# Distribuirane relacijske baze podataka

**Sandri, Christian**

**Master's thesis / Diplomski rad**

**2024**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:217:086051

**SVEUČILIŠTE U ZAGREBU**
**PRIRODOSLOVNO-MATEMATIČKI FAKULTET**
**MATEMATIČKI ODSJEK**

Christian Sandri

# Distribuirane relacijske baze podataka

Diplomski rad

Voditelji rada:
prof. dr. sc. Robert Manger,
doc. dr. sc. Ana Klobučar Barišić

Zagreb, listopad 2024.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

*“Based”*

# Sadržaj

# Uvod

U ovom radu ćemo se baviti distribuiranim relacijskim bazama podataka. Prvo ćemo definirati centraliziranu relacijsku bazu podataka kako bismo vidjeli svojstva koja ona ima. Ta ista svojstva želimo i u distribuiranom slučaju. Nakon toga ćemo se baviti prednostima i manama distribuirane relacijske baze podataka i njezinim dizajnom. Kod dizajna ćemo istražiti metode particioniranja baze podataka, to jest, kako iz centralizirane relacijske baze podataka dobiti distribuiranu relacijsku bazu podataka. Nakon metoda particije ćemo se baviti metodama obrade distribuiranih upita. U ovom dijelu ćemo analizirati postupke koje distribuirani sustav za upravljanje bazom podataka provodi kako bi osigurao efikasnost distribuirane baze podataka. Na kraju ćemo se baviti protokolima za čuvanje integriteta distribuirane baze podataka te mogućim jednostavnim kvarovima i njihovim rješenjima u kontekstu određenih protokola. Na kraju rada se nalazi opis studijskog primjera na temelju kojeg su dani primjeri u ostalim poglavljima.

U radu ću koristiti jezik SQL, koji je standard za DBMS-ove za relacijske baze podataka. U SQL-u upite postavljamo koristeći naredbu SELECT, koja, na temelju drugih ključnih riječi i vrijednosti unutar same naredbe, vraća privremenu tablicu kao rezultat. DBMS izvrednjava upite tako da ih pretvori u ekvivalentne izraze u relacijskoj algebri. O tome ćemo pričati više u četvrtom poglavlju.

U relacijskoj algebri ćemo koristiti sljedeće operacije:

1) Selekcija iz tablice *R* s predikatom *p* vraća retke koji zadovoljavaju predikat *p:*
   $$\sigma_p(R)$$
2) Projekcija skupa atributa *A* iz tablice *R* vraća podtablicu koja sadrži projicirane stupce:
   $$\Pi_A(R)$$
3) Kartezijev produkt tablica *R* i *S* vraća tablicu koja je dobivena Kartezijevim produktom tablica *R* i *S*:
   $$R \times S$$
4) Spoj tablica *R* i *S* po predikatu *p*, koji je tipa *R.A = S.A*, gdje je *A* zajednički atribut tablica *R* i *S*, je tablica koja sadrži retke tablice *R* spojene s redcima tablice *S* takvima da predikat *p* vrijedi:
   $$R \bowtie_p S$$
5) Poluspoj tablica *R* i *S* po predikatu *p* je tablica koja se sastoji od redaka tablice *R* koji bi sudjelovali u spoju tablica *R* i *S* po predikatu *p*:
   $$R \ltimes_p S$$
6) Unija tablica *R* i *S*, takvih da *R* i *S* imaju istu shemu, je tablica koja sadrži sve retke tablice *R* i tablice *S*:

$$R \cup S$$

7) Presjek tablica *R* i *S*, takvih da *R* i *S* imaju istu shemu, je tablica koja sadrži sve retke koji se nalaze u tablici *R* i u tablici *S*:

$$R \cap S$$

# 1. Relacijske baze podataka

Baze podataka su sustavi čiji je cilj trajno pohranjivanje velikih količina podataka u vanjskoj memoriji računala. Baza podataka nije bilo koja kolekcija podataka, već kolekcija podataka koja ima posebna svojstva: čuvanje integriteta (integrity control), istovremeni pristup (concurrency control), otpornost na kvarove (fault tolerance), zaštita od neovlaštene uporabe, efikasan pristup podatcima te mogućnost podešavanja za razne primjene.

*Primjer 1.1*
Excel tablica nije baza podataka jer ne zadovoljava gore navedena svojstva: npr. istovremeni pristup jer istu excel datoteku ne mogu istovremeno mijenjati više korisnika. Ako dva korisnika otvore istu verziju datoteke, bilo koje promjene koje jedan korisnik pohrani neće biti vidljive drugom korisniku bez ponovnog otvaranja ("refreshanja") datoteke.
Excel tablica također nema nikakvu kontrolu integriteta.
◇

Relacijske baze podataka su podskup baza podataka koji se sastoji samo od baza podataka koje koriste relacijski model povezivanja podataka. U relacijskom modelu imamo tablice koje se sastoje od redaka i stupaca. Unutar tablice svaki redak predstavlja jednu n-torku koja pripada relaciji koju ta tablica predstavlja. Svaki stupac u tablici odgovara nekom tipu podataka n-torke. Relacijske baze podataka također imaju mogućnost upita, pomoću kojih možemo, iz trajno pohranjenih tablica u bazi, graditi privremene tablice koje će prikazati podatke koji su nam u danom trenutku od interesa. Najčešći jezik za modeliranje i pristupanje relacijskim bazama podataka je SQL, pa ćemo se njime služiti.

Dodatnu strukturiranost baza podataka postižemo primjenjivanjem normalnih formi, koje pojednostavljuju shemu baza podataka dodavanjem ograničenja. Pojednostavljenu shemu je naizgled teže modelirati jer zahtjeva spajanje više tablica za željene rezultate, ali takva shema olakšava održavanje i fleksibilnost baze za daljnja proširenja na nezamjenjiv način.

*Primjer 1.2*
Ako u bazi podataka imamo tablicu STUDENT, onda možemo u njoj imati uređene četvorke. U tablici pamtimo JMBAG, ime studenta, prezime studenta te godinu studija kao svaki redak, to jest četvorku. Tada za tablicu STUDENT vrijedi:

$$STUDENT \subseteq D_1 \times D_2 \times D_3 \times D_4$$

Gdje $D_1$ predstavlja skup JMBAG-ova (nizovi od točno 11 znamenki), $D_2$ skup imena (nizovi do duljine od 20 znakova), $D_3$ skup prezimena (također nizovi do duljine od 20 znakova), $D_4$ skup godina studija (cijeli brojevi).

Na analogan način definiramo relacije PREDAVAČ, PREDMET, PREDAJE te UPISAO pa imamo sljedeće tablice, koje čine bazu FAKULTET:

STUDENT (jmbag, ime, prezime, godina_studija)
PREDAVAČ (oib, ime, prezime, plaća)
PREDMET (šifra, ime, semestar, ects)
PREDAJE (oib, šifra)
UPISAO (jmbag, šifra, ocjena, godina_upisa)

godina_upisa se odnosi na akademsku godinu u kojoj je upis izvršen, a semestar je redni broj semestra u petogodišnjem studiju.

Sada, ako nas zanimaju parovi student-predavač takvi da su u paru predavač i student koji sluša neki njegov predmet, možemo napisati upit koji će nam vratiti tablicu svih parova (zapravo četvorki) imena i prezimena svih studenata i predavača s gore navedenim kriterijem.
◇

Baze podataka rade pomoću DBMS-a (DataBase Management System), sustava za upravljanje bazom podataka, koji služi kao sučelje između programera baze podataka i samih datoteka, u koje su pohranjeni podaci koje organiziramo i pregledavamo pomoću baze podataka. DBMS prima naredbe, te na temelju njih mijenja ili čita datoteke koje čine bazu podataka i vraća programeru povratne informacije. Realizacija cijelog niza naredbi, od početka do kraja, naziva se transakcija.

## 1.1 Svojstva baza podataka

Svojstva navedena u prethodnom poglavlju ćemo sada preciznijie definirati i kratko opisati mehanizme kojima se ta svojstva osiguravaju.

**Čuvanje integriteta** je istovremeno čuvanje korektnosti i konzistentnosti

**Čuvanje korektnosti** je svojstvo da se u našoj bazi podataka ne pojavljuju semantičke pogreške. Baze podataka obično imaju veliki promet u smislu broja upita, to jest konstantno se događaju čitanja, pisanja ili brisanja podataka iz baze podataka. Ovakav veliki broj upita povećava vjerojatnost da će neko pisanje u bazu htjeti napisati neku informaciju koja nema smisla, pa prema tome ne može biti istinita. Takve slučajeve želimo automatski odbaciti i javiti korisniku ili programeru grešku da je probao upisati nešto semantički besmisleno.

Da bi ovakve probleme izbjegli, baza podataka zahtjeva definiranje tipa atributa i omogućuje pisanje provjera (checks) ili okidača (triggers) za kompleksnije provjere podataka koji se pišu u bazu. U slučaju nepodudaranja tipa podataka koji se pokušava unijeti u bazu ili u slučaju neprolaska dodatne provjere, baza podataka vraća grešku i abortira transakciju.

*Primjer 1.3*
Ako za studente u našoj bazi podataka pamtimo JMBAG, ne bismo htjeli da da netko unese neki niz znakova, koji sadrži slovo, kao nečiji jmbag. Onda se očito čini kao dobra ideja definirati JMBAG kao int (integer, to jest cijeli broj), ali to ne bi bilo dobro jer JMBAG može sadržavati vodeće nule. Prvo sljedeće rješenje bi bilo da se JMBAG definira kao niz od 11 znakova, ali dodamo provjeru (check) tako da taj niz znakova može sadržavati samo znamenke.
◇

**Čuvanje konzistentnosti** je svojstvo da svaka baza podataka koja je u konzistentnom stanju, nakon provođenja transakcije, opet bude u konzistentnom stanju. Za bazu podataka kažemo da je konzistentna ako su joj svi podatci usklađeni, to jest nema kontradiktorne podatke. Isto kao u slučaju čuvanja korektnosti, zbog velikog broja upita, može se dogoditi pokušaj unosa neke vrijednosti u bazu, koja bi trebala biti ista kao neka druga već postojeća vrijednost (jer se radi o referenci), ali, greškom korisnika ili programera, to nije slučaj. Opet želimo automatski odbaciti sve takve slučajeve i javiti vrstu greške korisniku ili programeru.

Za izbjegavanje takvih problema koristimo ograničenja (constraints):

PRIMARY KEY, koji određuje neki stupac kao primarni ključ. Stupac koji je primarni ključ služi kao informacija kojom se dohvaća neki redak tablice. Očito svaka vrijednost svakog primarnog ključa mora biti jedinstvena jer kad ne bi bila, ne bismo mogli jedinstveno odabrati neki redak tablice za ispisivanje ili uređivanje. Također, iz istog razloga, primarni ključ ne može imati null vrijednost. Primarni ključ može biti više stupaca u slučaju da tablica nema jedan stupac koji jedinstveno određuje redak.

NOT NULL, koji brani unosa retka u tablicu bez unesene vrijednosti u tom stupcu.

UNIQUE, koji određuje da neki stupac tablice ne može imati dvije iste vrijednosti, ali može biti null. Možemo obuhvatiti više stupaca istim unique, tada k-torka stupaca mora biti jedinstvena unutar tablice.

FOREIGN KEY … REFERENCES, koji određuje da je neki stupac u tablici strani ključ koji se referencira na neki stupac u nekoj drugoj tablici. To znači da, pri unosu retka u tablicu koja ima strani ključ, ne možemo unijeti redak koji za vrijednost stranog ključa ima neku vrijednost koja se ne pojavljuje u stupcu koji referencira taj strani ključ. Također nećemo moći promijeniti vrijednost koja se pojavljuje u nekoj tablici kao strani ključ.

*Primjer 1.4*
Za tablicu PREDAVAČ iz *Primjera 1.2* smo uzeli oib kao primarni ključ jer možemo očekivati da nećemo imati dva predavača s istim oibom. Za tablicu PREDAJE imamo primarni ključ koji

se sastoji od uređenog para (oib, šifra) jer isti predavač može predavati više predmeta i isti predmet mogu predavati više predavača, pa ne možemo jedinstveno odrediti o kojem se retku radi na temelju samo jednog od tih stupaca. Želimo da većina podataka u tablicama definiranima u *Primjeru 1.2*, bude NOT NULL, ali neke moramo ostaviti da mogu biti null. Naime, u tablici UPISAO pamtimo ocjenu koju je student dobio iz predmeta, ali student upisuje predmet nekoliko mjeseci prije nego što ga položi. Zbog toga želimo da ocjena bude null dok se ne upiše. Alternativno možemo dodati vrijednost -1, koja nam može značiti da ocjena nije upisana, ali to smanjuje transparentnost naše baze podataka i može dovesti do zbunjivanja programera ili korisnika. U tablici UPISAO pamtimo jmbag studenta i šifru predmeta. Oba ta stupca deklariramo kao strane ključeve jer ne želimo da se dogodi upisivanje nepostojećeg studenta na neki predmet ili obratno. Strani ključ će ovdje spriječiti da iz baze podataka izbrišemo nekog studenta koji je upisao neki predmet jer ne bi imalo smisla da pamtimo upise i ocjene studenta kojeg smo izbrisali. Nakon brisanja upisa, kada se jmbag nekog studenta ne pojavljuje više kao strani ključ, možemo tog studenta izbrisati iz baze.

◇

**Efikasan pristup** postaje netrivijalan čim se radi o većim količinama podataka. Od baze podataka se očekuje da u što manjem vremenskom roku dohvati ili izmijeni podatke. Za izmijeniti podatak, prvo trebamo znati gdje je on pohranjen. Za dohvatiti podatak također trebamo znati gdje je pohranjen, pa se problem efikasnog pristupa svodi na problem pretraživanja baze podataka.

Za rješenje tog problema se služimo indeksima. Indeksi rade pomoću pointera, sortiranja i B-stabala, te osiguravaju brže pretraživanje po bazi uz potrebu za više memorije. Indeksi ne mijenjaju samu bazu podataka, već kreiraju nove podatke pomoću kojih se u manje koraka pronađe traženi redak, ali te podatke treba negdje i spremiti, jer su oni trajni.

**Istovremeni pristup** je obično dio DBMS-a. Ne možemo serijski izvršavati transakcije jer bi to predugo trajalo, zbog samog broja transakcija koji se mora izvršavati. Svaku veću bazu će istovremeno koristiti velik broj korisnika koji ne znaju jedan za drugoga.

Da bismo to omogućili, istovremeno izvršavamo više transakcija, dokle god dvije transakcije ne pristupaju istom dijelu baze podataka u isto vrijeme. Da bi se spriječilo istovremeno pristupanje istom dijelu baze podataka od strane više transakcija, koriste se lokoti. Kada transakcija pristupa podatku, ona zaključa dio baze podataka u kojoj se taj podatak nalazi. Kada transakciji taj podatak više nije potreban, ona ga otključa. Ovdje se pojavljuje problem blokade (deadlock): jedna transakcija zaključa jedan dio baze podataka, druga transakcija zaključa drugi dio baze podataka. Sada obje transakcije žele pristupiti podatku koji je druga transakcija zaključala i obje čekaju jedna drugu u nedogled. Jedno rješenje je periodična provjera postoji li blokada, te ako postoji, jedna transakcija se obustavlja i nakon kratkog vremena ponovno pokreće.

**Otpornost na kvarove** želimo jer u baze podataka pohranjujemo bitne podatke i u velikim količinama, pa bi bilo kakav gubitak podataka imao visoku cijenu. Bez planiranja unaprijed i preparacija, bilo kakav gubitak ili korupcija podataka se čine nepopravljivi.

Jedno rješenje za oporavak baze podataka nakon kvara je periodična izrada rezervne kopije (backup). Što češće radimo rezervnu kopiju, to ćemo imati ažurniju bazu u slučaju oporavka, ali pri češćoj izradi rezervne kopije, raste cijena održavanja baze podataka i smanjuje se njezina dostupnost za vrijeme izrade rezervne kopije.

**Zaštita od neovlaštene uporabe** je obično zadaća administratora baze (DBA - DataBase Admin). Neki podatci mogu biti povjerljivi, pa ne želimo da im svatko može pristupiti. Također nekom skupu korisnika ne želimo dati da unose podatke u bazu, već samo da ih mogu čitati; potencijalno nekom skupu korisnika želimo zabraniti brisanje podataka iz baze.

DBMS i baza podataka imaju sustav autentikacije korisnika, recimo username i password. DBA dodjeljuje korisnicima ovlasti za tipove naredbi koje određeni korisnik može izvršavati. Također baze podataka imaju mogućnost pogleda (views). Pogledi su virtualne, ali trajne tablice, koje su napravljene selekcijom podataka iz već postojećih tablica. DBA može napraviti pogled koji sadrži neki podskup podataka iz baze podataka i onda pristup tom pogledu pridružiti nekom skupu korisnika umjesto cijele baze podataka.

*Primjer 1.5*
Možemo za bazu definiranu u *primjeru 1.2* htjeti da predavači mogu vidjeti pa i unositi samo ocjene za predmete koje predaju i to samo studentima kojima predaju. Tada bismo mogli napraviti view u kojem se nalaze, za dani oib, samo redci tablice UPISAO koji sadrže šifru predmeta koje predavač s tim oibom predaje.
◇

**Mogućnost podešavanja za razne primjene** je rezultat gore navedenih svojstava i činjenice da su većina njih podesiva. Ovisno o primjeni imamo različite konfiguracije mehanizama za otpornost kvarova i zaštite od neovlaštene uporabe. Indeksi su također podesivi ovisno o primjeni. U nekim situacijama se čak i pravila integriteta zanemaruju kako bi se povećala efikasnost.

## 1.2 Normalne forme

Normalne forme su stupnjevi normalizacije baze podataka. Normalizirati bazu podataka znači dodavati pravila koja sve tablice u bazi trebaju zadovoljavati. Što više pravila imamo, to će baza podataka biti normaliziranija, pa će ju biti lakše modelirati jer će podatci imati veća ograničenja. Nametanjem ograničenja na podatke smanjujemo mogući broj različitih tipova

unosa te smanjujemo mogućnost unosa nepoželjnih oblika podataka. Time smanjujemo količinu podataka koje je teško interpretirati. Također znatno olakšavamo održavanje baze podataka jer je lakše oblikovati, ažurirati i korigirati podatke koji su granularniji, te smanjujemo mogućnost redundancije. Kažemo da je relacijska baza podataka u nekoj normalnoj formi ako su joj sve relacije u toj istoj normalnoj formi.

**Prva normalna forma (1NF)** je automatski zadovoljena relacijskim modelom. Zahtjev prve normalne forme je da nemamo višestruke stupce u tablicama, to jest, da je podatak unutar svake ćelije tablice nedjeljiv.

*Primjer 1.6*
Ako bismo u tablici STUDENT htjeli pamtiti, kao neki stupac, popis predmeta koje taj student sluša, to ne bi bilo u 1NF, jer bi taj stupac bio višestruki. Da bismo u tom slučaju pamtili ocjene koje je student dobio iz nekih predmeta, morali bismo imati još jedan višestruki stupac, i kojem bi i-ta ocjena odgovara i-tom predmetu. Takvo rješenje je nepregledno i vrlo lako može doći do rušenja integriteta baze podataka pri bilo kojem upisivanju ocjene ili novog predmeta. Taj problem lako rješavamo prevođenjem u 1NF, to jest, umjesto višestrukih stupaca napravimo novu tablicu PREDMET, te retke te tablice povežemo ključevima tablice STUDENT s redcima iste. Sada se ponovno pojavljuje isti problem u tablici PREDMET, pa dodajemo tablicu UPISAO kao analogno rješenje za analogni problem.
◇

**Druga normalna forma (2NF)** zahtjeva da je relacija u 1NF, te da su svi stupci, koji nisu dio primarnog ključa, potpuno funkcionalno ovisni o primarnom ključu. Kažemo da je stupac potpuno funkcionalno ovisan o nekom skupu stupaca ako je funkcionalno ovisan o tom skupu, ali nije funkcionalno ovisan ni o jednom njegovom pravom podskupu. Kažemo da je stupac funkcionalno ovisan o nekom skupu stupaca, koji može biti jednočlan, ako vrijednosti stupaca tog skupa jednoznačno određuju vrijednost ovisnog stupca.

*Primjer 1.7*
Ako bismo u tablici UPISAO pamtili još i prezime studenta, tada prezime ne bi bilo potpuno ovisno o skupu {jmbag, šifra} jer sami jmbag određuje prezime studenta.
◇

**Treća normalna forma (3NF)** zahtjeva da je relacija u 2NF, te da unutar relacije nemamo tranzitivne funkcionalne ovisnosti. Zapravo želimo granulirati našu preveliku relaciju u više manjih, koje ćemo onda međusobno povezati s još malih relacija.

*Primjer 1.8*
Kad bismo u tablici PREDMET pamtili, uz pretpostavku da svaki predmet predaje najviše jedan predavač, oib predavača koji ga predaje i broj ureda koji pripada predavaču. Sada je broj ureda

ovisan o oibu predavača, koji je ovisan o šifri predmeta, koja je primarni ključ. To je tranzitivna ovisnost, pa ovakva tablica PREDMET ne bi bila u 3NF.
◇

Postoje još i Boyce-Codd-ova (BCNF), četvrta (4NF), peta (5NF) te šesta (6NF) normalna forma koje nameću još rigoroznija pravila na relacije, ali nisu zanimljive u kontekstu demonstrativne baze podataka FAKULTET, jer većina tablica koje zadovoljavaju 3NF obično zadovoljavaju i više normalne forme osim u nekim rubnim slučajevima.

Baza podataka FAKULTET zadovoljava sve gore navedene normalne forme. Normalizaciju baze obično gradimo od nižih formi prema višima, iako dobro dizajniranu bazu podataka ne treba puno normalizirati, te se neki koraci u normalizaciji mogu preskočiti jer će njihovi uvjeti od početka biti zadovoljeni.

## 1.3 Transparentnost

Još jedno svojstvo baza podataka je da su transparentne prema korisniku ili programeru. Transparentnost u ovom kontekstu znači da korisnikov ili programerov pogled na bazu podataka nije zamagljen ili zamućen detaljima o radu baze podataka, već jedino što on vidi ili dodiruje su usluge koje baza podataka pruža. Kao što korisnik aplikacije ne mora znati na koji način je ona implementirana, tako programer ili korisnik baze podataka ne mora znati na koji način točno ona radi. Što manje korisnici znaju o detaljima baze, to je baza sigurnija od napada koji bi ju htjeli oštetiti ili dočepati se nekih povjerljivih podataka. Razvoj kompleksnih aplikacija je također jednostavniji uz transparentniju bazu podataka.

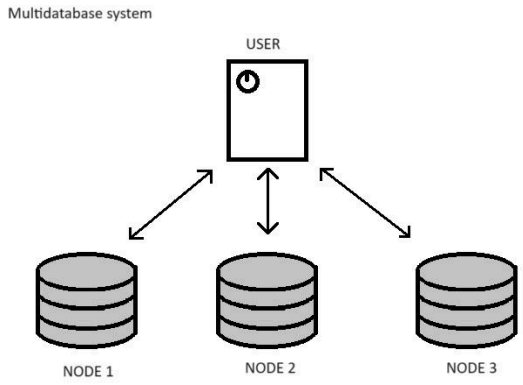
U kontekstu distribuiranih baza podataka transparentnost obuhvaća i mrežu preko koje distribuirani čvorovi komuniciraju, način replikacije te način fragmentacije, o kojima ćemo se baviti kasnije. Generalno, većinu transparentnosti nam daje DBMS, ali i dizajn baze igra veliku ulogu.

# 2. Motivacija za distribuciju

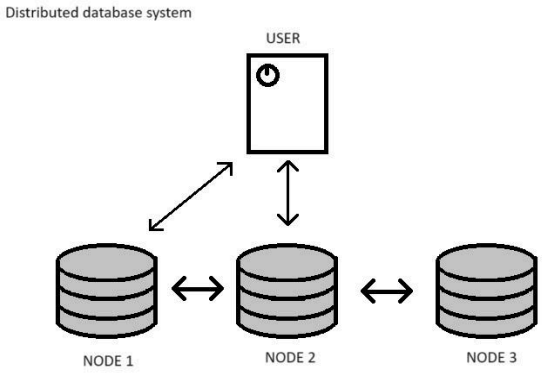
U prvom poglavlju smo definirali što je to centralizirana baza podataka. U ovom poglavlju ćemo pričati o razlozima zašto bismo htjeli imati distribuiranu bazu podataka umjesto centralizirane. Navest ćemo prednosti ali i probleme koji se pojavljuju pri izradi i održavanju distribuirane baze podataka.

Distribuirana baza podataka je baza podataka koja će se krajnjem korisniku ili programeru činiti kao centralizirana, ali će podatci te baze biti pohranjeni na više čvorova (nodes). Svaki čvor je jedan stroj koji je geografski udaljen od ostalih u distribuiranom sustavu. Svaki čvor se može promatrati kao zasebna baza podataka, ali njegovi podatci će se u nekim kontekstima činiti nepotpuni. Još jedno bitno svojstvo distribuiranih baza podataka je da sve lokalne baze podataka na svakom čvoru međusobno čuvaju integritet. To se postiže spajanjem svih čvorova u jednu povezanu mrežu i dodavanjem složenih algoritama za analizu i provođenje upita i transakcija te komunikacijom između čvorova. Glavni problem u distribuiranoj bazi čini nepouzdanost mreže.

Bitno je razlikovati distribuirane baze podataka od višestrukih baza podataka (multidatabase). Višestruka baza podataka je istovremeno pristupanje više baza podataka, koje su međusobno povezane mrežom, ali, za razliku od distribuirane baze podataka, višestruka baza podataka ne čuva integritet između svojih čvorova. Čvorovi u višestrukoj bazi podataka međusobno ne komuniciraju.




**Slika 2.1** višestruka baza podataka



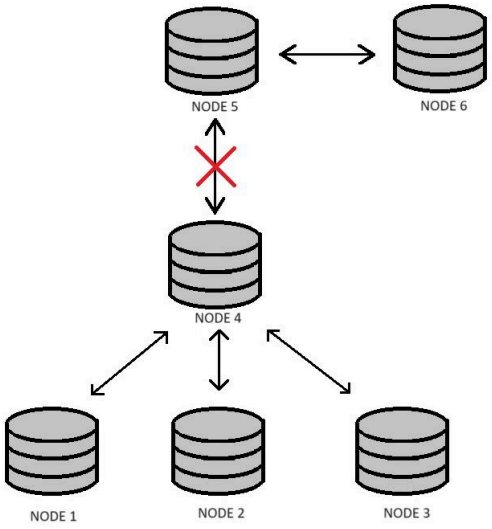

**Slika 2.2** distribuirana baza podataka

## 2.1 Prednosti i problemi distribuiranih baza podataka

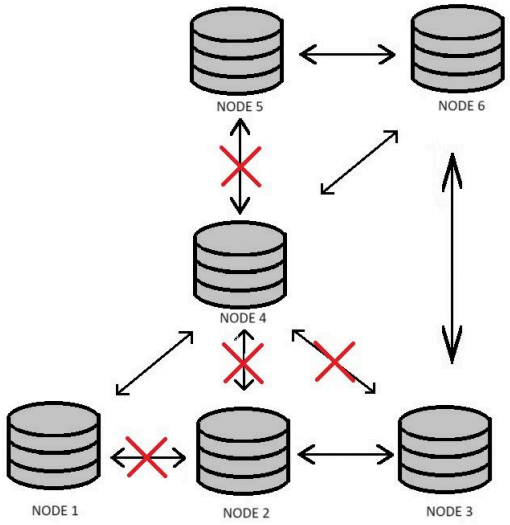
Distribuirane baze podataka su modularnije od centraliziranih baza podataka u smislu da možemo više ili manje resursa trošiti na dostupnost baze podataka, sigurnosne kopije, otpornost na kvar komunikacijske mreže, te na integritet. Naravno, što više resursa trošimo na neki segment, to će taj segment biti kvalitetniji u svojoj zadaći. Kasnije ćemo vidjeti CAP teorem, koji dokazuje da distribuirana baza podataka ne može imati sva gore navedena svojstva u potpunosti zadovoljena u isto vrijeme.

U centraliziranim bazama podataka je očito taj jedan sustav usko grlo u smislu kvara i u smislu opterećenja. Iako možemo spojiti proizvoljan broj aplikacija na bazu podataka, sama baza podataka ima gornju granicu u količini podataka koje može u nekom vremenu primiti i poslati. Iako možemo imati rezervne kopije, one neće biti u potpunosti ažurne i u slučaju kvara centralizirane baze, korisni će trebati čekati da se rezervna kopija ažurira do suvremenog stanja baze podataka i da se dovede u pogon. Distribuirane baze podataka ne moraju imati ovaj problem. Kod distribuiranih baza podataka možemo imati rezervnu kopiju, koja je u potpunosti ažurna. Svaka transakcija koja se provede na jednom čvoru, automatski se provodi i na drugom čvoru. U slučaju kvara jednog čvora, jednostavno preusmjerimo korisnike tog čvora na ostale čvorove koji su egzaktne kopije čvora u kvaru. Ovakav sustav kopija također povećava mogući broj istovremenih korisnika jer možemo dodati proizvoljan broj čvorova koji će međusobno biti egzaktni duplikati.

S druge strane, distribuirana baza podataka se oslanja na komunikacijsku mrežu, koja je nepredvidiva. Ne znamo u kojem trenutku se može dogoditi kvar dijela mreže ili kvar cijele mreže. Otpornost na kvar dijela veze možemo povećati tako da dodamo više komunikacijskih kanala. Više komunikacijskih kanala neće samo povećati otpornost na neke kvarove mreže, već će također povećati brzinu sustava.




**Slika 2.3** Primjer mreže a)




**Slika 2.4** Primjer mreže b)

U svakom trenutku želimo imati potpuno povezanu mrežu. **Slika 2.3** prikazuje slučaj u kojem imamo minimalan potreban broj komunikacijskih kanala za održati povezanost mreže. Tada kvarom jednog kanala možemo izgubiti više od pola baze. Kasnije ćemo vidjeti koji se problemi pojavljuju ako neki čvor ne dobije povratnu informaciju nekog drugog čvora u razumnom roku. **Slika 2.4** prikazuje slučaj kada je dodano više komunikacijskih kanala. Robusnost takve mreže je puno značajnija, a na slici je prikazan kvar skoro pola kanala, unatoč tome mreža je i dalje povezana. Za mrežu kao na **slici 2.3**, da bi čvor 3 komunicirao s čvorom 6, njegovu poruku i njezin odgovor moraju proslijediti čvor 4 i čvor 5, dok za mrežu kao na **slici 2.4** čvor 3 i čvor 4 mogu direktno komunicirati. Očito je komunikacija u drugom slučaju brža nego u prvom. O brzini komunikacije treba voditi računa kada se dizajnira komunikacijska mreža distribuirane baze podataka: želimo da čvorovi, koji će često komunicirati, imaju što kraći put jedan do drugog u smislu broja drugih čvorova koji moraju posredovati u njihovoj komunikaciji.

Još jedan problem je čuvanje integriteta distribuirane baze podataka. U centraliziranoj bazi podataka rješenje problema čuvanja integriteta nije trivijalno. U distribuiranoj bazi podataka je komplikacija još veća. Zbog nepouzdanosti mreže se implementiraju protokoli i kompromisi koji, ovisno o namjeni sustava, daju optimalnu funkcionalnost.

*Primjer 2.1*

Recimo da imamo tri čvora: čvor 1, čvor 2 i čvor 3. Neka postoje dva komunikacijska kanala: između čvora 1 i čvora 2, te između čvora 2 i čvora 3. Neka svaki od tih čvorova ima dio baze FAKULTET:

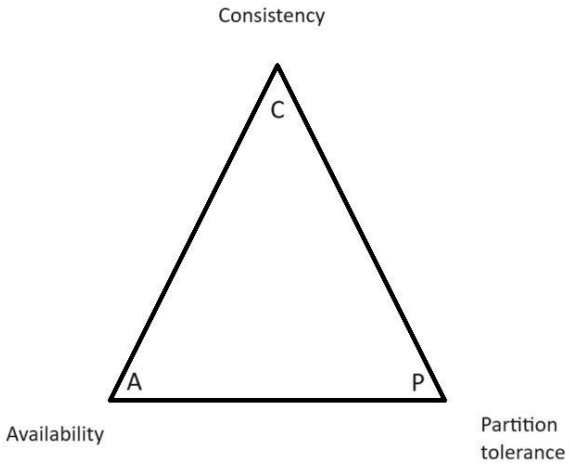
- Čvor 1 pamti tablicu PREDAVAČ, ne pamti tablice STUDENT i UPISAO, te pamti tablice PREDMET i PREDAJE.

- Čvor 2 pamti tablicu STUDENT, PREDMET i UPISAO, ali samo studente i predmete preddiplomskog studija (prve 3 godine), te pamti tablice PREDAVAČ i PREDAJE.
- Čvor 3 pamti tablicu STUDENT, PREDMET i UPISAO, ali samo studente i predmete diplomskog studija (zadnje 2 godine), te pamti tablice PREDAVAČ i PREDAJE.

Pretpostavimo da smo spojeni samo na čvor 1. Pokrenimo sada transakciju koja bi izbrisala jednog predavača *P* te upisala jednog studenta *S1* na prvu godinu i jednog studenta *S2* na petu godinu. Naša transakcija će se propagirati kroz mrežu, ali pretpostavimo da je do kvara u mreži došlo u komunikacijskom kanalu između čvora 2 i čvora 3. Sada zapravo imamo dvije disjunktne mreže. Jedna mreža se sastoji od čvorova 1 i 2, a druga samo od čvora 3. Znamo da se naša transakcija propagirala kroz čvorove 1 i 2. Čvorovi 1 i 2 isto to znaju, ali ni mi ni čvorovi 1 i 2 ne znamo hoće li čvor 3 provesti transakciju. Ne znamo niti zna li čvor 3 da postoji pokušaj provedbe transakcije. Što sada? O rješenju ili kompromisu u ovakvim situacijama ćemo pričati kasnije, ali neizbježnost takvih situacija nam dokazuje CAP teorem.
◇

## 2.2 CAP teorem

***Teorem 2.1*** **(CAP teorem)**
Distribuirana baza podataka može zadovoljavati najviše dva od sljedeća tri svojstva istovremeno:
- Konzistentnost (Consistency)
- Dostupnost (Availability)
- Otpornost na kvar mreže (Partition tolerance)




**Slika 2.5** Ilustracija CAP teorema

U kontekstu distribuirane baze podataka, konzistentnost se treba provoditi kroz više čvorova. Ako se na jednom čvoru unosi neki podatak, čiji se strani ključ referencira na neki podatak na drugom čvoru, mora se provjeriti valjanost tog podatka da bi konzistentnost bila održana. Dostupnost se odnosi na korisnikovu mogućnost da pristupa bazi podataka. Otpornost na kvarove mreže se odnosi na mogućnost baze podataka da nastavi raditi i ako je mreža u kvaru, to jest ako je proizvoljan broj čvorova nedostupan.

*Dokaz:*
Pretpostavimo da vrijedi dva od tri i dobimo da treće ne može vrijediti. Neka je sustav dostupan i otporan na kvarove mreže. Sad neka je mreža odvojena na dvije podmreže: MA i MB (Mreža A i Mreža B). Imamo dva klijenta KA i KB (Klijent A i Klijent B). KA je spojen na čvor iz MA, a KB na čvor iz MB. Neka KA unese novu informaciju u dio baze podataka koji čine čvorovi u MA. Sada neka KB pokuša pročitati sve podatke tog tipa iz baze podataka. Kako je KB spojen na MB, a MB ne komunicira ni s jednim čvorom iz MA, KB neće dobiti informaciju da je KA unio novi podatak u bazu. Dakle, baza podataka je nekonzistentna.

□

U praksi obično distribuirane baze podataka ne zadovoljavaju u potpunosti niti jedno svojstvo CAP teorema, već se rade kompromisi i protokoli ovisno o namjeni i vanjskim uvjetima.

*Primjer 2.2*
Za online trgovinu, ne moramo ažurirati broj dostupnih predmeta svaki put kada kupac stavi predmet u košaricu, već to radimo samo kada kupac zapravo kupi proizvod. U slučaju da kupac pokuša kupiti nešto što nije dostupno, vratit ćemo grešku. Ovdje žrtvujemo konzistentnost ali dobivamo na efikasnosti sustava.
◇

*Primjer 2.3*
Za online kockarnicu nam je iznimno bitno da usluga bude dostupna za vrijeme velikih sportskih događaja. Pa u slučaju pada dijela sustava, veći profit postižemo ako nekim korisnicima upišemo kriva stanja računa i održimo dostupnost, te kasnije to nadoknadimo na svoj trošak. Nekoliko sati nedostupnosti u pravo vrijeme nas može koštati previše.
◇

## 2.3 Dizajn distribuirane baze podataka

Način na koji gradimo distribuiranu bazu podataka može biti od dna prema gore (bottom-up) i od vrha prema dolje (top-down). Gradnja distribuirane baze podataka od dna prema gore je spajanje više disjunktnih, već postojećih centraliziranih baza podataka u jednu distribuiranu bazu podataka. Da bi se to postiglo, potrebno je uskladiti sheme svih tih baza podataka i napraviti novu globalnu shemu koja će pokriti sve usklađene sheme i omogućiti da sve te baze čine cjelinu. Proces je kompliciran i nećemo se više baviti njime u ovom radu, nego ćemo se fokusirati na razvoj prema dolje jer je taj slučaj implementiran u praktičnom dijelu rada.

Razvoj prema dolje uzima jednu centraliziranu bazu podataka i od nje čini distribuiranu bazu podataka. Distribucija se postiže horizontalnom i vertikalnom fragmentacijom, kojima jednu bazu podataka rascijepamo na više njih, te ih onda alociramo na udaljene čvorove.

# 3. Particioniranje

U distribuiranoj bazi podataka ne želimo imati identične replike centralne baze podataka na svakom čvoru. Kako su čvorovi geografski udaljeni, realno je za očekivati da nećemo imati istu frekvenciju pristupanja svim podatcima. Očito će korisnici koji se spajaju na jedan čvor biti skloni manipulirati nekim podskupom naše centralizirane baze podataka. Zbog toga, ako bismo na svakom čvoru čuvali cijelu bazu podataka, trošili bismo resurse bez razloga na veće količine lokalne memorije. Pri particioniranju je neophodno predvidjeti kojem podskupu će se najfrekventnije pristupati na kojem čvoru. Bitno je naglasiti da podatci na čvorovima ne moraju činiti particiju centralizirane baze podataka u smislu da njihovi podatci moraju biti disjunktni. Dapače, neki podatci će često biti potrebni većini čvorova, pa ćemo takve podatke pohraniti na te čvorove. Još jedan bitan razlog za razlamanje baze podataka jest taj da to olakšava izvođenje konkurentnih transakcija. Naime, ako imamo dvije transakcije koje obje moraju pristupiti istoj tablici u centraliziranom smislu, onda svaka može pristupiti svom fragmentu tablice i tako provoditi obje transakcije istovremeno. Bitno je napomenuti da pri particioniranju baze podataka treba imati na umu primjenu baze podataka, te koje će aplikacije pristupati toj bazi podataka.

Bazu podataka particioniramo tako da particioniramo njene relacije, to jest, tablice. Razlamanje relacije zovemo fragmentacija. Tablicu možemo fragmentirati vertikalno, to jest, izdvojiti stupce ili horizontalno, odnosno izdvojiti retke. Sada ćemo definirati svojstva koja fragmentacija mora zadovoljavati da ne bi promijenila semantiku baze podataka:

- **Potpunost** (completeness): Ako je relacija $R$ rastavljena na fragmente $F_R = \{R_1, R_2, \dots, R_n\}$, tada za svaki podatak koji se nalazi u $R$, vrijedi da se taj podatak nalazi u $R_i$ za barem jedan $i \in \{1, 2, \dots, n\}$.
- **Rekonstrukcija** (reconstruction): Ako je relacija $R$ rastavljena na fragmente $F_R = \{R_1, R_2, \dots, R_n\}$, tada postoji operator $\nabla$ takav da
  $R = \nabla_{i=1}^{n} Ri$
- **Disjunktnost** (disjointness): Ako je relacija rastavljena na fragmente $F_R = \{R_1, R_2, \dots, R_n\}$, tada ne postoji podatak $p$ takav da $p \in R_i$ i $p \in R_j$ za $i \neq j$.

Zahtjev za potpunosti je očit. Ne želimo da fragmentacijom izgubimo neke podatke. Zahtjev za rekonstrukcijom zahtjeva da postoji neki operator koji će rekonstruirati bazu podataka kada se primjeni na sve fragmente. Obično je to operator unije ili spoja (join). Želimo moći rekonstruirati cijelu tablicu iz njenih fragmenata pri provođenju upita. Disjunktnost nam je neophodna jer ne želimo imati duplikate podataka kada ne trebamo, te ne želimo provoditi spajanja na duplikatima. Duplikate ćemo imati na repliciranim dijelovima baze, ali ne i na fragmentiranim. Kod horizontalne fragmentacije će disjunktnost biti zadovoljena, a kod vertikalne fragmentacije ćemo podrazumijevati da vrijedi disjunktnost ako su svi podatci osim primarnih ključeva disjunktni. Kasnije ćemo pokazati zašto je bitno replicirati primarne ključeve u vertikalnom slučaju.

## 3.1 Horizontalna fragmentacija

Horizontalna fragmentacija ima dva oblika: primarna horizontalna fragmentacija (primary horizontal fragmentation) i izvedena horizontalna fragmentacija (derived horizontal fragmentation). Primarna horizontalna fragmentacija se postiže definiranjem predikata, to jest, uvjeta nad vrijednostima stupaca tablice. S druge strane, izvedena horizontalna fragmentacija se postiže definiranjem predikata, to jest uvjeta na stupce u tablici koja sadrži stupac na koji se referencira strani ključ tablice koju želimo fragmentirati. Na temelju definiranih predikata razdvajamo retke tablice u skupove koji onda čine podtablice.

*Primjer 3.1*
Ako imamo tablice

PREDAVAČ (<u>oib</u>, ime, prezime, plaća)
PREDMET (<u>šifra</u>, ime, semestar, ects)
PREDAJE (<u>oib</u>, <u>šifra</u>)

tada bi jedna primarna horizontalna fragmentacija tablice PREDAVAČ bila podjela tablice PREDAVAČ na tablice $\text{PREDAVAČ}_{<1000}$ i $\text{PREDAVAČ}_{\geq 1000}$, gdje u tablici $\text{PREDAVAČ}_{<1000}$ pamtimo sve retke tablice PREDAVAČ za koje vrijedi plaća < 1000, a u tablici $\text{PREDAVAČ}_{\geq 1000}$ pamtimo sve retke tablice PREDAVAČ za koje vrijedi plaća $\geq 1000$.

Jedna izvedena horizontalna fragmentacija tablice PREDAVAČ bi podjela tablice PREDAVAČ na temelju vrijednosti semestar u tablici PREDMET. Možemo napraviti tablice $\text{PREDAVAČ}_{\text{diplomski}}$ i $\text{PREDAVAČ}_{\text{preddiplomski}}$ takve da ako postoji predmet na diplomskom dijelu studija (semestar $\in \{7, 8, 9, 10\}$), kojeg neki predavač predaje (predavač i predmet su povezani tablicom PREDAJE), onda ćemo informacije tog predavača pamtiti u tablici $\text{PREDAVAČ}_{\text{diplomski}}$, inače ćemo informacije predavača pamtiti u tablici $\text{PREDAVAČ}_{\text{preddiplomski}}$.

Bitno je napomenuti da ne možemo tražiti da u jednom fragmentu pamtimo predavače diplomskog studija, u drugom predavače preddiplomskog studija, ako može postojati predavač koji predaje neki predmet diplomskog studija i neki predmet preddiplomskog studija. Naime, tada bi oba fragmenta sadržavali isti redak, pa bi gore navedeni uvjet disjunktnosti bio prekršen.
◇

Sada ćemo definirati predikate, pomoću kojih ćemo provoditi horizontalnu fragmentaciju.

***Definicija 3.1*** Jednostavan predikat

Za relaciju $R(A_1, A_2, \dots, A_n)$, gdje je $A_i \in D_i$, $\forall i \in \{1, 2, \dots, n\}$ za $D_i$, kao iz *primjera 1.2*, domena *i*-tog stupca, definiramo jednostavan predikat kao formulu oblika

$$p_k : A_i \ \theta \ Vrijednost$$

gdje je $\theta \in \{<, >, =, \leq, \geq, \neq\}$, a $Vrijednost \in D_i$. Skup jednostavnih predikata nad relacijom $R_i$ pišemo kao $Pr_i$ a elemente tog skupa pišemo $p_{ij}$.

□

*Primjer 3.2*
Za tablicu STUDENT (jmbag, ime, prezime, godina_studija)
neki jednostavni predikati su

$p_1$: godina_studija = 4
$p_2$: jmbag < '1000050000'

ali

$p_3$: godina_studija < 'a'

nije jednostavan predikat.
Naime, 'a' nije iz domene stupca godina_studija, čija domena su cijeli brojevi.
◇

***Definicija 3.2*** Minterm predikat

Minterm predikat je konjunkcija svih jednostavnih predikata iz *Pr* ili njihovim negacija nad nekom relacijom.

Za skup $Pr_i$ $\{p_{i1}, p_{i2}, \dots, p_{in},\}$ jednostavnih predikata definiranih nad relacijom $R_i$, skup minterm predikata $M_i = \{m_{i1}, m_{i2}, \dots, m_{iz},\}$ je dan formulom

$$M_i = \{m_{ij} | m_{ij} = \bigwedge_{p_{ik} \in Pr_i} p_{ik}^*\}, 1 \leq k \leq n, 1 \leq j \leq z$$

gdje je $p_{ik}^* = p_{ik}$ ili $p_{ik}^* = \neg p_{ik}$.

□

Negacija terma je neophodna jer ona daje komplement skupa koji je definiran tim termom. Nekad nećemo zapisati sve jednostavne predikate kao dio konjunkcije jer će se podrazumijevati da jedan jednostavan predikat implicira negacije nekih drugih jednostavnih predikata.

*Primjer 3.3*
Za tablicu STUDENT (jmbag, ime, prezime, godina_studija)
neki jednostavni predikati su

$p_1$: ime = 'Ivan'
$p_2$: ime = 'Marko'
$p_3$: studija < 3
$p_4$: godina_studija $\geq$ 2
$p_5$: jmbag > '1000050000'

Neki minterm predikati definirani na temelju gore navedenih jednostavnih predikata
za $Pr = \{p_1, p_2, p_3\}$:
$m_1$: $p_1 \wedge \neg p_2 \wedge \neg p_3 \equiv$
$\equiv$ (ime = 'Ivan') $\wedge \neg$(ime = 'Marko') $\wedge \neg$(godina_studija < 3)
$\equiv$ (ime = 'Ivan') $\wedge$ (ime $\neq$ 'Marko') $\wedge$ (godina_studija $\geq$ 3)

Minterm $m_1$ možemo zapisati i ovako:

$m_2$: $p_1 \wedge \neg p_3 \equiv$ (ime = 'Ivan') $\wedge \neg$(godina_studija < 3) $\equiv$
$\equiv$ (ime = 'Ivan') $\wedge$ (godina_studija $\geq$ 3)

Ne moramo zapisati $p_2$ jer $p_1$ implicira $\neg p_2$. Nekad možemo imati veliki broj jednostavnih predikata koji se odnose na stupac u tablici koji ima vrijednosti koje nisu usporedive osim u smislu jednakosti. U takvim situacijama ćemo pisati samo jedan jednostavan predikat tog tipa.

za $Pr = \{p_3, p_4, p_5\}$:
$m_3$: $p_3 \wedge p_4 \wedge \neg p_5 \equiv$
$\equiv$ (godina_studija < 3) $\wedge$ (godina_studija $\geq$ 2) $\wedge \neg$(jmbag > '1000050000')
$\equiv$ (godina_studija < 3) $\wedge$ (godina_studija $\geq$ 2) $\wedge$ (jmbag $\leq$ '1000050000')

Možemo vidjeti da su, poznavajući semantičko značenje stupaca, neki jednostavni predikati suvišni u neki minterm predikatima. Naime, kod $m_2$ imamo (ime = 'Ivan') i (ime $\neq$ 'Marko'). Očito ako je ime = 'Ivan', onda ne može biti ime = 'Marko'. Zapravo su $m_1$ i $m_2$ logički ekvivalentni. Također znamo da je $m_3$ ekvivalentan tome da je godina_studija = 2, jer znamo semantičko značenje retka godina studija.

Primijetimo:
za $Pr = \{p_1, p_2, p_3, p_4, p_5\}$:
$m_4$: (godina_studija = 1) $\wedge$ $\neg$(jmbag > '1000050000') $\equiv$
$\equiv$ (godina_studija = 1) $\wedge$ (jmbag $\leq$ '1000050000')

nije minterm predikat. Naime, nemamo jednostavan predikat koji odgovara (godina_studija = 1) ili $\neg$(godina_studija = 1). Očito, lako se doda jednostavan predikat da $m_4$ bude minterm predikat, ali čak i da dodamo taj jednostavan predikat, $m_4$ i dalje ne bi sadržavao sve predikate skupa *Pr*, pa i dalje ne bi bio minterm predikat nad skupom *Pr*.
◇

Sada definiramo svojstva mintermova pomoću kojih ćemo graditi i analizirati fragmentaciju. Svojstva gledamo u kontekstu da postoji neka korisnička aplikacija koja pokreće upite i transakcije.

***Definicija 3.3*** Minterm selektivnost i frekvencija pristupa

1) **Minterm selektivnost** (minterm selectivity): udio redaka tablice kojima će minterm pristupiti. Minterm selektivnost ovisi o trenutnim podatcima u tablici.
   Pišemo $sel(m_i)$ za selektivnost minterma $m_i$.
2) **Frekvencija pristupa** (access frequency): frekvencija s kojom aplikacija pristupa podatcima.
   Pišemo $acc(q_i)$ za frekvenciju pristupa upita $q_i$.
   Pišemo $acc(m_i)$ za frekvenciju pristupa minterma $m_i$, koja ovisi o frekvenciji pristupa upita koji odgovaraju mintermu $m_i$.
   U nastavku će $acc(m_i)$ biti apstraktna varijabla jer bez konkretne aplikacije ne možemo znati njezinu vrijednost.

□

*Primjer 3.4*
Neka tablica STUDENT (jmbag, ime, prezime, godina_studija), ima sljedeći sadržaj:

| | jmbag | ime | prezime | godina_studija |
|---|---|---|---|---|
| 1 | 1191200304 | Jan | Janic | 1 |
| 2 | 1191200315 | Klara | Klaric | 2 |
| 3 | 1191200320 | Danica | Noc | 3 |
| 4 | 1191200325 | Borna | Bornic | 4 |
| 5 | 1191200331 | Mia | Miic | 5 |

**Slika 3.1** Sadržaj tablice STUDENT

Tada, za upit u jeziku SQL
*q:* `SELECT * from STUDENT where godina_studija < 4;`
vrijedi:
*sel*(*q*) = 0.6 jer će *q* pristupiti ⅗ redaka.
*acc*(*q*) ne znamo jer to ovisi o aplikaciji.
◇

### 3.1.1 Primarna horizontalna fragmentacija

Kod svake fragmentacije pa tako i kod primarne horizontalne fragmentacije želimo da fragmentacija bude potpuna. To svojstvo možemo osigurati na minterm predikatima pomoću kojih radimo fragmentaciju pa će potpunost same fragmentacije biti naslijeđena od mintermova.

Također želimo da skup jednostavnih predikata, preko kojih se definiraju minterm predikati, preko kojih se definiraju fragmenti, bude minimalan. Ne želimo fragmentirati bazu na fragmente kojima će se pristupati kao da su jedan fragment od strane korisničke aplikacije. Takva fragmentacija bi bila suvišna i proizvela bi nepotreban trošak vremena i resursa na dodatnu, odnosno, nepotrebnu komunikaciju. Za to uvodimo pojam relevantnosti.

***Definicija 3.4*** Relevantan predikat

Neka je *m* neki minterm predikat i *p* neki jednostavan predikat.
Neka $m_i = m \wedge p$ i $m_j = m \wedge \neg p$ minterm predikati.
Neka su $f_i$ *i* $f_j$ dva fragmenta definirani, redom, minterm predikatima $m_i$ *i* $m_j$
Tada je *p* **relevantan** ako vrijedi:

$$\frac{acc(m_i)}{card(f_i)} \neq \frac{acc(m_j)}{card(f_j)}$$

□

U definiciji relevantnosti zapravo imamo tablicu *f* koja je ili ne mora biti fragment. Predikat *p*, na temelju kojeg fragmentiramo *f* na $f_i$ *i* $f_j$, će biti relevantan ako postoje aplikacije koje će na drugi način pristupati bazi podataka u slučaju dodatne fragmentacije na temelju predikata *p*.

*Primjer 3.5*
Neka imamo tablicu STUDENT (jmbag, ime, prezime, godina_studija).
Neka imamo aplikaciju koja pristupa samo studentima preddiplomskog studija i aplikaciju koja pristupa samo studentima diplomskog studija.
Tada imamo $Pr_{\text{STUDENT}} = \{p_1: \text{godina\_studija} \leq 3\}$.

Definiramo fragmente $\text{STUDENT}_i$ tako da na tablicu STUDENT primijenimo minterme $m_1 = p_1$ i $m_2 = \neg p_1$.
Skup $Pr_{\text{STUDENT}}$ je potpun jer unijom $\text{STUDENT}_1$ i $\text{STUDENT}_2$ dobivamo cijelu tablicu STUDENT. On je i minimalan, jer imamo aplikacije koje pristupaju svaka svojem fragmentu.

Za $Pr_{\text{STUDENT}} = \{p_1$: godina_studija $\leq 3$, $p_2$: ime = 'Ivan'$\}$ nemamo aplikaciju čiji pristup bazi mijenja dodavanje predikata $p_2$, pa ovakav $Pr_{\text{STUDENT}}$ nije minimalan jer $p_2$ nije relevantan.
◇

Sada navodimo algoritam koji reducira skup jednostavnih predikata *Pr* na potpuni minimalni skup jednostavnih predikata *Pr*'. Algoritam radi u kontekstu relacije *R*.

**Ulaz**: relacija *R;* skup jednostavnih predikata *Pr*
**Izlaz**: *Pr*' potpuni minimalni skup jednostavnih predikata
**Varijable**: *F* skup fragmenata, *Pr*' = ∅
**START**
uzmi $p \in Pr$ takav da $p$ particionira $R$ tako da svakom fragmentu pristupa posebno bar jedna aplikacija;
$Pr' = Pr' \cup \{p\}$;
$Pr = Pr \setminus \{p\}$;
$F = F \cup \{f\}$; #*f* je fragment kojeg generira *p*
**while**(*Pr*' nije potpun)
    uzmi $p \in Pr$ takav da $p$ particionira neki fragment *f',* definiran nekim minterm predikatom definiranim nad skupom *Pr*', tako da svakom fragmentu pristupa posebno bar jedna aplikacija;
    $Pr' = Pr' \cup \{p\}$;
    $Pr = Pr \setminus \{p\}$;
    $F = F \cup \{f\}$; #*f* je fragment kojeg generira *p*
    **if** ($\exists p \in Pr'$ koji nije relevantan)
        $Pr' = Pr' \setminus \{p\}$;
        $F = F \setminus \{f\}$; #*f* je fragment kojeg generira *p;*
**END**

Algoritam uzima jedan predikat koji je relevantan, a zatim iterativno dodaje elemente u *Pr*' pazeći na minimalnost u svakom koraku.

Sada želimo neki smisleni skup *M* minterm predikata za našu fragmentaciju. Kardinalnost skupa svih minterm predikata nad skupom *Pr*' je jednaka kardinalnosti partitivnog skupa skupa *Pr*', to jest eksponencijalan je. Očito onda imamo velik broj suvišnih minterm predikata. Prvi suvišni minterm predikati koji nam padaju na pamet su kontradiktorni predikati, to jest

antitautologije. Minterm predikat koji je antitautologija neće fragmentirati relaciju jer nijedan redak neće zadovoljavati uvjete minterm predikata. Osim antitautologija, ne možemo odbaciti druge minterm predikate. Možemo imati neke minterm predikate koji će nam napraviti fragmente koji će biti prazni, ali ih ne možemo odbaciti jer nisu prazni zbog semantike relacije i njezinih atributa, već zbog sadržaja baze podataka koji je promjenjiv.

*Primjer 3.6*
Za $Pr' = \{p_1, p_2\}$ imamo:
$p_1$: jmbag = '1234567890'
$p_2$: jmbag = '0987654321'
$m_1$: $p_1 \wedge p_2 \equiv$ (jmbag = '1234567890') $\wedge$ (jmbag = '0987654321')
$m_2$: $p_1 \wedge \neg p_2 \equiv$ (jmbag = '1234567890') $\wedge \neg$(jmbag = '0987654321')
$m_3$: $\neg p_1 \wedge p_2 \equiv \neg$(jmbag = '1234567890') $\wedge$ (jmbag = '0987654321')
$m_4$: $\neg p_1 \wedge \neg p_2 \equiv \neg$(jmbag = '1234567890') $\wedge \neg$(jmbag = '0987654321')

Očito je $m_1$ antitautologija. Ne može jmbag studenta istovremeno biti i 1234567890 i 0987654321. Dakle $m_1$ možemo izbaciti iz skupa minterm predikata koje razmatramo za fragmentaciju.
◇

### 3.1.2 Izvedena horizontalna fragmentacija

Izvedena horizontalna fragmentacija je obično sljedeći korak nakon primarne fragmentacije. Nakon što smo relaciju $R$ fragmentirali na fragmente $F = \{F_1, F_2, \ldots F_n\}$ i pohranili ih na $n$ različitih čvorova, imamo relacije koje imaju strane ključeve koji se referenciraju na podatke u drugim čvorovima. Ovo se ne može riješiti jednostavnim tehnikama baza podataka već zahtjeva dodatnu softversku podršku, to jest okidače koji čuvaju integritet baze podataka umjesto tradicionalnih ograničenja ili izvedenu horizontalnu fragmentaciju.

*Primjer 3.7*
Recimo da smo tablicu STUDENT (jmbag, ime, prezime, godina_studija) fragmentirali na dva fragmenta $\text{STUDENT}_{\text{PREDDIPLOMSKI}}$ i $\text{STUDENT}_{\text{DIPLOMSKI}}$ kao u *primjeru 3.5*. Dakle, $\text{STUDENT}_{\text{PREDDIPLOMSKI}}$ sadrži informacije o studentima koji su trenutno upisani na prve tri godine fakulteta, a $\text{STUDENT}_{\text{DIPLOMSKI}}$ o studentima koji su trenutno upisani na zadnje dvije godine fakulteta. Ta dva fragmenta svaki stavimo na jedan čvor.

Pogledajmo sada tablicu UPISAO (jmbag, šifra, ocjena, godina_upisa) koja nije fragmentirana. Oba čvora sadrže cijelu tablicu UPISAO. Ovdje je jmbag strani ključ koji se referencira na tablicu STUDENT. Sada ako probamo unijeti informacije o studiju studenata koji su trenutno

upisani na diplomski studij u tablicu UPISAO, dobit ćemo grešku i ta transakcija se neće provesti. Naime, na jednom čvoru, koji sadrži informacije samo o studentima preddiplomskog studija, nema jmbagova studenata diplomskog studija, dok na drugom čvoru nema jmbagova studenata preddiplomskog studija. Ovo je problem jer trenutno na oba čvora trebamo pamtiti cijelu tablicu UPISAO.
Ovaj problem možemo riješiti na dva načina:

1) Provedemo izvedenu horizontalnu fragmentaciju nad tablicom UPISAO u skladu s već provedenom primarnom fragmentacijom nad tablicom STUDENT. Na taj način ćemo podatke o preddiplomskom studiju studenata pamtiti na istom čvoru gdje pamtimo i podatke o samim studentima.
2) Uklonimo direktno ograničenje stranog ključa i dodamo okidače koji će zajedno osigurati sve funkcionalnosti stranog ključa. Radi se o okidačima koji će, pri unosu u tablicu UPISAO, provjeravati postoji li u lokalnoj tablici STUDENT ili u udaljenoj tablici STUDENT redak takav da se njegov jmbag unosi kao stupac jmbag u tablicu UPISAO. Ovi okidači će također trebati zabraniti brisanje iz bilo koje tablice STUDENT ako jmbag tog studenta postoji u nekoj tablici UPISAO.

◇

Za relaciju $S$, koja je fragmentirana na fragmente $\{S_1, S_2, \dots, S_n\}$, definiramo fragmente $R_i$ relacije $R$ tako da redak $r$ od $R$ svrstavamo u $R_j$ ako je redak $s$ iz $S$, čija vrijednost je strani ključ retka $r$, svrstan u $S_j$. Ovdje koristimo operaciju poluspoj (semijoin), čiji je rezultat skup redaka iz lijeve tablice koji bi sudjelovali u prirodnom spoju (natural join) s desnom tablicom.
Pišemo $R_i = R \ltimes S_i$.

Ako relacija ima samo jedan strani ključ koji dolazi iz fragmentirane tablice, onda je izvedena fragmentacija trivijalna za provesti, kako je gore navedeno. Ako pak, s druge strane, relacija $R$, koju želimo fragmentirati, ima više stranih ključeva koji pripadaju različitim tablicama, koje su fragmentirane, onda moramo izabrati jednu od njih, na temelju koje ćemo provesti izvedenu fragmentaciju. Za taj izbor imamo dva kriterija. Oba kriterija se temelje na analizi primjene baze podataka i na analizi upita i transakcija koje će se provoditi nad bazom podataka. Možemo fragmentirati tablicu $R$ tako da što više aplikacija svaka imaju svoj fragment ili tako da spajanje tablica, u smislu spajanja, učinimo efikasnijim.

Prvi kriterij je vrlo intuitivan i direktan. Jednostavno pogledamo sve izvedene fragmentacije $F^k$, gdje je $k$ relacija na temelju koje se vrši izvedena fragmentacija te izaberemo fragmentaciju takvu da najviše aplikacija svaka ima svoj fragment.

Drugi kriterij nije tako lako za evaluirati. Provođenje spojeva ovisi o fragmentaciji na dva načina. Što manje fragmente imamo, to će se spajanje brže provesti na dva fragmenta i što više fragmenata imamo, to više podspajanja možemo provesti paralelno da bismo njihovu uniju predstavili kao rezultat spajanja dvije nefragmentirane tablice.

*Primjer 3.8*

Neka imamo dvije aplikacije: prva pristupa samo studentima preddiplomskog studija, a druga samo studentima diplomskog studija.

Pogledajmo sada dva slučaja fragmentacije tablice STUDENT:

1) Radimo fragmentaciju kao u *primjeru 3.5*.
2) Dodatno fragmentiramo preddiplomski fragment na studente prve te studente druge i treće godine. Dodatno fragmentiramo diplomski fragment na studente četvrte i pete godine.

Za oba slučaja ćemo provesti izvedenu fragmentaciju tablice UPISAO. Neka naša tablica STUDENT sadrži 32 redaka (studenta), te tablica UPISAO 276 redaka (upisa)

Pogledajmo prvi slučaj:

$STUDENT_{PREDDIPLOMSKI}$ sadrži 22 retka

$STUDENT_{DIPLOMSKI}$ sadrži 10 redaka

$UPISAO_{PREDDIPLOMSKI} = UPISAO \ltimes STUDENT_{PREDDIPLOMSKI}$

sadrži 120 redaka

$UPISAO_{DIPLOMSKI} = UPISAO \ltimes STUDENT_{DIPLOMSKI}$

sadrži 156 redaka

Ovdje imamo dvije aplikacije koje svaka ima svoj fragment, ali ako želimo spojiti neki podskup podataka iz neke od tih dviju podtablica tablice UPISAO s podskupom odgovarajuće podtablice STUDENT, moramo proći kroz dvije velike tablice. Recimo da jedan korisnik želi ispisati imena svih studenata prve godine, a drugi korisnik želi spojiti sve studente treće godine s njihovim upisima predmeta u tekućoj godini. Tada će prvi upit zaključati tablicu $STUDENT_{PREDDIPLOMSKI}$, a kako drugi upit također treba pristupit tablici $STUDENT_{PREDDIPLOMSKI}$, drugi upit treba čekati da prvi upit završi s tablicom $STUDENT_{PREDDIPLOMSKI}$. Oba upita će trajati dulje jer pretražuju i spajaju veće tablice.

Pogledajmo drugi slučaj:

$STUDENT_1$ sadrži 9 redaka

$STUDENT_{2,3}$ sadrži 13 redaka

$STUDENT_4$ sadrži 6 redaka

$STUDENT_5$ sadrži 4 redaka

$UPISAO_1 = UPISAO \ltimes STUDENT_1$

sadrži 18 redaka

$UPISAO_{2,3} = UPISAO \ltimes STUDENT_{2,3}$

sadrži 102 redaka

$UPISAO_4 = UPISAO \ltimes STUDENT_4$

sadrži 84 redaka

$UPISAO_5 = UPISAO \ltimes STUDENT_5$

sadrži 72 redaka

Ovdje svaka aplikacija ima dva fragmenta kojima pristupa. To malo komplicira implementaciju tih aplikacija te još trebamo više čvorova ili više baza podataka na istom broju čvorova, to jest više hardvera. Ali ako želimo spojiti neke podskupove tih podataka, to možemo puno efikasnije napraviti. Recimo da imamo upite kao u prvom slučaju: jedan upit ispisuje imena studenata prve godine, a drugi upit spaja studente treće godine s njihovim upisima predmeta tekuće godine. Tada prvi upit pristupa tablicama $\text{STUDENT}_1$ i $\text{UPISAO}_1$, a drugi upit pristupa tablicama $\text{UPISAO}_{2,3}$ i $\text{STUDENT}_{2,3}$. Sada su podatci kojima upiti pristupaju disjunktni, pa se upiti mogu izvršavati paralelno. Dodatno, tablice su manje, pa će se i sami upiti izvršiti brže.
◇

### 3.1.3 Korektnost horizontalne fragmentacije

**Potpunost** primarne horizontalne fragmentacije je garantirana ako uzmemo potpuni i minimalni skup predikata *Pr*. Ako je potpun, neće se dogoditi da postoji redak početne tablice koji nije u nijednom fragmentu. Potpunost derivirane horizontalne fragmentacije slijedi iz potpunosti primarne horizontalne fragmentacije i iz ograničenja stranih ključeva.

**Rekonstrukcija** slijedi iz potpunosti. Primjenom operatora unije imamo

$$R = \bigcup_{i=1}^{n} R_i$$

za fragmente $F_R = \{R_1, \ldots, R_n\}$ relacije $R$.

**Disjunktnost** primarne horizontalne fragmentacije je garantirana ako su minterm predikati, na temelju kojih je definirana fragmentacija, međusobno isključivi, to jest, nijedna dva minterm predikata nisu istiniti za isti redak. Disjunktnost izvedene horizontalne fragmentacije je garantirana ako postoji samo jedna veza između relacije koja je fragmentirana primarnom horizontalnom fragmentacijom i relacije koja je fragmentirana izvedenom horizontalnom fragmentacijom. Ako nemamo ovo svojstvo, onda se disjunktnost treba provjeravati redak po redak.

## 3.2 Vertikalna fragmentacija

Vertikalna fragmentacija je kompliciraniji proces od horizontalne fragmentacije jer je broj mogućih opcija puno veći. Naime, vertikalna fragmentacija je razdvajanje tablice u podtablice

tako da svaka podtablica sadrži neke od stupaca početne tablice zajedno s primarnim ključem, uzimajući u obzir svojstva potpunosti, rekonstrukcije i disjunktnosti. Zbog toga je broj mogućih fragmentacija tablice s $m$ stupaca jednak broju particija $m$-članog skupa, to jest, bellovom broju, $B(m)$. Bellov broj broja $m$ je približno jednak broju $m^m$ za velike vrijednosti. Takav veliki broj mogućnosti zahtjeva korištenje heuristika da bi se našlo zadovoljavajuće rješenje. Nude se dvije heuristike: grupiranje (grouping) i razdvajanje (splitting). Oba algoritma uzimaju u obzir aplikacije koje pristupaju bazi podataka i frekvencije kojima te aplikacije pristupaju određenim atributima. Na temelju frekvencija pristupanja atributima se definiraju fragmenti tako da rad aplikacija bude što efikasniji. Kriterij za fragmentaciju ne mora biti efikasnost, već može biti i sigurnost. Možemo povjerljive podatke, to jest, atribute koji ih sadrže, stavljati na čvorove s većim stupnjem sigurnosti.

Grupiranje definira svaki fragment kao jedan atribut zatim iterativno spaja fragmente dok ne zadovolji neke kriterije.

Razdvajanje uzima početnu tablicu te na temelju matrica definiranih pomoću kvantifikacije srodnosti (affinity) atributa razdvaja tablicu u podtablice tako da atributi kojima se često pristupa istovremeno, to jest jednim upitom, stavlja u isti fragment.

*Primjer 3.9*
Pogledajmo tablicu PREDAVAČ (<u>oib</u>, ime, prezime, plaća). Recimo da je plaća predavača povjerljiv podatak, pa ga želimo držati na čvoru s većom sigurnošću.
Tada dobivamo dvije tablice:
PREDAVAČ$_1$ (<u>oib</u>, ime, prezime)
PREDAVAČ$_2$ (<u>oib</u>, plaća)
◇

Korektnost vertikalne fragmentacije će biti zadovoljena ako stupci fragmenata, osim primarnog ključa, čine particiju stupaca početne relacije. Svaki korak u procesu grupiranja i razdvajanja će paziti na svojstva particije, pa će ih finalni fragmenti također zadovoljavati. Svaki element početnog skupa se nalazi u nekom od elemenata particije, pa svaki fragment sadrži neki stupac iz početne relacije, pa je potpunost zadovoljena. Elementi particije su disjunktni, pa će i fragmenti biti međusobno disjunktni, osim primarnog ključa, pa će disjunktnost biti zadovoljena. Rekonstrukcija se postiže primjenom operatora spoja s uvjetom jednakosti primarnog ključa na sve fragmente.

## 3.3 Hibridna fragmentacija (mixed/nested fragmentation)

Ako samo jedna horizontalna ili vertikalna fragmentacija nije bila dovoljna za neku relaciju, onda fragmentiramo dalje. Dvije uzastopne vertikalne fragmentacije ne smatramo odvojenima jer ta dva koraka fragmentiranja možemo promatrati kao jedan korak. Isto tako za

horizontalnu fragmentaciju. Zbog toga hibridnu fragmentaciju prikazujemo kao stablo, gdje svaka druga razina stabla odgovara istoj vrsti fragmentacije. Razlozi za hibridnu fragmentaciju su srodne aplikacije koje pristupaju sličnim podatcima na temelju jednog atributa, ali različitima na temelju drugog atributa, čija je vrijednost obično ovisna o prvom atributu. Hibridnom fragmentacijom povećavamo efikasnost takvih aplikacija. Korektnost hibridne fragmentacije slijedi iz korektnosti horizontalnih i vertikalnih fragmentacija od kojih se ona sastoji.

# 4. Obrada distribuiranih upita (distributed query processing)

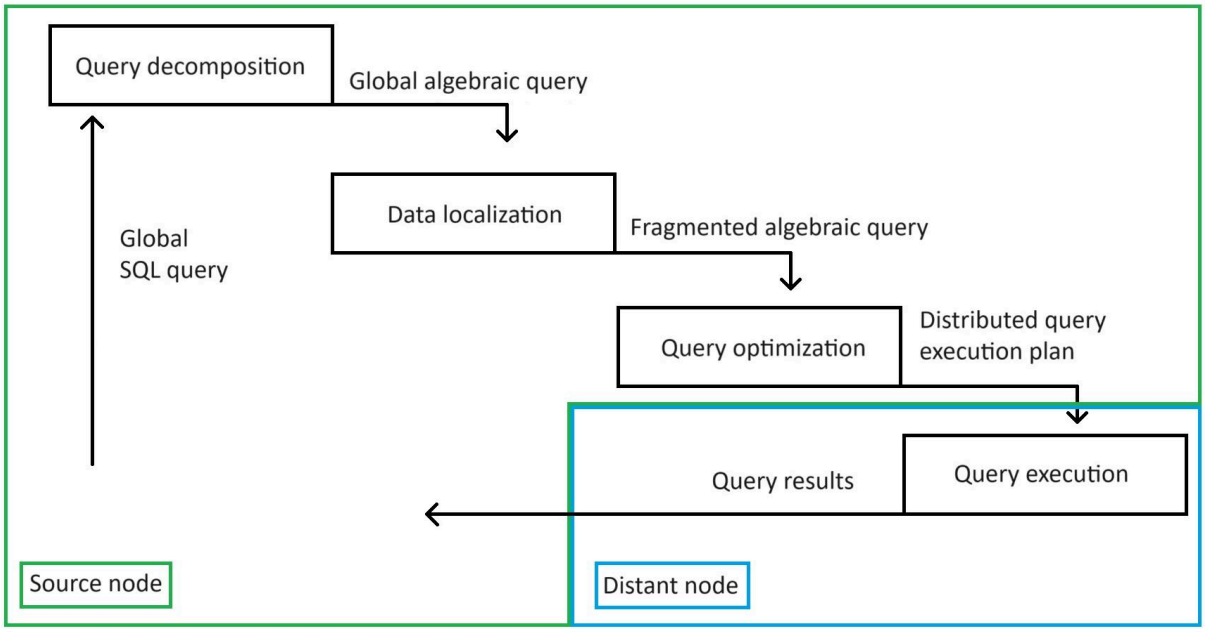
Kada korisnik ili programer pokrene upit ili transakciju na svom DBMS-u, obično u obliku SQL koda, on ne mora znati je li baza distribuirana ili ne. Korisnik ili programer ne treba znati koje čvorove treba kontaktirati da bi dobio ili ažurirao podatke koje želi dobiti ili ažurirati. I još najbitnije: korisnik ne treba znati koji je najbrži način za dohvatiti neke podatke. O svemu tome se brine DBMS i sama baza podataka, koji analiziraju svaku transakciju ili upit i rastavljaju ih na manje segmente kako bi ih kasnije ponovno sastavili tako da provedba bude što efikasnija. Obrada distribuiranog upita ili transakcije se dijeli na četiri koraka. Prva tri koraka se odvijaju na čvoru na kojem je pokrenut upit. Takav čvor zovemo majstor čvor (master site). Četvrti korak se odvija na čvorovima koji su potrebni za provođenje upita. Takve čvorove zovemo šegrt čvorovi (apprentice sites).

Prvi korak prevodi upit u formulu logike sudova, koju kasnije evaluira i transformira u jednostavniju, ali ekvivalentnu formulu. Transformirana formula se prevodi u izraz relacijske algebre i predaje drugom koraku. Prvi korak obrade ne zna za distribuciju podataka, te je analogan koraku centralizirane obrade upita.

Drugi korak prima upit relacijske algebre, te ga pretvara u distribuirani upit. U ovom koraku se uspoređuje dobiveni upit sa shemom distribuirane baze podataka, to jest s načinom fragmentacije i replikacije. Za takvo prevođenje se koristi stablo upita (query tree). Nakon ovog koraka imamo distribuirani upit koji uzima u obzir način fragmentacije i replikacije podataka.

Treći korak optimizira upit tako da redoslijed kojim pristupamo fragmentima i redoslijed kojim spajamo podatke bude što efikasniji. U većini slučajeva se radi o prevelikom broju kombinacija da bismo dobili točno optimalno rješenje, pa se korištenjem heuristika dolazi do zadovoljavajućeg rješenja koje je dovoljno blizu optimalnom. Nakon ovog koraka imamo skup podupita ili podtransakcija koje se šalju čvorovima koji ih trebaju provesti.

U zadnjem, četvrtom, koraku svaki čvor, koji je dobio podupit, ga ponovno analizira i optimizira u odnosu na lokalnu shemu, kao u centraliziranoj bazi podataka. Nakon još jedne optimizacije, čvor provodi lokalni upit ili lokalnu transakciju i rezultate šalje čvoru od kojeg je dobio podupit.




**Slika 4.1** Ilustracija koraka obrade distribuiranih upita.

## 4.1 Rastav upita (query decomposition)

Rastav upita se dijeli na četiri podkoraka. Prvo prevodimo SQL kod u formulu logike sudova i normaliziramo ju. U drugom podkoraku analiziramo dobivenu formulu te ju, u slučaju neispravnosti, odbacujemo. U trećem podkoraku uklanjamo redundancije u formuli logike sudova te ju u četvrtom podkoraku zapisujemo u obliku relacijske algebre. Između svaka dva podkoraka se čuva ekvivalentnost značenja formule.

U prvom podkoraku gledamo kvalifikaciju upita, to jest, dio SQL koda nakon ključne riječi WHERE. Dio nakon ključne riječi WHERE je zapravo skup predikata povezanih logičkim veznicima i (AND) i ili (OR). Na trivijalan način možemo taj dio SQL koda prevesti u formulu logike sudova. Dobivenu formulu logike sudova zatim normaliziramo, to jest, zapisujemo ju u obliku KNF (konjunktivna normalna forma) ili DNF (disjunktivna normalna forma). Obično želimo KNF jer upiti obično sadrže više AND nego OR veznika.

1. $p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$
2. $p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
3. $p_1 \wedge (p_2 \wedge p_3) \Leftrightarrow (p_1 \wedge p_2) \wedge p_3$
4. $p_1 \vee (p_2 \vee p_3) \Leftrightarrow (p_1 \vee p_2) \vee p_3$
5. $p_1 \wedge (p_2 \vee p_3) \Leftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$
6. $p_1 \vee (p_2 \wedge p_3) \Leftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
7. $\neg(p_1 \wedge p_2) \Leftrightarrow \neg p_1 \vee \neg p_2$
8. $\neg(p_1 \vee p_2) \Leftrightarrow \neg p_1 \wedge \neg p_2$
9. $\neg(\neg p) \Leftrightarrow p$

**Slika 4.2** Logičke ekvivalencije.

*Primjer 4.1*
Recimo da želimo rastaviti sljedeći upit, koji traži sve predavače koji predaju predmete zimskog semestra na diplomskom studiju :

```
SELECT distinct prezime
from PREDAVAČ, PREDAJE, PREDMET
WHERE PREDAVAČ.oib = PREDAJE.oib
AND PREDAJE.sifra = PREDMET.sifra
AND (semestar = 7 OR semestar = 9);
```

Tada je KNF:
predavač.oib = predaje.oib $\wedge$
predaje.šifra = predmet.šifra $\wedge$
(semestar = 7 $\vee$ semestar = 9)

Korištenjem pravila 5 sa **slike 4.1** dobivamo DNF:
(predavač.oib = predaje.oib $\wedge$ predaje.šifra = predmet.šifra $\wedge$ semestar = 7)
$\vee$
(predavač.oib = predaje.oib $\wedge$ predaje.šifra = predmet.šifra $\wedge$ semestar = 9)

Primijetimo da se ist predikati pojavljuju u obje elementarne konjunkcije u DNF.
$\diamond$

U drugom podkoraku želimo odbaciti sve upite koji nemaju smisla. Tražimo upite koji imaju pogrešne tipove ili semantičke greške. Takve upite nema smisla provoditi, pa niti dalje obrađivati. Upit ima pogrešne tipove ako se objekti, koje upit spominje, ne nalaze u globalnoj nedistribuiranoj shemi baze podataka ili ako se operatori pokušavaju primijeniti na tipove na koje se ne mogu primijeniti.

*Primjer 4.2*
Pogledajmo sljedeći upit :

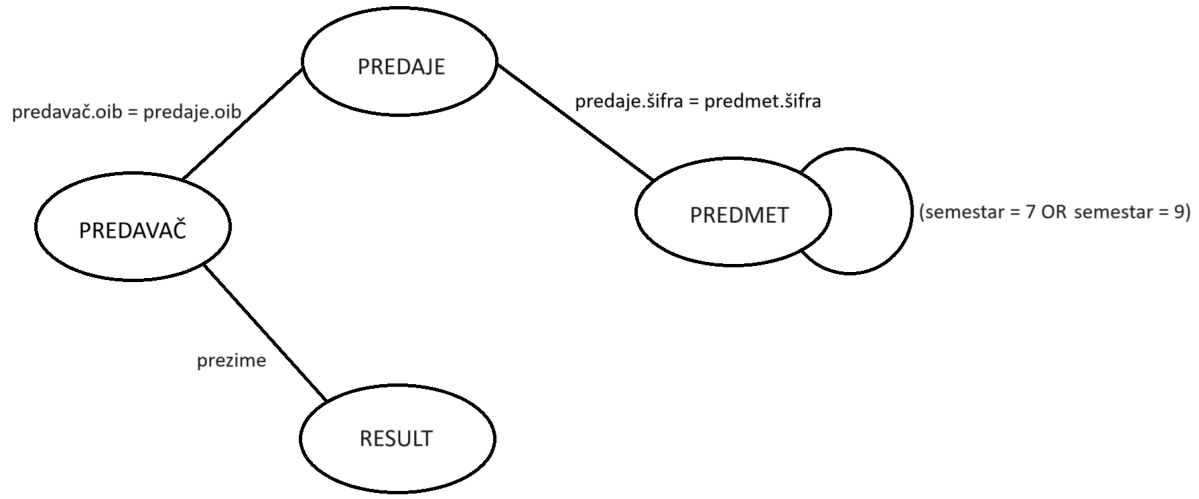
```
SELECT adresa
from PREDAVAČ
WHERE ime > 123
```

U tablici PREDAVAČ (oib, ime, prezime, plaća) ne postoji stupac adresa. Dodatno, ime je tipa varchar(20), koji nije usporediv s 123, koji je tipa int, cijeli broj, no neki DBMS-ovi bi proveli konverziju tipova i napravili usporedbu.
$\diamond$

Upit je semantički pogrešan ako ima dijelove koji ne doprinose rezultatu. Takve greške pronalazimo tako da upit zapišemo u obliku grafa koji zovemo graf upita (query graph). U grafu upita jedan čvor predstavlja rezultat upita, a svi drugi čvorovi predstavljaju operande (tablice) koje povezujemo lukovima. Lukovi predstavljaju operacije. Lukovi između operanada predstavljaju spajanja (joins), a lukovi između rezultata i operanada predstavljaju projekcije. Luk koji povezuje čvor sam sa sobom predstavlja selekciju te tvori podtablicu definiranu nekim uvjetom na stupce te tablice.

*Primjer 4.3*
Pogledajmo upit u *primjeru 4.1*:

```
SELECT distinct prezime
from PREDAVAČ, PREDAJE, PREDMET
WHERE PREDAVAČ.oib = PREDAJE.oib
AND PREDAJE.šifra = PREDMET.šifra
AND (semestar = 7 OR semestar = 9);
```



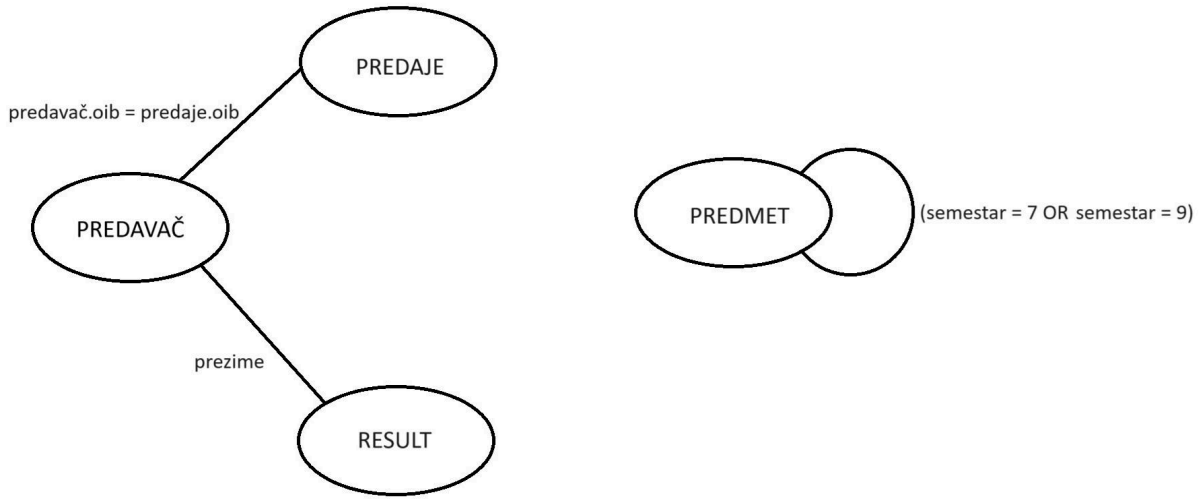

**Slika 4.3** Graf upita a).

Pripadni graf upita je **Slika 4.3**. Vidimo da je graf upita povezan, pa ne nalazimo semantičke greške.
◇

*Primjer 4.4*
Pogledajmo sljedeći upit :

```
SELECT distinct prezime
from PREDAVAČ, PREDAJE, PREDMET
```

```
WHERE PREDAVAC.oib = PREDAJE.oib
AND (semestar = 7 OR semestar = 9);
```




**Slika 4.4** Graf upita b).

Pripadni graf upita je **Slika 4.4**. Vidimo da graf upita nije povezan, dakle upit je semantički neispravan. Naime, biraju se redci tablice PREDMET koji imaju stupac semestar s vrijednosti 7 ili 9, ali ti redci, pa ni tablica PREDMET ne doprinose rezultatu. Umjesto odbacivanja upita možemo pretpostaviti da postoji Kartezijev produkt između tablica PREDMET i PREDAJE ili dodati upitu redak koji ga pretvara u upit iz *primjera 4.3*. Gledamo baš tablice PREDMET i PREDAJE, a ne tablice PREDAVAC i PREDMET, jer, po globalnoj shemi baze podataka, znamo da PREDAJE sadrži strani ključ koji referencira tablicu PREDMET.

◇

U trećem podkoraku uklanjamo redundancije iz logičke formule. Cilj ovog koraka je smanjiti broj pretraživanja uklanjanjem uvjeta koji ne doprinose konačnom rezultatu. Svaki rad koji se provodi košta vrijeme, pa želimo raditi minimalan potreban rad i želimo izbjeći ponavljanje već odrađenog rada.

1. $p \wedge p \Leftrightarrow p$
2. $p \vee p \Leftrightarrow p$
3. $p \wedge true \Leftrightarrow p$
4. $p \vee false \Leftrightarrow p$
5. $p \wedge false \Leftrightarrow false$
6. $p \vee true \Leftrightarrow true$
7. $p \wedge \neg p \Leftrightarrow false$
8. $p \vee \neg p \Leftrightarrow true$
9. $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$
10. $p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$

**Slika 4.5** Logičke ekvivalencije.

*Primjer 4.5*
Pogledajmo sljedeći upit :

```
SELECT prezime
from PREDAVAČ
WHERE
(NOT ime = 'August'
AND ime = 'Klaudije'
OR placa > 950
AND NOT placa > 950)
OR ime = 'Klaudije';
```

Neka je
$p_1$ : `ime = 'August'`
$p_2$ : `ime = 'Klaudije'`
$p_3$ : `placa > 950`

Tada je kvalifikacija upita

$((\neg p_1 \wedge p_2) \vee (p_3 \wedge \neg p_3)) \vee p_2$

korištenjem pravila 7 sa **slike 4.5** dobivamo

$((\neg p_1 \wedge p_2) \vee \mathit{false}) \vee p_2$

korištenjem pravila 4 sa **slike 4.5** dobivamo

$(\neg p_1 \wedge p_2) \vee p_2$

korištenjem pravila 10 sa **slike 4.5** dobivamo

$p_2$

Dakle, gore navedeni upit je ekvivalentan upitu

```
SELECT prezime
from PREDAVAČ
WHERE ime = 'Klaudije';
```
◇

U zadnjem, četvrtom, podkoraku rastava upita zapisujemo upit u relacijskoj algebri. Upit u relacijskoj algebri ćemo prikazivati u obliku grafa koji zovemo stablo operatora (operator tree). U stablu operatora je svaki list tablica u bazi podataka, korijen stabla je rezultat, te nekorijen i nelist čvorovi su relacije nastale primjenom operatora relacijske algebre. Za neki upit postoji velik broj stabala operatora koji su međusobno ekvivalentni u smislu da daju isti rezultat, iako bismo htjeli optimalno stablo u smislu broja operacija računala, u praksi ćemo heuristikom pronaći neko stablo koje je dovoljno dobro. Za optimizaciju stabla koristimo sljedeća transformacijska pravila (transformation rules):

1) **Komutativnost binarnih operatora:** Kartezijev produkt dviju relacija i spoj dviju relacija su komutativni.
   $R \times S = S \times R$
   $R \bowtie S = S \bowtie R$

2) **Asocijativnost binarnih operatora:** Kartezijev produkt i spoj su asocijativni.
   $(R \times S) \times T = R \times (S \times T)$
   $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

3) **Idempotencija unarnih operatora:**
   a) Niz projekcija nad istom relacijom ima isti rezultat kao zadnja primijenjena projekcija.
      $A' \subseteq A$ za $A', A \subseteq ATT$, gdje je *ATT* skup atributa relacije *R*.
      $\Pi_{A'}(\Pi_A(R)) = \Pi_{A'}(R)$
   b) Selekcija s konjunkcijom predikata se može prikazati kao niz selekcija, svaka s jednim predikatom iz konjunkcije.
      $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) = \sigma_{p_1(A_1) \wedge p_2(A_2)}(R)$

4) **Komutacija selekcije i projekcije:** Projekcija i selekcija komutiraju na sljedeći način:
   $\Pi_{A_1,\ldots,A_n}(\sigma_{p(A_p)}(R)) = \Pi_{A_1,\ldots,A_n}(\sigma_{p(A_p)}(\Pi_{A_1,\ldots,A_n,A_p}((R))))$

5) **Komutacija selekcije i binarnih operatora:**
   a) Selekcija i Kartezijev produkt komutiraju. (Atribut *A* pripada relaciji *R*)
      $\sigma_{p(A)}(R \times S) = (\sigma_{p(A)}(R)) \times S$
   b) Selekcija i spoj komutiraju. (Atribut *A* pripada relaciji *R*)
      $\sigma_{p(A)}(R \bowtie S) = (\sigma_{p(A)}(R)) \bowtie S$
   c) Selekcija i unija komutiraju. (Relacije *R* i *S* imaju istu shemu)
      $\sigma_{p(A)}(R \cup S) = (\sigma_{p(A)}(R)) \cup (\sigma_{p(A)}(S))$
   d) Selekcija i skupovna razlika komutiraju. (Atribut *A* pripada relaciji *R*)
      $\sigma_{p(A)}(R \setminus S) = (\sigma_{p(A)}(R)) \setminus S$

6) **Komutacija projekcije i binarnih operatora:**
   a) Kartezijev produkt i projekcija komutiraju. Za $C = A' \cup B'$, gdje $A' \subseteq A$ i $B' \subseteq B$, gdje su $A$ i $B$ skupovi atributa relacija $R$ i $S$ redom.

$$\Pi_C(R \times S) = \Pi_{A'}(R) \times \Pi_{B'}(S)$$

   b) Spoj i projekcija komutiraju. Za $C = A' \cup B'$, gdje $A' \subseteq A$ i $B' \subseteq B$, gdje su $A$ i $B$ skupovi atributa relacija $R$ i $S$ redom, te $C$ sadrži uvjete spoja.

$$\Pi_C(R \bowtie S) = \Pi_{A'}(R) \bowtie \Pi_{B'}(S)$$

   c) Unija i projekcija komutiraju.

$$\Pi_C(R \cup S) = \Pi_C(R) \cup \Pi_C(S)$$

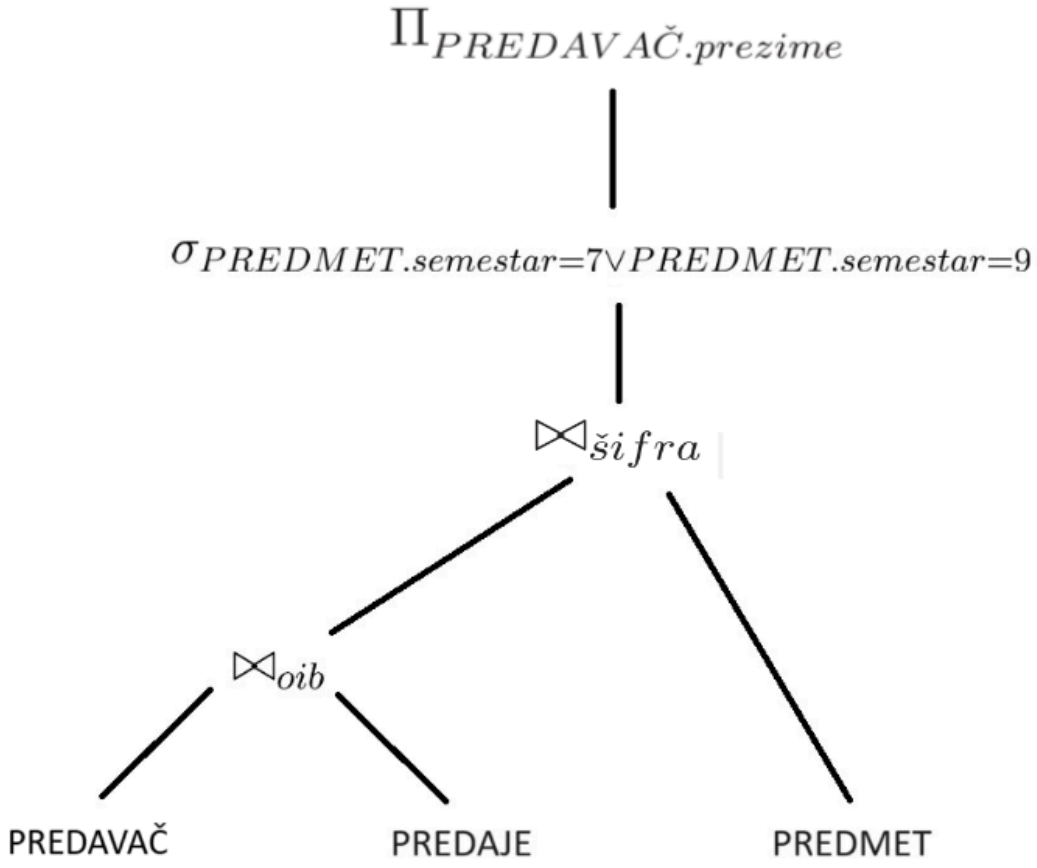
   d) Skupovna razlika i projekcija komutiraju.

$$\Pi_C(R \setminus S) = \Pi_C(R) \setminus \Pi_C(S)$$

*Primjer 4.6*
Pogledajmo upit iz *primjera 4.1*:

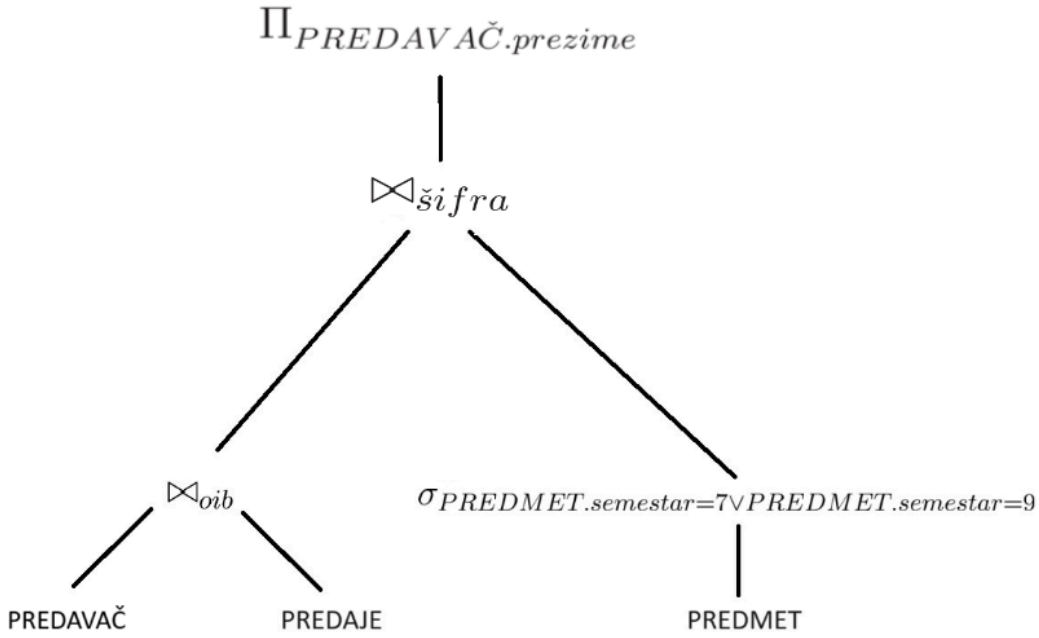
```
SELECT distinct prezime
from PREDAVAČ, PREDAJE, PREDMET
WHERE PREDAVAČ.oib = PREDAJE.oib
AND PREDAJE.šifra = PREDMET.šifra
AND (semestar = 7 OR semestar = 9);
```

Jedno stablo operatora za gornji upit je




**Slika 4.6** Stablo operatora a).

Na stablu na **slici 4.6** prvo spajamo tablice koje sadrže podatke koji nas zanimaju u jednu tablicu i nakon toga iz dobivene velike tablice selektiramo retke koji nas zanimaju te iz tih redaka projiciramo željeni stupac u konačni rezultat. Ovo stablo operatora je suboptimalno jer spaja velike tablice i onda selekciju vrši na još većoj tablici. Takav redoslijed operacija povećava ukupan posao koji procesor mora obaviti.




**Slika 4.7** Stablo operatora b).

Primjenom komutacije selekcije i spoja na stablo operatora sa **slike 4.6** dobivamo stablo operatora na **slici 4.7**. Drugo stablo operatora je bolje od prvog jer vrši selekciju na manjoj tablici i nakon toga vrši spajanje na manjoj tablici. Zbog toga je izvršavanje selekcije i spoja brže u drugom slučaju nego u prvom.

Općenito želimo u stablu operatora prvo primijeniti unarne operatore, a nakon njih binarne na što manjim operandima. Stablo na **slici 4.7** možemo još optimizirati tako da projiciramo podatke koji nas zanimaju iz tablica prije nego ih spajamo. Iz tablice PREDAVAC trebamo samo oib i prezime predavača, te iz tablice PREDMET trebamo, nakon selekcije, samo šifru predmeta.
◇

## 4.2 Lokalizacija podataka (data localization)

U prijašnjem koraku nismo koristili informacije o distribuciji podataka. Te informacije će nam biti potrebne u ovom koraku. Cilj drugog koraka je uzeti u obzir distribuciju podataka i pretvoriti naš upit koji je centraliziranog tipa u distribuirani upit čiji će se dijelovi provoditi na

različitim čvorovima. Prvo naivno rješenje bi bilo zamijeniti svaku fragmentiranu relaciju s unijom njenih fragmenata. Takav upit ćemo zvati lokalizirani upit. Očito lokalizirani upit nije općenito optimalan. Može se dogoditi da neki fragment ne sadrži niti jedan podatak koji pokušavamo dohvatiti, a to možemo znati na temelju pravila fragmentacije koja su dostupna DBMS-u. Komunikaciju s takvim fragmentima želimo izbjeći pri provođenju upita. Slijede pravila kojima reduciramo lokalizirane upite tako da uklonimo komunikaciju s čvorovima koji ne sadrže podatke koje tražimo.

**1)** $\sigma_{p_i}(R_j) = \varnothing$ ako $\forall x \in R$ vrijedi $\neg(p_i(x) \wedge p_j(x))$ gdje je $p_j$ predikat koji definira fragment $R_j$.
To jest, selekcija neće vratiti niti jedan redak tablice $R_j$ ako za svaki redak tablice $R$ vrijedi da on ne zadovoljava predikat selekcije ili se on ne nalazi u fragmentu $R_j$.

**2)** $R_i \bowtie R_j = \varnothing$ ako je $(p_i(x) \wedge p_j(x))$ antitautologija, gdje su $R_i$ i $R_j$ horizontalni fragmenti relacije $R$ definirani predikatima $p_i$ i $p_j$ nad istim stupcem.
To jest, spoj dviju fragmenata je prazan ako su predikati koji ih definiraju kontradiktorni.

**3)** $\Pi_A(R_i) = \varnothing$ ako vertikalni fragment $R_i$ ne sadrži atribut $A$.

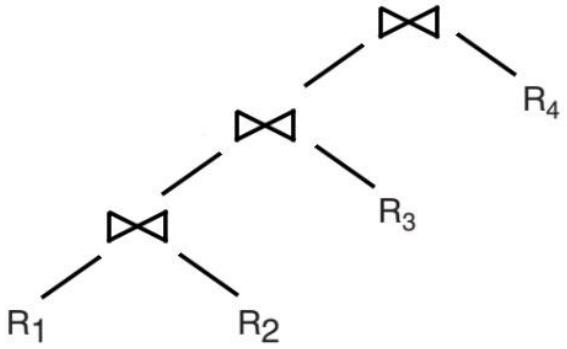
Ova pravila možemo provjeriti bez komunikacije s udaljenim čvorovima. Sve što nam je potrebno za provjeru ovih pravila i provođenje koraka lokalizacije podataka su pravila fragmentacije, koja su dostupna lokalno svakom čvoru.

## 4.3 Optimizacija upita (query optimization)

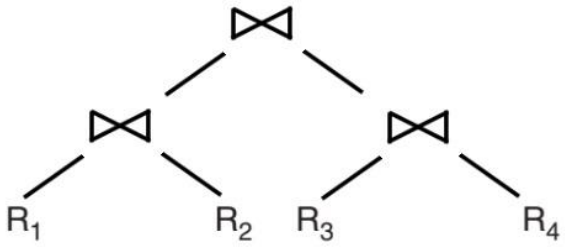
Nakon prethodnog koraka imamo upit u obliku formule relacijske algebre koji pristupa udaljenim čvorovima. Zadatak ovog koraka obrade upita je da nađemo ekvivalentno stablo operatora onome koje smo dobili iz prethodnog koraka, ali takvo da je njegova cijena što manja. Kao što je slučaj s većinom drugih problema optimizacije u ovom radu, tako i ovaj ima jako velik broj slučajeva. Provjera svih slučajeva košta previše, pa se opet zadovoljavamo dovoljno dobrim rezultatom nađenim pomoću heuristike. U problemu optimizacije upita proučavamo tri objekta: prostor pretraživanja (search space), model cijene (cost model) i strategiju pretraživanja (search strategy).

Prostor pretraživanja je skup svih stabala operatora koji su ekvivalentni onome koje smo dobili iz prethodnog upita, to jest, skup svih mogućih stabala operatora koji daju isti rezultat kao ono koje predstavlja početan upit. Možemo se fokusirati samo na stabla spajanja (join trees). Stabla spajanja su stabla operatora koja sadrže samo operatore spoja i Kartezijevog produkta. Nekad proučavamo takva stabla, jer je doprinos cijeni upita ostalih operatora često zanemariv u odnosu na doprinos operatora spoja i Kartezijevog produkta. Da bismo smanjili prostor

pretraživanja, obično se ograničavamo na stabla operatora takva da projekcije i selekcije primjenjujemo samo na baznim tablicama, to jest, na tablicama koje se nalaze u samoj bazi podataka, ne na privremenim tablicama koje su nastale kao spoj ili selekcija nekih drugih tablica. Ovakav pristup se podudara s heuristikom po kojoj je bolje prvo primijeniti unarne operatore te nakon njih binarne. Možemo prostor pretraživanja također smanjiti tako da promatramo samo linearna stabla operatora. Stablo je linearno ako je barem jedan operator svaki binarne operacije bazna tablica. Ako stablo nije linearno, onda je nelinearno. Linearno stablo spaja manje tablice, pa se obično brže izvrši, ali kada imamo distribuirane baze podataka možemo paralelno raditi spojeve.




**Slika 4.8** Linearno stablo operatora.




**Slika 4.9** Nelinearno stablo operatora.

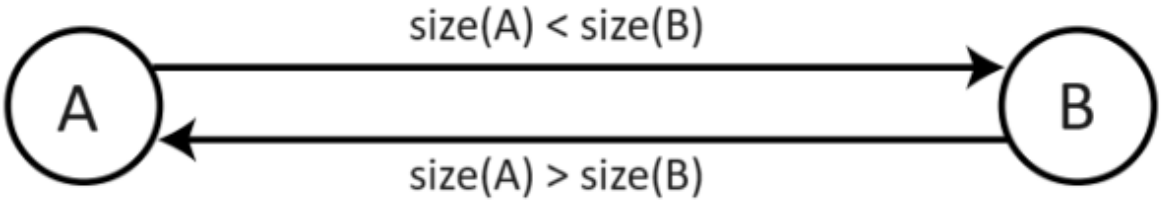
Model cijene predviđa koja će biti cijena izvršavanja nekog stabla operatora. Obično cijenu mjerimo u vremenu koje je potrebno da poslani upit dobije odgovor. Cijena ovisi o brzini lokalnog računala na kojem radi DBMS, o brzini komunikacije u mreži kojom su povezani naš DBMS i čvorovi distribuirane baze podataka, o broju čvorova s kojima treba komunicirati da bi se upit izvršio, o brzini računala na kojima su smješteni ti čvorovi, o količini podataka koje naš upit pokušava dohvatiti, itd.

Strategije pretraživanja mogu biti deterministčke ili nasumične. Deterministčke grade stablo operatora po nekim heuristikama i daju vrlo dobre rezultate, ali postaju prespore za upite koji pristupaju broju relacija većem od 5 ili 6. Za složenije upite se koriste nasumične strategije pretraživanja koje koriste pohlepne algoritme za dobivanje početnog stabla, zatim rade nasumične transformacije tog stabla dok ne dobiju stablo zadovoljavajuće cijene. Iako nasumična strategija ne daje jednako dobro rješenje kao deterministčka strategija, ukupno vrijeme potrebno za optimizaciju nasumičnom strategijom i za provođenje upita je manje od vremena potrebnog samo za optimizaciju deterministčkom strategijom za složenije upite.

Cilj optimizacije distribuiranog upita je smanjiti vrijeme potrebno za provođenje upita, a to postižemo tako da minimiziramo komunikaciju između čvorova, te u koraku provođenja upita na svakom od čvorova provedemo optimizaciju sada manjeg i centraliziranog upita. Kada spajamo udaljene tablice, moramo na nekom čvoru provesti to spajanje. To znači da trebamo

prenositi neke operande s jednog čvora na drugi. Da bismo minimizirali komunikaciju među čvorovima trebamo uzimati u obzir veličinu tablica koje sudjeluju u spoju i topologiju mreže.

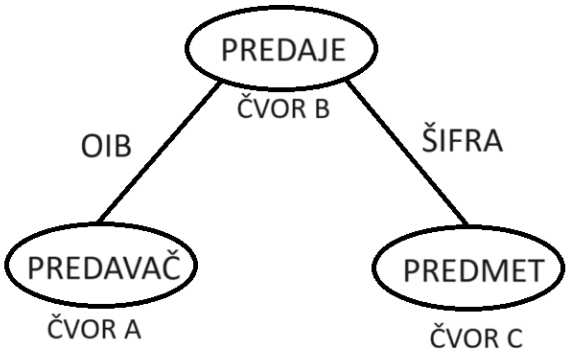
*Primjer 4.7*
Recimo da imamo dva čvora A i B. Neka želimo spojiti tablice koje se nalaze na tim čvorovima, tada trebamo podatke jednog od čvorova poslati drugom čvoru kako bismo na drugom čvoru mogli provesti spoj. U tom slučaju ćemo htjeti da čvor, koji ima manju tablicu kao operand spoja, šalje tablicu drugom čvoru.




**Slika 4.10** Spoj dvije udaljene tablice.

Ovdje također treba uzeti u obzir i topologiju mreže. Ako je naš DBMS spojen na čvor A i nema direktnu komunikaciju s čvorom B te vrijedi da je size(A) < size(B), tada će A poslati svoju tablicu čvoru B, koji će poslati spojenu tablicu A $\bowtie$ B natrag čvoru A, koji će ju poslati našem DBMS-u. U ovom slučaju bi bilo manje komunikacije ako čvor B pošalje svoju tablicu čvoru A, ali i tada možemo imati degenerirane slučajeve gdje se tablica čvora B sastoji od 1000 redaka, a tablica čvora A samo od jednog retka takvog da bi se on spojio samo s jednim retkom tablice B. Tada šaljemo 1000 redaka od čvora B do čvora A, umjesto da šaljemo jedan redak od čvora A do čvora B i jedan redak od čvora B do čvora A.
$\diamond$

*Primjer 4.8*
Pogledajmo sada upit koji treba spojiti tablice koje se nalaze na tri udaljena čvora. Neka se radi o tablicama
PREDAVAČ (oib, ime, prezime, plaća)
PREDMET (šifra, ime, semestar, ects)
PREDAJE (oib, šifra)
Te neka je potrebno provesti upit
PREDAVAČ $\bowtie_{OIB}$ PREDAJE $\bowtie_{ŠIFRA}$ PREDMET
Neka je mreža takva da su svi čvorovi međusobno povezani.




**Slika 4.11** Veze tablica u upitu.

Tada ima barem četiri moguća načina za provesti taj upit:

1) Čvorovi A i C pošalju tablice PREDAVAČ i PREDMET čvoru B, koji provede PREDAVAČ $\bowtie$ PREDAJE $\bowtie$ PREDMET
2) Čvor A pošalje tablicu PREDAVAČ čvoru B, koji provede PREDAVAČ$\bowtie$ PREDAJE, te tu, novu, tablicu pošalje čvoru C, koji provede (PREDAVAČ$\bowtie$ PREDAJE) $\bowtie$ PREDMET
3) Čvor C pošalje tablicu PREDMET čvoru B, koji provede PREDAJE$\bowtie$ PREDMET, te tu, novu, tablicu pošalje čvoru A, koji provede PREDAVAČ$\bowtie$ (PREDAJE $\bowtie$ PREDMET)
4) Čvor B pošalje tablicu PREDAJE čvoru C, koji provede PREDAJE$\bowtie$PREDMET, te tu, novu, tablicu pošalje čvoru A, koji provede PREDAVAČ$\bowtie$(PREDAJE $\bowtie$ PREDMET)

Postoji još načina za provesti upit. Za pronalazak što boljeg načina ćemo koristiti algoritme koje ćemo kasnije proučiti.

$\diamond$

Komunikaciju među čvorovima također možemo smanjiti koristeći poluspojeve umjesto spojeva. Naime, vrijedi

$$\begin{aligned} R \bowtie_A S &= (R \ltimes_A S) \bowtie_A S \\ &= R \bowtie_A (S \ltimes_A R) \\ &= (R \ltimes_A S) \bowtie_A (S \ltimes_A R) \end{aligned}$$

Pomoću gornjeg pravila, umjesto da šaljemo cijele tablice, možemo poslati samo atribute po kojima radimo spoj, pa samo retke koji će sudjelovati u spoju poslati natrag. Na taj način šaljemo znatno manje podataka ako se radi o tablicama s puno podataka.

Sada ćemo proučiti četiri algoritama za optimizaciju distribuiranog upita.

### 4.3.1 Dinamički algoritam: INGRES

Cilj dinamičkog algoritma optimizacije upita je smanjiti komunikacijsko vrijeme i vrijeme potrebno za dobivanje odgovora. Može se dogoditi da povećavanje jedne vrijednosti smanjuje drugu vrijednost. Naime, možemo povećati komunikacijsko vrijeme tako da komuniciramo s više čvorova, ali time dobijemo da čvorovi paralelno provode dijelove upita, pa vrijeme potrebno za dobivanje odgovora bude manje, te na kraju suma tih dviju vrijednosti bude manja nego što bi bila kad bismo prvo minimizirali komunikacijsko vrijeme.

Sljedeći algoritam uzima kao ulaz lokalizirani višerelacijski (multirelational) upit zapisan u relacijskoj algebri. Algoritam se izvršava na majstor čvoru, dok se izvršavanje dijelova upita provodi na majstor ili na šegrt čvoru ovisno o tome gdje algoritam procijeni da je efikasnije. Naime, ovaj algoritam odmah napravi i četvrti korak obrade upita, to jest, pri optimizaciji upita se paralelno i izvršava upit.

**Ulaz**: višerelacijski upit *VRU*
**Izlaz**: rezultat zadnjeg višerelacijskog upita *VRU'*
**Varijable**: *JRU* jednorelacijski upit, *Skup* je skup upita, *Skup_parova* skup parova (tablica, čvor)
**START**
**for**(svaki odvojivi *JRU* iz *VRU*)
    Izvrši *JRU*;
*Skup* = REDUCE(*VRU*); #*Skup* je sada skup ireducibilnih upita
**while**(!(isempty(*Skup*))
    *VRU'* = MIN_FRAGMENT(*Skup*);
    *Skup* = *Skup* \ {*VRU'*};
    *Skup_parova* = STRATEGIJA(*VRU'*);
    **for**(svaki par (*T*, *Č*) iz *Skup_parova*)
        Pošalji tablicu *T* čvoru *Č*;
    Izvrši *VRU';*
**END**

U algoritmu spominjemo odvojive upite (detachable queries). Upit je odvojiv (detachable) ako se može zapisati kao niz od dva upita tako da drugi upit koristi rezultat prethodnog upita, ako se ne može odvojit na dolje opisani način, onda je neodvojiv. Upit je neodvojiv (irreducible) ako je graf upita niz od dva čvora ili ciklus od k>2 čvorova. Za upit

*q*:

```
SELECT R2.A2, … ,Rn.An
from R1, … , Rn
where P1(R1.A1')
AND P2(R1.A1, … , Rn.An);
```

odvajanje upita *q* pišemo kao $q = q_1 \rightarrow q_2$, Odvajanjem zajedničke relacije $R_1$.

$q_1$:

```
SELECT R1.A1 INTO R1'
from R1
where P1(R1.A1');
```

$R_1$' je privremena relacija čiji rezultat koristi sljedeći odvojeni upit. $R_1$' je manja relacija od relacije $R_1$.

$q_2$:

```
SELECT R2.A2, … ,Rn.An
from R1', … , Rn
where P2(R1'.A1, … , Rn.An);
```

*Primjer 4.8*

Pogledajmo upit koji pronalazi sve predavače koji predaju predmet linearna algebra

*q*:

```
SELECT prezime
from PREDAVAČ, PREDAJE, PREDMET
where PREDAVAČ.oib = PREDAJE.oib
AND PREDAJE.šifra = PREDMET.šifra
AND ime = 'Linearna algebra';
```

Sada ćemo rastaviti upit *q* na upite $q_1$ i *s*, tada $q = q_1 \rightarrow s$, gdje je

$q_1$ :

```
SELECT šifra INTO VAR1
from PREDMET
where ime = 'Linearna algebra';
```

*s*:

```
SELECT prezime
from PREDAVAČ, PREDAJE, VAR1
where PREDAVAČ.oib = PREDAJE.oib
AND PREDAJE.šifra = VAR1.šifra;
```

Primijetimo da odvajamo od kraja. To jest, zadnji spoj u upitu prvi odvajamo. Rastavimo sada *s* na $q_2$ i $q_3$, tada s = $q_2 \rightarrow q_3$

$q_2$:
```
SELECT oib INTO VAR2
from PREDAJE, VAR1
where PREDAJE.šifra = VAR1.šifra;
```

$q_3$:
```
SELECT prezime
from PREDAVAČ, VAR2
where PREDAVAČ.oib = VAR2.oib;
```

Sada $q = q_1 \rightarrow q_2 \rightarrow q_3$
◇

Prvo izvršavamo jednorelacijske upite jer su to uglavnom projekcije i selekcije, to jest unarni operatori. Ovakav pristup se podudara s heuristikom da je dobro što prije u stablu operatora provesti unarne operatore. Zatim provodimo odvajanje početnog upita, nakon kojeg provodimo redom neodvojive podupite. Ne izvršavamo jednorelacijske upite koje smo već izvršili. U algoritmu spominjemo funkcije REDUCE, MIN_FRAGMENT i STRATEGIJA. Za upit *q* REDUCE vraća niz neodvojivih upita $q_1 \rightarrow \ldots \rightarrow q_n$ takvih da $q = q_1 \rightarrow \ldots \rightarrow q_n$. Funkcija MIN_FRAGMENT vraća upit koji koristi najmanje fragmente. Funkcija STRATEGIJA vraća parove (tablica *T*, čvor *Č*) takve da želimo poslati tablicu *T* čvoru *Č* kako bi se podupit, koji koristi tablicu *T*, proveo na čvoru *Č*. Nakon provedbe algoritma, imamo rezultat zadnjeg upita u nizu nerazdvojivih upita. Kako je niz tih upita ekvivalentan našem početnom upitu, tako će rezultat početnog upita biti jednak rezultatu zadnjeg upita u nizu nakon izvršavanja svih njegovih prethodnika redom.

### 4.3.2 Statički algoritam: R*

R* je deterministički algoritam koji pretražuje cijeli prostor pretraživanja, te unutar prostora pretraživanja pronalazi optimalnu strategiju. Bitno je napomenuti da to nije točno optimalna strategija, već optimalna strategija unutar prostora pretraživanja koji je biran pomoću heuristike, to jest, aproksimativno. Izgradnja prostora pretraživanja i samo pretraživanje tog prostora je skupo u smislu procesiranja i u smislu vremena, no ima smisla ako se optimizirani upit koristi često, jer se tada može rezultat optimizacije zapamtiti, pa se ne treba ponavljati više puta. Naravno, ne možemo pamtiti rezultat optimizacije za svaki upit jer različitih upita ima beskonačno mnogo. Algoritam se provodi na majstor čvoru, ali lokalna optimizacija, koja ovisi o

šegrt čvorovima, se provodi na šegrt čvorovima. Cilj algoritma je optimizirati vrijeme izvršavanje početnog upita i minimizirati potrebnu komunikaciju.

Sljedeći algoritam uzima kao ulaz lokalizirani višerelacijski upit zapisan u obliku stabla upita. Izvršavanje dijelova upita će se provesti na majstor ili na šegrt čvoru u četvrtom koraku obrade distribuiranog upita.

**Ulaz**: stablo upita *SU*
**Izlaz**: optimalna strategija *strat*
**Varijable**: $R_i$ relacija, $PP_{ij}$ put pristupa do relacije $R_i$, $best\_PP_i$ optimalna cijena puta pristupa do $R_i$, *temp_strat* strategija, *k* identifikator čvora, *lok_strat* strategija
**START**
**for**(svaka relacija $R_i$ iz *SU*)
    **for**(svaki $PP_{ij}$ do $R_i$)
        izračunaj COST($PP_{ij}$);
    $best\_PP_i$ = min(COST($PP_{ij}$));
**for**(svaki niz ($R_{i1}$, … , $R_{in}$) uz $i$ = 1, … , $n$! # $i$ je identifikator niza, drugi indeks je identifikator relacije)
    *temp_strat* = STRATEGIJA(BEST( … (BEST($PP_{i1} \bowtie R_{i2}$) $\bowtie R_{i3}$) … ) $\bowtie R_{in}$);
    izračunaj COST(*temp_strat* );
*strat* = strategija s minimalnom cijenom;
**for**(svaki čvor *k* koji pohranjuje relaciju koja je u *SU*)
    *lok_strat* = LOK_STRATEGIJA(*strat, k*);
    pošalji *lok_strat* čvoru *k*;
**END**

Algoritam prvo računa najefikasniji pristup svakoj relaciji koja je potrebna za provođenje upita. Zatim, za svaki mogući redoslijed relacija koje su potrebne za upit, pronalazi strategiju takvu da optimalno pristupa prvoj relaciji, pa optimalno pristupa spoju prve dvije itd. Algoritam koristi ulančano spajanje. Naime, mogli bismo probati prvo spajati bazne relacije, no to bi drastično povećalo prostor pretraživanja. Nakon što je pronađena optimalna strategija u ovakvom prostoru pretraživanja, algoritam šalje svakom čvoru lokalnu strategiju, koju svaki čvor optimizira lokalno. Algoritam koristi informacije i statistike koje su mu dostupne o mreži i o ostalim čvorovima, pa njegova preciznost ovisi o istima. U algoritmu spominjemo funkcije STRATEGIJA, BEST, COST i LOK_STRATEGIJA. Funkcija BEST vraća put pristupa do tablice, COST vraća cijenu puta pristupa ili strategije, STRATEGIJA pronalazi strategiju za provođenje distribuiranog upita, dok LOK_STRATEGIJA pronalazi strategiju za provođenje podupita na nekom čvoru. Najbitniji dio strategije je odrediti na kojem čvoru će se provesti koji spoj. Za čvor na kojem se provodi spoj imamo tri kandidata: čvor na kojem je prva relacija spoja, čvor na kojem je druga relacija spoja ili čvor na kojem je neka treća relacija s kojom će se

kasnije spajati spoj ovih prvih dviju relacija. Za slanje podataka među čvorovima imamo dvije metode:

1) Slanje cjeline (ship-whole) šalje cijelu tablicu na čvor na kojem će se provesti spajanje, te ju pohranjuje u privremenu tablicu, koja sudjeluje u spoju ili, ako je drugi operand spoja sortiran, prima n-torku po n-torku i odmah radi spoj u privremenoj tablici.
2) Dohvat po potrebi (fetch-as-needed) šalje vrijednosti, po kojima se radi spoj, čvoru koji sadrži drugi operand, te taj čvor odgovara s n-torkama koje će sudjelovati u spoju. Ova metoda je ekvivalentna poluspoju.

Relacija koja je operand spoja te čiji se redci čitaju zovemo vanjska (external) relacija spoja. Relacija koju pretražujemo za retke koji će sudjelovati u spoju zovemo unutarnja (internal) relacija spoja. Tako ih zovemo jer postoje dva algoritma za spajanje: ugniježđena petlja (nested-loop) i spoj stapanjem (merge-join). Prvi algoritam ima dvije ugniježđene petlje, kod kojih prva (vanjska) prolazi kroz prvu (vanjsku) relaciju, a druga (unutarnja) petlja za svaki redak prve relacije prolazi kroz drugu (unutarnju) relaciju i traži retke koji zadovoljavaju uvjet spoja. Drugi algoritam koristi činjenicu da su relacije sortirane u odnosu na uvjet spoja. Generalno je drugi algoritam brži, $O(n_1 + n_2)$, dok je prvi sporiji, $O(n_1 * n_2)$. Ali u nekim situacijama suma vremena provođenja sortiranja i drugog algoritma može biti veća od vremena provođenja prvog algoritma.

Sada ćemo navesti četiri moguće strategije.

1) Šaljemo cijelu vanjsku relaciju čvoru koji pohranjuje unutarnju relaciju. U ovoj strategiji možemo odmah spajati tablice kako dolaze n-torke.
2) Šaljemo cijelu unutarnju relaciju čvoru koji pohranjuje vanjsku relaciju. U ovoj strategiji moramo pohraniti dolazeću relaciju u privremenu relaciju koju ćemo kasnije spojiti. Ova strategija je poželjna ako je unutarnja relacija znatno manja od vanjske.
3) Dohvaćamo n-torke unutarnje relacije za svaku n-torku vanjske relacije. U ovoj strategiji čvorovi moraju slati više poruka međusobno, ali to će umanjiti ukupan sadržaj poruka jer se neće slati cijela relacija.
4) Šaljemo obje relacije trećem čvoru, te tamo vršimo spoj. Prvo šaljemo unutarnju relaciju koju pohranjujemo u privremenu tablicu. Zatim šaljemo vanjsku tablicu te, kao u strategiji 1), kako n-torke dolaze, spajamo ih s k-torkama unutarnje tablice.

*Primjer 4.9*
Recimo da imamo upit koji zahtjeva samo spajanje tablica
PREDMET (<u>šifra</u>, ime, semestar, ects)
i
UPISAO (<u>jmbag</u>, <u>šifra</u>, ocjena, godina_upisa)
s uvjetom PREDMET.šifra = UPISAO.šifra.
Neka su tablice PREDMET i UPISAO pohranjene na različitim čvorovima.

Pretpostavimo također da je PREDMET vanjska, a UPISAO unutarnja relacija u spoju.
Sada prema strategijama 1) - 4) imamo:

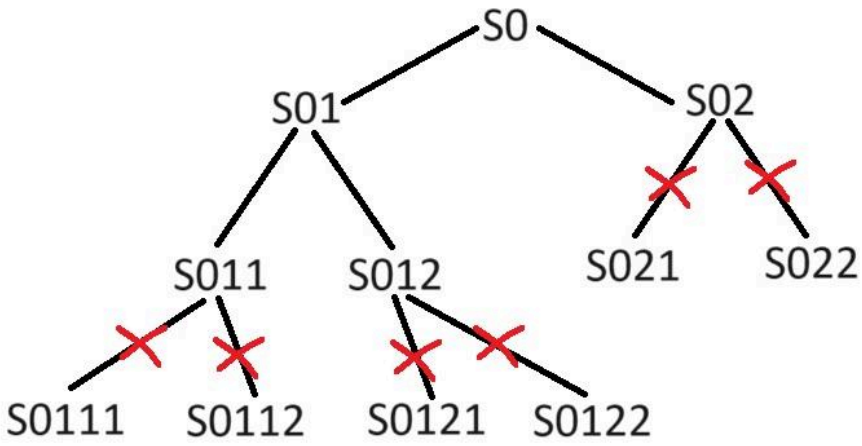
1) Cijelu tablicu PREDMET pošaljemo čvoru s tablicom UPISAO.
2) Cijelu tablicu UPISAO pošaljemo čvoru s tablicom PREDMET.
3) Dohvatimo n-torke tablice UPISAO koje bi se spojile s n-torkama tablice PREDMET.
4) Pošaljemo obje tablice nekom trećem čvoru.

Algoritam će na temelju dostupnih shema i statistika odlučiti koju strategiju primijeniti. Očito strategija 4) košta najviše, jer bismo slali obje tablice te nema treće tablice za kasnije spajanje. Ako je tablica PREDMET puno veća od tablice UPISAO, onda je strategija 2) vjerojatno najbolja, ako je suprotno, onda je to strategija 1). Također treba uzeti u obzir lokalnu brzinu obrade podataka. Ako je komunikacija mreže znatno brža od lokalne brzine obrade podataka, onda je potencijalno strategija 3) optimalna. Također ako se radi o malom broju parova koji će biti spojeni i ako algoritam uspije to predvidjeti na temelju dostupnih informacija, također će odabrati strategiju 3).
◇

### 4.3.3 Algoritam baziran na poluspoju: SDD-1

Algoritam SDD-1 je baziran na algoritmu penjanja na brdo (hill-climbing algorithm). Algoritam penjanja na brdo je pohlepan algoritam koji uzima graf upita kao ulaz, te na temelju sheme distribucije i statistike vezane za bazu pronalazi strategiju kojom se šalje što manje podataka među čvorovima, ne uzimajući u obzir cijenu slanja rezultata do majstor čvora. Algoritam prvo izabere neku očitu strategiju, te ju razdvaja na niz podstrategija koje su, kad se provedu redom, ekvivalentne u smislu rezultata, ali ne i efikasnosti, početnoj strategiji. Ako algoritam zaključi da je niz strategija efikasniji od početne strategije, početnu strategiju zamjenjuje tim nizom, te rekurzivno primjenjuje postupak na sve članove niza. Algoritam zapravo gradi stablo čiji će listovi u konačnici biti najbolja pronađena strategija. Problem algoritma je to što neke strategije koje od prve imaju veću cijenu, se mogu bolje optimizirati od nekih koje bi u prvom koraku algoritma odabrali. Algoritam može pronaći lokalni minimum i zbog toga promašiti globalni.

*Primjer 4.10*
Neka imamo neki upit za koji algoritam penjanja na brdo pronalazi strategiju. On će prvo izabrati početnu strategiju S0, pa ju razdvojiti, recimo, na strategije S01 i S02, te neka je cijena provođenja S01 i S02 manja od provođenja S0. Sada je S0 jednak nizu S01, S02. Analogno S01 zamjenjujemo nizom S011, S012. Pri razdvajanju strategije S02, dobiveni niz nema manju cijenu od same strategije S02, pa smatramo da S02 ne može biti bolji. Analogno ne razdvajamo S011 i S012. Sada izvođenje niza strategija S011, S012, S02 ima isti rezultat kao strategija S0, ali je izvođenje niza jeftinije u smislu funkcije cijene.




**Slika 4.12** Ilustracija algoritma penjanja na brdo.

◇

Algoritam SDD-1 koristi statistike o relacijama, čvorovima i mreži, te poluspojeve za minimizaciju komunikacije među čvorovima. Algoritam uzima stablo upita kao ulaz, te daje globalnu strategiju za provođenje upita koristeći poznate statistike i poznatu shemu distribucije.

**Ulaz**: stablo upita *SU*
**Izlaz**: globalna strategija *strat*
**Varijable**: *DPS* skup dobrih poluspojeva, *PS* poluspoj, *Č* čvor, *R* relacija
**START**
*strat* = LOK_OP_STRAT(*SU*);
EDIT_STATISTIC;
*DPS* = ∅;
**for**(svaki poluspoj *PS* iz *SU*)
    **if**(DOBAR(*PS*))
        *DPS* = *DPS* ∪ {*PS*};
**while**(*DPS* ≠ ∅)
    *PS* = NAJBOLJI(*DPS*);
    *DPS* = *DPS* \ {*PS*};
    *strat* = DOPUNI_STRAT(*SJ*);
    EDIT_STATISTIC;
    *DPS* = *DPS* \ {*PS* : *PS* nije dobar};
    *DPS* = *DPS* ∪ {*PS* : *PS* dobar};
*Č* = IZBOR_ČVORA(*strat*);
*strat* = DOPUNI_STRAT(slanje relacija čvoru *Č*);
**for**(svaka relacija $R_i$ na čvoru *Č*)
    **for**(svaki poluspoj *PS* relacija $R_i$ i $R_j$)
        **if**(NOT DOBAR_ZA_STRAT(*PS*))

*strat* = IZBACI(*PS*);

**END**

Algoritam prvo ukomponira lokalne operacije u strategiju. Zatim mijenja statistiku s obzirom na te lokalne operacije. Zatim stavlja sve dobre poluspojeve u skup te dok taj skup nije prazan uzima najbolji poluspoj iz skupa, pa ga ukomponira u strategiju. Pri svakom komponiranju se mijenja statistika, te na temelju te statistike se iz skupa dobrih poluspojeva miču svi poluspojevi koji više nisu dobri i dodaju se novi dobri poluspojevi. Nakon što je odabrana globalna strategija, određuje se čvor koji sadrži najviše podataka koji sudjeluju u upitu, pa se dodaje u strategiju slanje svih ostalih podataka, koji sudjeluju u upitu, na taj čvor. Na kraju algoritma se iz strategije izbacuju svi poluspojevi koji nisu dobri na tom čvoru. U algoritmu koristimo funkcije LOK_OP_STRAT, EDIT_STATISTIC, DOBAR, DOPUNI_STRAT, IZBOR_ČVORA, DOBAR_ZA_STRAT i IZBACI. Funkcija LOK_OP_STRAT gradi početnu strategiju na temelju lokalnih operacija, EDIT_STATISTIC mijenja statistiku na temelju novosaznalih informacija, DOBAR vraća *true* ili *false*, ovisno je li poluspoj dobar ili ne. Poluspoj je dobar ako je cijena provođenja poluspoja manja od njegovog doprinosa. DOPUNI_STRAT dopunjuje strategiju dodavanjem novog poluspoja, IZBOR_ČVORA bira čvor koji ima najviše podataka koji sudjeluju u upitu, DOBAR_ZA_STRAT vraća *true* ili *false*, ovisno je li cijena strategije, manja s poluspojem ili bez njega, te IZBACI izbacuje poluspoj iz strategije.

Sličnost SDD-1 algoritma i algoritma penjanja na brdo se vidi u njihovom ulazu, koji je isti, te u pohlepnom dijelu SDD-1 algoritma, gdje s uzima najbolji poluspoj u danom kontekstu.

### 4.3.4 Hibridni pristup

Dinamički i statički pristup imaju prednosti i mane. Dinamički pristup istovremeno optimizira i provodi upite, pa ne mora predviđati kako će rezultati nekih operacija izgledati, no baš zbog te prednosti se proces optimizacije mora ponavljati za svaki upit. Dakle, dinamički pristup je dobar za upite koji se ne pojavljuju često. S druge strane, statički pristup traje puno dulje, ali obično daje bolje rješenje. Za statički pristup je preciznost modela cijene i statistike iznimno bitna jer se na temelju njih rade predviđanja pri razvoju strategije. Statički pristup je bolji za upite koji se često koriste te za upite koji su vezani za neku aplikaciju i ne mijenjaju se, već se pokreću točno ti upiti, recimo pritiskom gumba. Veliki problem statičkog pristupa je nepouzdanost mreže. Strategija može koristiti neki čvor, koji nije dostupan u trenutku provođenja upita ili je preopterećen.

Zbog komplementarnog svojstva koje imaju dinamički i statički pristup, pojavljuje se hibridni pristup koji koristi prednosti i statičkog i dinamičkog pristupa minimizirajući njihove mane. Prvo gradimo statički plan koji odredi redoslijed operacija i metode pristupa, ali ignorira informacije o tome na kojem čvoru se nalazi koja relacija. Zatim, za vrijeme provođenja upita, se

dinamički generira plan koje će se relacije poslati s kojeg čvora kojem čvoru te na kojim čvorovima će se odvijati koje operacije.

## 4.4 Provođenje upita (query execution)

U zadnjem koraku obrade distribuiranog upita provodimo upit po planu određenom u prethodnom koraku. Korak provođenja upita se isprepliće s prethodnim korakom jer se neki algoritmi ili dijelovi algoritama za optimizaciju upita odvijaju u ovom koraku. U ovom koraku se obično provodi i optimizacija lokalnih upita, koja ovisi o lokalnim statistikama koje nisu dostupne majstor čvoru, na kojem se odvija globalna optimizacija. Konačan rezultat ovog koraka je rezultat upita.

# 5. Čuvanje integriteta u distribuiranim bazama podataka

Kao što smo napomenuli u prvom poglavlju, sustav nije baza podataka ako nema određena svojstva. Jedno od tih svojstava je integritet. Integritet za distribuiranu bazu podataka definiramo analogno kao za centraliziranu bazu podataka, ali ga puno teže održavamo. Zbog CAP teorema znamo da ne možemo imati distribuiranu bazu koja istovremeno ima integritet, dostupna je te radi u slučaju kvara dijela mreže. Ovaj problem rješavamo, to jest, umanjujemo implementiranjem protokola koji, ovisno o tome koja od tri svojstva baze podataka su nam bitnija, osiguravaju da manje bitna svojstva budu popravljiva nakon njihovog narušavanja. U nekim primjenama distribuirane baze podataka će integritet podataka biti neophodan, pa ćemo žrtvovati dostupnost baze u slučaju kvara dijela mreže. Ovakvi slučajevi su iznimno rijetki. U većini slučajeva žrtvovanje dostupnosti nije opcija. Za neke druge primjene će prekid rada baze biti preskup u odnosu na kasnije popravljanje integriteta, pa ćemo narušiti integritet radom sustava u slučaju kvara dijela mreže.

Integritet čuvamo lokalno na svakom čvoru i globalno na razini cijele baze podataka. Lokalno čuvanje integriteta je analogno čuvanju integriteta u centraliziranim bazama podataka. Čuvanje integriteta na globalnoj razini distribuirane baze podataka uzima u obzir replikaciju i fragmentaciju tablica. U slučaju repliciranih tablica trebamo se pobrinuti da se u slučaju mijenjanja podataka (unos, brisanje ili ažuriranje) promjena provede na svim čvorovima ili na što više njih. U slučaju fragmentiranih tablica se ne trebamo posebno brinuti oko integriteta, jer čuvanje tog integriteta je lokalno na čvoru na kojem je tablica. Trebamo, doduše, paziti na replikacije fragmenata, ali taj problem je analogan problemu s repliciranim cijelim tablicama.

## 5.1 Protokoli

Cilj protokola je osigurati čuvanje integriteta ili postići minimalno odstupanje baze od stanja konzistentnosti. Protokol će pokretati čvor koordinator (coordinator). To je obično čvor koji pokreće transakciju. U protokolu će sudjelovati čvorovi sudionici (participants). Sudionik je svaki čvor koji sudjeluje u provođenju transakcije, osim koordinatora.

Postoje protokoli obvezivanja (commit protocols), protokoli otkazivanja (termination protocols) i protokoli oporavka (recovery protocols). Protokoli obvezivanja osiguravaju dogovor svih sudionika da se transakcija provede. Protokoli otkazivanja osiguravaju da, ako se transakcija ne može provesti, recimo jer se dio transakcije ne može provesti na jednom od čvorova, tada svi čvorovi koji sudjeluju u transakciji znaju hoće li ju svejedno provesti, hoće li ju obustaviti ili hoće li ju ponovno probati provesti, ali koristeći neke druge čvorove sudionike. Protokoli oporavka osiguravaju da pri obustavljanju transakcije, čvor zna kako se vratit u stanje prije

provođenja dijelova transakcije. Svi ovi protokoli zajedno osiguravaju da transakcije budu atomarne nerazdvojive cjeline te da se mogu obustaviti nakon pokretanja. Pomoću ovih protokola možemo osigurati da prije i poslije provođenja transakcije distribuirana baza podataka bude konzistentna.

### 5.1.1 DPO: Dvofazni protokol obvezivanja (2PC: Two-Phase Commit Protocol)

Dvofazni protokol obvezivanja (DPO) osigurava da se svi čvorovi obvežu za provođenje transakcije prije nego što se ona krene provoditi. Protokol koristi dvije faze. U prvoj fazi koordinator daje do znanja sudionicima da se treba provesti transakcija, a u drugoj fazi koordinator šalje poruku svim sudionicima koja potvrđuje provođenje transakcije ili ju obustavlja. Slijedi opis DPO protokola koji ne predviđa greške.

Koordinator pamti dnevnik (log) događaja u kojem pamti tekuće transakcije. Koordinator zapisuje u dnevnik da je započeo obvezivanje te šalje poruku "priprema" (prepare) svim čvorovima sudionicima. Koordinator ulazi u stanje čekanja. Kad čvor sudionik primi poruku "priprema", on provjerava može li provesti transakciju, ako može, šalje poruku "glas-za-obvezivanje" (vote-commit) koordinatoru te ulazi u stanje pripravnosti. Ako sudionik ne može provesti transakciju, onda šalje poruku "glas-za-obustavljanje" (vote-abort) koordinatoru, te nastavlja raditi normalno, to jest, zaboravlja na transakciju. Ako je koordinator dobio poruku "glas-za-obvezivanje" od svih sudionika, šalje svim sudionicima poruku "globalno-obvezivanje" (global-commit) te ulazi u stanje obvezivanja. Ako je koordinator dobio poruku "glas-za-obustavljanje" od barem jednog sudionika, šalje svim sudionicima poruku "globalno-obustavljanje" (global-abort) te ulazi u stanje obustavljanja. Ovisno o globalnoj poruci koju sudionici dobe, provode ili ne provode transakciju te šalju potvrdnu poruku koordinatoru. Kad koordinator dobije potvrdnu poruku od svih sudionika, upisuje u svoj dnevnik da je transakcija gotova, bila ona izvršena ili ne. Gotova transakcija više nije tekuća.

Ovakav protokol ima očite probleme koji su posljedica pretpostavke da nema kvarova. Očito se može dogoditi da se poruka izgubi u mreži, te nikada ne stigne. Da bismo to riješili, dodajemo štoperice koje mjere koliko dugo se neki čvor nalazi u nekom stanju. Nakon što je čvor u nekom stanju dulje od predodređenog vremena, on pokreće prekovremeni protokol (timeout protocol) koji ćemo kasnije proučiti. Postoje i alternativne verzije DPO protokola koje dopuštaju da neki broj sudionika odbije provesti transakciju te ju se svejedno provede, ali samo na čvorovima koji su prihvatili tu obvezu. Ovaj oblik komunikacije, gdje imamo centralni koordinator te nema komunikacije među sudionicima, zovemo centralizirani DPO.

Drugi način implementacije je linearni DPO (nested 2PC). U ovoj implementaciji svim čvorovima koji sudjeluju u transakciji pridružujemo prirodan broj iz skupa $\{1, \ldots, N\}$, gdje je $N$ broj čvorova koji sudjeluju u transakciji. Koordinatoru pridružujemo broj 1. Za pokrenuti

protokol, koordinator šalje poruku “priprema” svom susjedu. Čvorovi su susjedi ako je razlika njima pridruženih brojeva jednaka 1 ili -1. Čvor A je desni susjed čvoru B ako je razlika brojeva koji su pridruženi čvoru A i čvoru B jednaka 1. Čvor A je lijevi susjed čvoru B ako je razlika brojeva koji su pridruženi čvoru A i čvoru B jednaka -1. Kada sudionik primi poruku “priprema”, ako može provesti transakciju, proslijedi poruku “priprema” svom desnom susjedu. Kada sudionik primi poruku “priprema”, a ne može provesti transakciju ili je primio poruku “globalno-obustavljanje”, on šalje poruku “globalno-obustavljanje” svojem lijevom susjedu. Kada čvor s brojem *N* primi poruku “priprema”, ako može provesti transakciju, šalje svom lijevom susjedu poruku “globalno-obvezivanje”. Kada sudionik primi poruku “globalno-obvezivanje”, on ju prosljeđuje svojem lijevom susjedu. Kada koordinator primi poruku “globalno-obvezivanje” zna da su ju primili svi sudionici, te proveli transakciju, a kada primi poruku “globalno-obustavljanje” zna da su svi sudionici koji znaju za transakciju i odustali od nje. Čvorovi ulaze u adekvatna stanja ovisno o porukama koje dobivaju.

Ovaj način implementacije je zanimljiv u odnosu na prethodni jer se može dogoditi da čvorovi s malim brojem ne mogu provesti transakciju te se tada ne stupa u komunikaciju s čvorovima kojima su pridruženi veći brojevi. No ovaj način komunikacije ne implementira paralelizam, pa provođenje linearnog DPO-a traje dugo.

Treći način implementacije je distribuirani DPO (distributed 2PC). U ovoj implementaciji sudionici komuniciraju svi sa svima. Koordinator šalje poruku “priprema” svim sudionicima te svaki sudionik šalje svoj glas svim ostalim sudionicima i koordinatoru. Ovdje nema poruka globalnih obvezivanja i obustavljanja, već svi sudionici donose odluku na temelju broja sudionika koji su se obvezali provesti transakciju.

### 5.1.2 Specijalizacija DPO protokola

DPO možemo unaprijediti tako da ga specijaliziramo za određene tipove upita. U ovom kontekstu dijelimo upite na dvije skupine. Prva skupina su upiti koji samo čitaju (read-only) i upiti koji parcijalno samo čitaju (partially read-only). Za upite koji ažuriraju podatke u bazi podataka, ali veliki dio svakog od tih upita se sastoji od čitanja podataka, kažemo da su upiti koji parcijalno samo čitaju podatke. Druga skupina su općeniti upiti koji ažuriraju podatke. DPO s pretpostavljenim obustavljanjem (presumed abort 2PC) je specijaliziran za prvu skupinu upita, dok je DPO s pretpostavljenim obvezivanjem (presumed commit 2PC) specijaliziran za drugu skupinu upita. Razlika između skupina upita je ta da je manji problem prikazati pogrešne informacije nego unijeti pogrešne informacije. Pa kod read-only upita žrtvujemo malo preciznosti za malo efikasnosti.

#### 5.1.2.1 DPO s pretpostavljenim obustavljanjem

DPO s pretpostavljenim obustavljanjem ima dodatno svojstvo da bilo koja poruka koordinatoru koja je vezana za transakciju koja nije tekuća dobije naredbu za obustavljanje te transakcije kao odgovor. Koordinator pamti u svom dnevniku koje transakcije su tekuće i koje nisu. Ovim dodatnim svojstvom koordinator može otpisati transakciju čim jedan sudionik pošalje poruku "glas-za-obustavljanje" te automatski svim drugim sudionicima javljati da se transakcija obustavlja. Zbog smanjene komunikacije i smanjenog pisanja u dnevnik, DPO s pretpostavljenim obustavljanjem je efikasniji od običnog DPO.

#### 5.1.2.2 DPO s pretpostavljenim obvezivanjem

DPO s pretpostavljenim obvezivanjem pretpostavlja da ako neka transakcija nije tekuća, onda je ona obvezana, to jest provedena ili su se svi sudionici obvezali provesti ju. Zamislimo sljedeću situaciju. Koordinator je poslao "priprema" poruku svim sudionicima te čeka njihove odgovore. Prije nego što su svi odgovori stigli, čvor koordinatora se kvari i prestaje raditi. Čvorovi sudionici će pokrenuti svoj prekovremeni protokol te provesti transakciju zbog toga što smo pretpostavili da se obvezujemo provesti transakciju ako nije tekuća. Kada se koordinator oporavi, on će imati neprovedenu transakciju pa ćemo imati nekonzistentnost u bazi podataka. Da bismo to izbjegli, sudionici i koordinator će zapisivati više podataka u svoj dnevnik. U ovom protokolu koordinator u svoj dnevnik zapiše sve sudionike u transakciji pri slanju "priprema" poruka. Kada sudionici dobiju poruku, oni, nakon odluke, šalju povratnu informaciju koordinatoru te pišu svoju odluku u svoj dnevnik. Kada koordinator primi odgovore od svih sudionika, on šalje globalnu poruku s konačnom odlukom svim sudionicima, te zaboravlja na transakciju. Ne čeka potvrdne poruke da su sudionici primili globalnu poruku jer, ako ju ne prime, svejedno će provesti transakciju. Ovaj protokol smanjuje broj potrebnih poruka među čvorovima, ali umjesto toga pamti više informacija u svom dnevniku.

### 5.1.3 Kvar čvorova

Gore opisani DPO i njegove varijante ne uzimaju u obzir mogućnost kvara nekog od čvorova. DPO s pretpostavljenim obvezivanjem rješava problem u slučaju kvara koordinatora, ali ne i proizvoljnog čvora. Sada ćemo početnom nespecijaliziranom DPO-u dodati prekovremeni protokol (termination protocol) i protokol oporavka (recovery protocol). Kasnije ćemo objasniti i trofazni protokol obvezivanja, TPO (Three-Phase Commit Protocol, 3PC).

#### 5.1.3.1 Prekovremeni protokol

Prekovremeni protokol provodimo u slučaju da neki čvor, bio on koordinator ili sudionik, ne dobije očekivanu poruku u predodređenog vremenu. Tada pretpostavljamo da je čvor, čija poruka nije stigla, u kvaru. Protokol nalaže različite postupke ovisno o trenutku i mjestu kvara.

Koordinator može ući u prekovremeno čekanje u jednom od dva trenutka u procesu provođenja transakcije: dok čeka da mu svi sudionici jave svoje glasove ili dok čeka da mu svi sudionici jave da su primili njegovu konačnu odluku. U prvom slučaju će koordinator poslati "globalno-obustavljanje" svim sudionicima i u svoj dnevnik zapisati da je transakcija obustavljena. U drugom slučaju koordinator šalje poruke "globalno-obvezivanje" ili "globalno-obustavljanje" čvorovima koji nisu potvrdili primitak, dok ga ne potvrde. U drugom slučaju imamo problem jer ne znamo radi li se o kvaru čvora ili je došlo do greške u mreži. Također ne znamo je li čvor, koji ne odgovara, proveo lokalnu transakciju ili nije. U slučaju da nije, koordinator bi trebao svim svim drugim sudionicima javiti da vrate svoje lokalne podatke u stanje prije provođenja lokalnih transakcija, to jest da ipak obustave transakciju.

Sudionik može ući u prekovremeno čekanje samo u jednom trenutku: dok čeka odluku koordinatora. Napomenimo da, ako sudionik zna da će se transakcija obustaviti ako barem jedan sudionik pošalje "glas-za-obustavljanje", te je on poslao "glas-za-obustavljanje", sudionik može odmah pretpostaviti da će se transakcije obustaviti i nastaviti dalje s normalnim radom. Ako sudionik ne zna hoće li se transakcija provesti ili obustaviti te uđe u prekovremeno čekanje odluke koordinatora, on postaje blokiran dok ne sazna od koordinatora ili drugih sudionika provodi li se transakcija ili ne.

U gore opisanom protokolu, ako se dogodi prekovremeno čekanje, sudionik koji čeka postaje blokiran. To je vrlo nepoželjno, pa ćemo napraviti neka poboljšanja. Ako se radi o centraliziranom načinu komunikacije, nemamo poboljšanja, pa je i centralizirana mreža nepoželjna. Ako sudionici mogu međusobno komunicirati, onda blokirani sudionik $P_i$ može pitati sve ostale sudionike $P_j$ što oni znaju. Drugi sudionici mogu biti u jednom od tri stanja:

1) $P_j$ nije još dobio poruku "priprema" pa niti ne zna da se transakcija pokušava provesti. Tada $P_j$ odgovara sudioniku $P_i$ s "glas-za-obustavljanje" te istu poruku šalje koordinatoru. Kako $P_j$ još nije dobio poruku "priprema", znamo da koordinator još nije dobio glas svih sudionika, pa će sada dobiti "glas-za-obustavljanje" te odlučiti obustaviti transakciju.
2) $P_j$ čeka odluku koordinatora i glasao je za provođeje transakcije. Da je $P_j$ glasao za obustavu transakcije, ne bi čekao odgovor koordinatora. U ovom slučaju $P_j$ ne može pomoći sudioniku $P_i$ saznati provodi li se transakcija ili ne.
3) $P_j$ zna odluku koordinatora, pa će poslati poruku "globalno-obvezivanje" ili "globalno-obustavljanje" sudioniku $P_i$.

$P_i$ interpretira poruke drugih sudionika na sljedeći način:

1) Ako je primio "glas-za-obustavljanje" od barem jednog sudionika, može obustaviti transakciju, jer će koordinator također odlučiti obustaviti transakciju.
2) Ako je primio barem jednu poruku "globalno-obustavljanje" ili "globalno-obvezivanje", tada ne može primiti suprotnu globalnu poruku zbog rada DPO-a. Ne može se dogoditi da neki sudionici dobe poruku "globalno-obustavljanje" od koordinatora, a neki "globalno-obvezivanje" ili obrnuto. Pa u ovom slučaju $P_i$ može saznati odluku koordinatora.
3) Ako svi sudionici čekaju odluku koordinatora, sustav je blokiran. Potencijalno, ako sudionici mogu zaključiti da je došlo do kvara koordinatora, tada mogu međusobno izabrati novog koordinatora koji će ponovno pokrenuti transakciju.

#### 5.1.3.2 Protokol oporavka

Protokol oporavka osigurava da se transakcija može provesti u slučaju kvara nekog čvora. Protokol zapravo nastavlja transakciju nakon što je čvor ponovno u funkcionalnom stanju. Procedura koju provodimo ovisi o čvoru koji se oporavlja: koordinator ili sudionik; te o trenutku u kojem se dogodio kvar.

Koordinator je mogao prestati raditi u jednom od dva trenutka: dok čeka glasove sudionika ili dok čeka potvrde sudionika da su primili njegovu odluku. U slučaju kvara nakon slanja poruke "priprema" te prije primanja svih glasova, koordinator će ponovno poslati poruku "priprema" svim sudionicima i nastaviti s radom DPO-a nakon dolaska u funkcionalno stanje. U slučaju kvara za vrijeme čekanja potvrda sudionika, koordinator provjerava koje sve potvrde koje su stigle u međuvremenu. Ako su stigle sve potvrde, transakcija je uspješno provedena ili obustavljena. Ako neke potvrde nisu stigle, onda se pokreće prekovremeni protokol.

Sudionik je mogao prestati raditi dok je čekao odluku koordinatora, za vrijeme provođenja lokalne transakcije ili za vrijeme normalnog rada. U slučaju kvara rada prije primanja poruke "priprema", to jest, za vrijeme normalnog rada, sudionik samo nastavlja s normalnim radom nakon što postane funkcionalan, jer će tada koordinator ponovno poslati poruku "priprema", a u slučaju kvara pri izvršavanju lokalne transakcije će, na temelju dnevnika, nastaviti provoditi lokalnu transakciju gdje je stao te nastaviti s normalnim radom. U slučaju kvara dok čeka odluku koordinatora, kada je sudionik ponovno funkcionalan i u međuvremenu nije dobio odluku koordinatora, on pokreće prekovremeni protokol.

Gornje analize pretpostavljaju da, ako je čvor u kvaru, tada poruke mogu i dalje pristizati u međuspremnik, te biti pročitane kada čvor bude funkcionalan. Također pretpostavljamo da se kvar ne može dogoditi između pisanja informacija u dnevnik i slanja poruka te između primanja poruka i slanja odgovora na te poruke. Ova ograničenja nisu istinita u realnom slučaju.

#### 5.1.3.3 TPO: Trofazni protokol obvezivanja (3PC: Three-Phase Commit Protocol)

DPO je blokirajući protokol u smislu da, u slučaju kvara nekog čvora, provođenje transakcije staje sve dok taj čvor ne postane ponovno funkcionalan jer je potencijalno neki od čvorova proveo lokalnu transakciju, a neki nije. Takvu transakciju moramo provesti do kraja. U realnim situacijama to može trajati predugo. Nekada bi bilo profitabilnije samo obustaviti transakciju. Možemo također isključiti pokvareni čvor iz mreže, ako je to moguće, te u nekom kasnijem trenutku, kada će cijeli sustav biti isključen radi održavanja ili nekih drugih razloga, ažurirati isključeni čvor i vratiti ga u mrežu, ako je to moguće. TPO omogućuje obustavljanje transakcije unatoč kvaru jednog od čvorova.

TPO radi slično kao DPO, samo ima još jedan međukorak. Koordinator šalje poruku "priprema" svim sudionicima. Oni šalju svoje glasove kao odgovore. Ako je barem jedan od glasova negativan, koordinator šalje poruku "globalno-obustavljanje" svim sudionicima te se transakcija obustavlja. Ako je koordinator primio glas od svakog sudionika te nije primio niti jedan negativan glas, on šalje poruku "predobveza" (precommit) svim sudionicima. Kada sudionik primi poruku "predobveza", on šalje poruku "spreman-za-obvezu" (ready-to-commit) koordinatoru. Nakon što koordinator dobije "spreman-za-obvezu" poruku od svih sudionika, on šalje svim sudionicima poruku "globalno-obvezivanje". Sudionici koji prime poruku "globalno-obvezivanje" šalju potvrdnu poruku koordinatoru, te provode transakciju. Slanje i primanje svake poruke svaki čvor piše u svoj dnevnik.

Dodavanje još jedne faze omogućuje jednostavno rješenje u slučaju kvara koordinatora. Ako neki od sudionika nije dobio poruku "globalno-obvezivanje" u predviđenom roku, a dobio je poruku "predobveza", tada zna da svi sudionici, pa čak i koordinator mogu provesti transakciju, pa ju provodi. Ako sudionik nije dobio poruku "predobveza" niti "globalno-obustavljanje" u predviđenom roku, tada može to javiti drugim sudionicima, pa svi sudionici mogu obustaviti transakciju, jer ju nitko još nije proveo lokalno, jer ne može sudionik primiti poruku "globalno-obvezivanje" bez da svi sudionici prime poruku "predobveza". Uz običan DPO gornji slučajevi nisu mogući.

### 5.1.4 Particija mreže

U slučaju particije mreže bismo htjeli da baza nastavi funkcionirati. Nažalost to nije moguće osim u slučaju jednostavne particije, to jest, u slučaju kada se mreža razdvoji na dvije podmreže, ne više. Problem su transakcije koje su bile u tijeku za vrijeme nastanka particije. Transakcije koje se pokreću nakon nastanka particije, a pristupaju podatcima koji nisu dio podmreže u kojoj su te transakcije pokrenute će jednostavno biti odbijene dok se mreža ne popravi. Imamo dva pristupa: možemo nastaviti rad obje podmreže, ali ćemo izgubiti integritet baze ili možemo ugasiti jednu podmrežu te zadržati integritet ali izgubiti dio dostupnosti.

Ako se dogodi particija mreže za vrijeme provođenja transakcije, tada će jedna podmreža imati koordinatora u sebi a druga neće. Sudionici transakcije, koji su u drugoj mreži, će izabrati novog koordinatora. Ali sada može biti slučaj da će se transakcija u prvoj podmreži provesti, a u drugoj ne. Naime, može biti slučaj da je koordinator odlučio obustaviti transakciju, ali sudionici u drugoj podmreži to ne znaju, te će oni ponovno pokrenuti transakciju s novim koordinatorom. Ovaj problem možemo riješiti tako da čvorovi u svakoj podmreži, međusobnom komunikacijom, shvate ima li ih manje od pola, te ako ima, svi prestaju raditi. Na ovaj način smo ugasili manju podmrežu, te osigurali konzistentnost cijele baze. Prije integracije ugašene podmreže u neugašenu, njezini će se čvorovi trebati ažurirati da budu u skladu s čvorovima u neugašenoj podmreži. Ovo rješenje ne radi u slučaju da particija mreže sadrži više od dvije podmreže, jer se tada može dogoditi da svaka podmreža sadrži manje od pola ukupnog broja čvorova, pa se sve podmreže ugase. Također u slučaju da imamo paran broj čvorova, može se dogoditi da svaka od dvije podmreže sadrži točno polovicu ukupnog broja čvorova, tada bismo također ugasili sve (obje) podmreže.

Ako želimo nastaviti rad obje podmreže u slučaju particije mreže, trebamo novu verziju TPO-a. Prvo unapređenje koje ćemo dodati je novi sustav glasanja. Za ukupan broj glasova $G$, definiramo broj $G_+$ kao minimalan broj glasova potreban da bi koordinator odlučio provesti transakciju, te $G_-$ kao minimalan broj glasova potreban da bi koordinator odlučio obustaviti transakciju. Dodajemo i uvjet $G_+ + G_- \geq G$ da se ne bi dogodilo da koordinator primi dovoljan broj glasova za provesti transakciju, a zatim dobije dovoljan broj glasova za obustaviti ju. Želimo da, nakon prvog slučaja, drugi slučaj bude nemoguć i obratno. Analogno poruci "predobveza" dodajemo poruku "predodgoda" koja se šalje sudionicima kako oni ne bi odmah obustavili transakciju, već čekali poruku "globalno-obustavljanje". Dodavanjem ovog koraka prije obustavljanja transakcije osiguravamo atomarnost transakcije. Izbjegavamo situaciju da se transakcija na dva načina obustavi u različitim podmrežama. Pri particiji mreže treba uzeti u obzir i način replikacije podataka.

# 6. Studijski primjer

Studijski primjer je realiziran koristeći Microsoft SQL Server (MSS) i Microsoft SQL Server Management Studio (MSSMS). U studijskom primjeru su napravljena dva servera: MSSQLSERVER01 i MSSQLSERVER02 na istom računalu, te je početna centralizirana baza podataka fragmentirana i pretvorena u distribuiranu bazu podataka koristeći postupke opisane u ovom radu i ta dva servera. Centralizirana baza FAKULTET je pretvorena u distribuiranu bazu podataka koja se sastoji od čvorova PREDIPLOMSKI_FRAGMENT i DIPLOMSKI_FRAGMENT.

Baza FAKULTET:

STUDENT (jmbag, ime, prezime, godina_studija)

PREDAVAČ (oib, ime, prezime, plaća)

PREDMET (šifra, ime, ects, semestar)

PREDAJE (oib, šifra)

UPISAO (jmbag, šifra, ocjena, godina_upisa)

U centraliziranoj bazi je fragmentirana tablica STUDENT na temelju atributa godina_studija tako da studenti, koji su trenutno upisani ne preddiplomski studij (prve tri godine), budu na fragmentu PREDDIPLOMSKI_FRAGMENT, a studenti, koji su trenutno upisani na diplomski studij (zadnje dvije godine), budu na fragmentu DIPLOMSKI_FRAGMENT. Nakon što je provedena primarna horizontalna fragmentacija nad tablicom STUDENT, treba provesti izvedenu horizontalnu fragmentaciju nad tablicom UPISAO. Svi redci tablice upisao, čiji student je trenutno na diplomskom studiju, se nalaze na fragmentu DIPLOMSKI_FRAGMENT. Kada se upisuje novi student na fakultet, on će se upisivati na prvu godinu, pa će se on i njegovi upisi pamtiti na fragmentu PREDDIPLOMSKI_FRAGMENT. Ovdje je bitno napomenuti da, u slučaju upisa studenta na diplomski studij nakon završenog preddiplomskog studija, treba prebaciti sve retke tablice UPISAO vezane za tog studenta na DIPLOMSKI_FRAGMENT. Tablica PREDAVAČ je fragmentirana tako da na jednom fragmentu pamtimo plaću, a na drugom ime i prezime predavača. Zbog prirode vertikalne fragmentacije, oib pamtimo na oba fragmenta. Fragment tablice PREDAVAČ koji sadrži plaću, koja može biti povjerljivi podatak, možemo pamtiti na čvoru s većim stupnjem sigurnosti. U studijskom

primjeru se plaća predavača pamti na diplomskom fragmentu, a ime i prezime na preddiplomskom fragmentu. Tablice PREDAJE i PREDMET su replicirane.

Fragment PREDDIPLOMSKI_FRAGMENT:
STUDENT (jmbag, ime, prezime, godina_studija) [godina_studija < 4]
PREDAVAČ (oib, ime, prezime)
PREDMET (šifra, ime, ects, semestar)
PREDAJE (oib, šifra)
UPISAO (jmbag, šifra, ocjena, godina_upisa) [upisi studenata koji su trenutno na preddiplomskom studiju]

Kod za kreaciju tablica preddiplomskog fragmenta:

```
CREATE TABLE student (
    jmbag char(10) NOT NULL,
    ime varchar(20) NOT NULL,
    prezime varchar(20) NOT NULL,
      godina_studija int NOT NULL,
      CONSTRAINT student_chk_digits_jmbag CHECK (jmbag NOT LIKE '%[^0-9]%'),
      CONSTRAINT  student_chk_godina_studija  CHECK  (godina_studija   in  (1,  2,
3)),
    CONSTRAINT student_pk_student PRIMARY KEY (jmbag)
);

CREATE TABLE predavac (
    oib char(11) NOT NULL,
    ime varchar(20) NOT NULL,
    prezime varchar(20) NOT NULL,
      CONSTRAINT predavac_chk_digits_oib CHECK (oib NOT LIKE '%[^0-9]%'),
    CONSTRAINT predavac_pk_predavac PRIMARY KEY (oib)
);

CREATE TABLE predmet (
    sifra char(5) NOT NULL,
    ime varchar(40) NOT NULL,
    ects int NOT NULL,
```

```
      semestar int NOT NULL,
      CONSTRAINT predmet_chk_digits_sifra CHECK (sifra  NOT LIKE '%[^0-9]%'),
      CONSTRAINT predmet_chk_semestar CHECK (semestar in (1, 2, 3, 4, 5, 6, 7,
8, 9, 10)),
    CONSTRAINT predmet_pk_predmet PRIMARY KEY (sifra)
);

CREATE TABLE predaje (
    oib char(11) NOT NULL,
      sifra char(5) NOT NULL,
      CONSTRAINT predaje_fk_oib FOREIGN KEY(oib ) REFERENCES predavac(oib),
      CONSTRAINT    predaje_fk_sifra    FOREIGN    KEY    (sifra)    REFERENCES
predmet(sifra),
    CONSTRAINT predaje_pk_predaje PRIMARY KEY (oib, sifra)
);

CREATE TABLE upisao (
    jmbag char(10) NOT NULL,
      sifra char(5) NOT NULL,
      ocjena int,
      godina_upisa int NOT NULL,
      CONSTRAINT upisao_chk_ocjena CHECK (ocjena  in ('1', '2', '3', '4', '5',
NULL)),
      CONSTRAINT upisao_chk_godina_upisa CHECK (godina_upisa > 2020),
      CONSTRAINT upisao_fk_jmbag FOREIGN KEY(jmbag ) REFERENCES student(jmbag),
      CONSTRAINT upisao_fk_sifra FOREIGN KEY (sifra) REFERENCES predmet(sifra),
    CONSTRAINT upisao_pk_upisao PRIMARY KEY (jmbag, sifra)
);
```

Fragment DIPLOMSKI_FRAGMENT:

STUDENT (jmbag, ime, prezime, godina_studija) [godina_studija > 3]

PREDAVAČ (oib, plaća)

PREDMET (šifra, ime, ects, semestar)

PREDAJE (oib, šifra)

UPISAO (jmbag, šifra, ocjena, godina_upisa) [upisi studenata koji su trenutno na diplomskom studiju]

Kod za kreaciju tablica diplomskog fragmenta:

```
CREATE TABLE student (
    jmbag char(10) NOT NULL,
    ime varchar(20) NOT NULL,
    prezime varchar(20) NOT NULL,
      godina_studija int NOT NULL,
      CONSTRAINT student_chk_digits_jmbag CHECK (jmbag NOT LIKE '%[^0-9]%'),
      CONSTRAINT student_chk_godina_studija CHECK (godina_studija  in (4, 5)),
    CONSTRAINT student_pk_student PRIMARY KEY (jmbag)
);

CREATE TABLE predavac (
    oib char(11) NOT NULL,
      placa int NOT NULL,
      CONSTRAINT predavac_chk_digits_oib CHECK (oib NOT LIKE '%[^0-9]%'),
    CONSTRAINT predavac_pk_predavac PRIMARY KEY (oib)
);

CREATE TABLE predmet (
    sifra char(5) NOT NULL,
    ime varchar(40) NOT NULL,
    ects int NOT NULL,
      semestar int NOT NULL,
      CONSTRAINT predmet_chk_digits_sifra CHECK (sifra  NOT LIKE '%[^0-9]%'),
      CONSTRAINT predmet_chk_semestar CHECK (semestar in (1, 2, 3, 4, 5, 6, 7,
8, 9, 10)),
    CONSTRAINT predmet_pk_predmet PRIMARY KEY (sifra)
);

CREATE TABLE predaje (
    oib char(11) NOT NULL,
      sifra char(5) NOT NULL,
      CONSTRAINT predaje_fk_oib FOREIGN KEY(oib ) REFERENCES predavac(oib),
```

```
      CONSTRAINT    predaje_fk_sifra    FOREIGN    KEY    (sifra)    REFERENCES
predmet(sifra),
    CONSTRAINT predaje_pk_predaje PRIMARY KEY (oib, sifra)
);

CREATE TABLE upisao (
    jmbag char(10) NOT NULL,
      sifra char(5) NOT NULL,
      ocjena int,
      godina_upisa int NOT NULL,
      CONSTRAINT upisao_chk_ocjena CHECK (ocjena  in ('1', '2', '3', '4', '5',
NULL)),
      CONSTRAINT upisao_chk_godina_upisa CHECK (godina_upisa > 2020),
      CONSTRAINT upisao_fk_jmbag FOREIGN KEY(jmbag ) REFERENCES student(jmbag),
      CONSTRAINT upisao_fk_sifra FOREIGN KEY (sifra) REFERENCES predmet(sifra),
    CONSTRAINT upisao_pk_upisao PRIMARY KEY (jmbag, sifra)
);
```

U studijski primjer nije ukomponiran DBMS koji bi radio za distribuirane baze podataka. MSSMS ne zna da se radi o distribuiranoj bazi podataka, pa neće automatski pretvoriti upit oblika:

```
SELECT * from predavac;
```

u upit oblika:

```
select A.oib, ime, prezime, placa from
[MSSQLSERVER02].preddiplomski_fragment.dbo.predavac as A
join
[MSSQLSERVER01].diplomski_fragment.dbo.predavac as B
on A.oib = B.oib
```

Drugi upit bi trebalo ručno upisati. Analogna situacija je prisutna u slučaju ubacivanja podataka u bazu podataka. Pokušaj unosa novog predavača je zapravo unos u dvije tablice, to jest u dva vertikalna fragmenta. Također unos u replicirane tablice se treba provest na dvije tablice, recimo da želimo unijeti novi predmet u tablicu PREDMET:

```
INSERT INTO predmet(sifra, ime, ects, semestar)
```

```
VALUES
('10101', 'Elementarna matematika', 15, 1);
```

Umjesto gornjeg upita zapravo treba provesti sljedeći upit:

```
INSERT INTO [MSSQLSERVER01].predmet(sifra, ime, ects, semestar)
VALUES
('10101', 'Elementarna matematika', 15, 1),
INSERT INTO [MSSQLSERVER02].predmet(sifra, ime, ects, semestar)
VALUES
('10101', 'Elementarna matematika', 15, 1);
```

# Zaključak

U radu se bavimo teorijskim razlozima i postupcima za distribuciju baze podataka. Neka razmatranja spominju aplikacije i statistike o kojima ovise rezultati tih razmatranja. Ta konkretna ovisnost je izvan dosega ovog rada, ali je važna za primjenu algoritama koji se obrađuju u radu. Mnogi algoritmi u radu koriste statistike vezane za baze podataka, no ne pokazuju se primjeri tih baza i tih statistika, već se u radu uzimaju zdravo za gotovo. Također je izvan dosega rada bilokakva analiza pravih praktičnih podataka kao što su brzina komunikacijske mreže ili snaga procesora koji pristupa datotekama koje tvore bazu podataka. Ove vrijednosti nisu zanemarive i nevezane za temu rada. U radu se isto tako ne razrađuje, već se samo spominje, razvoj distribuirane baze podataka od dna prema gore. Na temelju tema koje jesu obrađene u radu se može jasno vidjeti da ne postoji univerzalno "one-size-fits-all" rješenje za probleme distribuiranih baza podataka, već, kao većina modernih tehnologija, distribuirane baze podataka zahtjevaju agilnost i prilagodbu za dobivanje dobrih rezultata.

# Bibliografija

[1] M.T. Özsu, P. Valduriez: Principles of Distributed Database Systems. Fourth Edition. Springer, Cham CH, 2020.

[2] A. Petrov: Database Internals - A Deep Dive into How Distributed Data Systems Work. O'Reilly Media, Sebastopol CA, 2019.

[3] S.K. Rahimi, F.S. Haug: Distributed Database Management Systems - A Practical Approach. Wiley - IEEE Computer Society Press, Los Alamitos, CA, 2011.

[4] R. Manger: Baze podataka. Element d.o.o., drugo izdanje, Zagred, 2018.

[5] SQL Management Studio for SQL Server - User's Manual. 1999-2024 EMS Software Development

# Sažetak

U radu se bavimo distribuiranim relacijskim bazama podataka. Proučavamo njihove prednosti i mane u odnosu na centralizirane relacijske baze podataka te analiziramo proces distribucije centralizirane relacijske baze podataka kako bismo dobili distribuiranu bazu podataka. Proučavamo postupke primarne horizontalne, izvedene horizontalne te vertikalne fragmentacije. Proučavamo potrebne algoritme i protokole za obradu upita i za slučaj kvara nekih čvorova baze ili komunikacijske mreže. Bavimo se rastavom upita, lokalizacijom podataka, optimizacijom upita i provođenjem upita. Proučavamo algoritme INGRES, R*, te SDD-1. Analiziramo Dvofazni protokol obvezivanja i trofazni protokol obvezivanja te njihove varijante. Primjeri, pomoću kojih prikazujemo neke specifične situacije, su izvedeni iz studijskog primjera koji se nalazi u radu.

# Summary

In this work, we deal with distributed relational databases. We study their advantages and disadvantages compared to centralized relational databases and we analyze the process of distributing a centralized relational database in order to obtain a distributed database. We study procedures of primary horizontal, derived horizontal, and vertical fragmentation. We study the necessary algorithms and protocols for query processing and for the event of failure of some base nodes or the communication network. We deal with query processing, data localization, query optimization and query execution. We study the INGRES, R* and SDD-1 algorithms. We analyze the two-phase commit protocol, the three-phase commit protocol, and their variants. The examples, with which we show some specific situations, are derived from the case study found in the work.

# Životopis

Rođen sam 05.02.2000. godine u Zagrebu, gdje sam pohađao Osnovnu školu Julija Klovića. Nakon osnovne škole sam upisao informatički program u V. gimnaziji u Zagrebu. Nakon što sam maturirao, upisao sam inženjerski preddiplomski studij matematike na PMF-u u Zagrebu. Nakon završenog preddiplomskog studija sam upisao diplomski studij računarstvo i matematika na istom fakultetu.