sistema, posebno u situacijama kada su transakcije kratkotrajne i kada su konflikti retki.

Međutim, jedan od nedostataka ovog pristupa je što može dovesti do situacije u kojoj se transakcije moraju ponoviti zbog detektovanih konflikata, što može povećati opterećenje sistema. Takođe, optimističko zaključavanje zahteva da transakcije budu dobro dizajnirane i da se oslanjaju na preciznu kontrolu verzija podataka kako bi se sprečili konflikti i očuvala doslednost podataka.

Vrste zaključavanja koje se smatraju optimističkim uključuju:

- **Verziono zaključavanje** (engl.*Version-based Locking*). Ova tehnika koristi verzije podataka kako bi se pratilo stanje podataka tokom izvršavanja transakcija. Umesto da se zaključavaju resursi, transakcije čitaju i ažuriraju podatke, a zatim se proverava da li je došlo do konflikta kada se pokušava završiti transakcija. Ako se otkrije konflikt, transakcija se poništava i može se ponovo izvršiti.
- **Optimističko konkurentno izvršavanje** (engl.*Optimistic Concurrency Control*). Ovaj pristup omogućava transakcijama da čitaju i ažuriraju podatke neometano, bez zaključavanja resursa. Konflikti se proveravaju pri kraju transakcije, a ako se otkrije konflikt, transakcija se poništava i može se ponovo izvršiti.

Prethodno navedene vrste zaključavanja se koriste kako bi se omogućila veća paralelnost između transakcija i smanjila potreba za čekanjem na resurse. To može značajno poboljšati performanse sistema, posebno u situacijama kada su transakcije kratkotrajne i kada su konflikti retki. Međutim, one zahtevaju preciznu kontrolu verzija podataka i dobro dizajniranu logiku upravljanja transakcijama kako bi se osigurala doslednost podataka.

Iako se zaključavanje koristi kao jedan od ključnih mehanizama za održavanje integriteta u relacionim bazama podataka, može dovesti do problema sa performansama ako se koristi neefikasno ili ako se zaključavanje traje duže vreme. Jedan od neželjenih problema je pojava deadlock-a.

11.5.2 DEADLOCK

Deadlock predstavlja situaciju u kojoj dve ili više transakcija ostaju blokirane jer svaka od njih čeka na resurs koji je zaključan od strane druge transakcije. Ovo se dešava kada transakcija poseduje resurs koji je potreban drugoj transakciji kako bi nastavila sa izvršavanjem, dok istovremeno čeka na resurs koji je zaključan od strane druge transakcije. Kao rezultat toga, nijedna od transakcija ne može da se završi. Najčešće je posledica pokušaja transakcija da dobiju pristup istim resursima, ali u različitim redosledima.

Na primer, transakcija A može zaključati resurs X, a zatim pokušati da pristupi resursu Y, dok transakcija B istovremeno zaključava resurs Y i pokušava pristupiti resursu X. Ako se ove dve transakcije nađu u takvoj situaciji, nastaje deadlock. Da bi se rešio problem deadlock-a, sistem mora prepoznati da se deadlock dogodio i preduzeti odgovarajuće korake za razrešenje. Postiže se prekidom jedne od blokiranih transakcija kako bi se oslobodili resursi koje drži, što omogućava drugim transakcijama da nastave sa radom.

Međutim, bolje je sprečiti deadlock nego ga rešavati. Sprečavanje deadlocka moguće je primenom neke od sledećih strategija:

- **Pravilan redosled zaključavanja**. Transakcije treba da zaključavaju resurse u istom redosledu kako bi se izbegli deadlock-ovi. Na primer, ako transakcija A zaključa resurs X pa zatim resurs Y, transakcija B treba da sledi isti redosled zaključavanja.
- **Korišćenje timeout-a**.Moguće je odrediti vremenski period čekanja transakcije na oslobađanje ciljnog resursa, pre nego što transakcija odustane ili bude opozvana. Ukoliko transakcija ne uspe da dobije pristup resursu u određenom vremenskom periodu, može biti prekinuta kako bi se izbegao deadlock. Ovo je efikasan način za sprečavanje blokiranja transakcija koje čekaju na resurse koji su trenutno zaključani od strane drugih transakcija.
- **Upotreba određenih nivoa izolacije**. Primena određenih izolacionih nivoa može pomoći u smanjenju rizika od deadlock-a, ali ne može ga u potpunosti eliminisati. Neki izolacioni nivoi, poput READ COMMITTED, omogućavaju transakcijama da drže zaključavanje samo dok aktivno rade sa resursima, što može smanjiti mogućnost stvaranja deadlock-a.

11.6 IZOLACIONI NIVOI

Izolacioni nivoi utiču na način izvršavanja komandi unutar transakcija. Predstavljaju određene šeme izolacije prema kojima se transakcije izvršavaju. Ove šeme definišu koliko i na koji način transakcija ima pristup podacima u bazi tokom svog izvršavanja, što utiče na to kako se promene u podacima reflektuju unutar same transakcije i u odnosu na druge transakcije.

Postoje sledeći izolacioni nivoi:

- **READ UNCOMMITTED**. Zaključavanje traje samo dok traju i izmene. Ako se dogode izmene u jednoj transakciji, odmah su vidljive i za druge transakcije, iako još uvek nisu prihvaćene. Zbog toga se ovaj izolacioni nivo naziva još i "dirty read".
- **READ COMMITTED**. Promene unutar transakcija vidljive su u drugim transakcijama tek nakon naredbe COMMIT, odnosno uspešnog završetka transakcije. Ovo znači da će SELECT unutar transakcija imati različite vrednosti u zavisnosti od toga da li su podaci promenjeni u nekoj drugoj transakciji koja je

završena. Za razliku od READ UNCOMMITTED, izolacija SELECT naredbi je bolja, ali ne i najbolja.

- **REPEATABLE READ**. Transakcija ekskluzivno zaključava resurse na kojima piše i deljeno resurse koje čita. Zaključavanje traje sve dok traje i transakcija, pa ne može doći do izmena u podacima tokom transakcije. Ovo je podrazumevani izolacioni nivo MySQL servera.
- **SERIALIZABLE**. Podrazumeva najviši stepen izolacije. Svaka transakcija je izolovana i drži resurse ekskluzivno. Za razliku od izolacionog nivoa REPEATABLE READ, jedina razlika je ta što se SELECT komande automatski izvršavaju kao da su napisane upotrebom sintakse SELECT … LOCK IN SHARE MODE.

11.7 PITANJA I ZADACI

1. Objasniti značaj mehanizma transakcija u relacionim bazama podataka.
2. Objasniti principe ACID modela.
3. Šta radi naredba COMMIT?
4. Šta radi naredba ROLLBACK?
5. Na koji način se može kontrolisati tok transakcije?
6. Navesti i objasniti probleme koji nastaju kao posledica konkurentnosti transakcija.
7. Objasniti mehanizam zaključavanja.
8. Navesti različite vrste zaključavanja.
9. Objasniti deadlock i kako ga sprečiti.
10. Objasniti koncept izolacionih nivoa.

LITERATURA

[1] J. L. Johnson, *Database: Models, Languages, Design*, Publisher: Oxford University Press, 1997.

[2] D.M. Kroenke, D J. Auer, *Database Concepts*, 4th Edition, Publisher: Pearson College Div; 2009.

[3] Silberschatz, H. F. Korth, S. Sudarshan, *Database System Concepts*, 6th Edition, Publisher: McGraw-Hill Education, 2010.

[4] C.Coronel, S. Morris, P. Rob, *DATABASE SYSTEMS: Design, Implementation and Management*, 9th Edition, Publisher: Joe Sabatino, 2011.

[5] J. Murach, *Murach's MySQL*, Publisher: Mike Murach & Associates; 5/14/12 edition, 2012.

[6] L. Stoimenov, *Uvod u baze podataka*, Univerzitet u Nišu, Elektronski fakultet, Edicija: Udžbenici, 2013.

[7] R.Elmasri, S. Navathe, *Fundamentals of Database Systems*, 7th Edition, Publisher: Pearson, 2015.

[8] D.Milošević, *Materijali predavanja*, VIŠER, 2018.

[9] T.Kramberger, S.Duk, R.Kovačević, *Baze podataka*, Tehničko veleučilište u Zagrebu, 2018.

[10] T. Connolly, *Database Systems: A Practical Approach to Design, Implementation, and Management*, 6th Edition, Publisher: PEARSON INDIA, 2019.

[11] A.Beaulieu, *Learning SQL*, 3rd Edition, Publisher: O'Reilly Media, Inc., 2020.

[12] A.Molinaro, R. de Graaf, *SQL Cookbook*, 2nd Edition, Publisher: O'Reilly Media, Inc., 2020.

[13] S. Smirnova, A.Tezuysal, *MySQL Cookbook*, 4th Edition, Publisher: O'Reilly Media, Inc., 2022.

[14] M. Reed, *SQL: 3 books 1 - The Ultimate Beginner, Intermediate & Expert Guides To Master SQL Programming Quickly with Practical Exercises*, Publisher : Independently published, 2022.

[15] M.Veinović, G. Šimić, A. Jevremović, M. Tair, *Baze podataka*, Univerzitet Singidunum, Beograd, 2022.

INDEKS POJMOVA

K

L

M

N

O

P

R

S

T

U

W

Z

Θ