Skripta iz predmeta

Projektovanje baza podataka

dr Jelena Graovac

Zasnovano na materijalima
prof. dr Saše Malkova i
prof. dr Gordane Pavlović-Lažetić

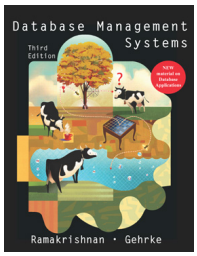
Beograd, 2016.

Predgovor

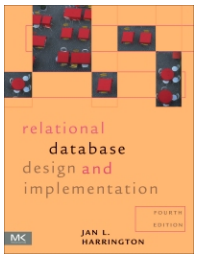
Ovaj tekst predstavlja skriptu iz predmeta ,,Projektovanje baza podataka", na Matematičkom fakultetu Univerziteta u Beogradu, i zasnovan je na materijalima *prof. dr Saše Malkova* (*www.matf.bg.ac.rs/~smalkov/nastava.ranije.html*) *i prof. dr Gordane Pavlović-Lažetić* iz istog predmeta.

Osnovna literatura:

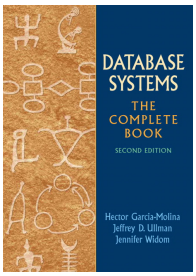
- Ramakrishnan, Gehrke, Database Management Systems, 3.ed, 2003.



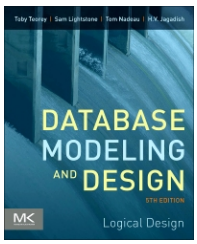
- Harrington, Relational Database Design and Implementation, 4.ed, 2016.



- Garcia-Molina, Ullman, Widom, Database Systems — The Complete Book, 2.ed, 2009.



- Teorey, Lightstone, Nadeau, Jagadish, Database Modeling and Design, 5.ed, 2011.



Skripta je prateći materijal pre svega studentima koji ovaj kurs slušaju u okviru svojih studija, ali i svima vama koji biste želeli da se upoznate sa ovom tematikom.

Ovaj materijal ne može zameniti pohađanje nastave niti drugu preporučenu literaturu.

Ukoliko ste pažljivi čitalac ove skripte, i ukoliko uočite bilo kakvu grešku ili propust, molim vas da mi to javite na jgraovac@matf.bg.ac.rs. Svi komentari, sugestije, kritike ali i pohvale vezane za ovaj materijal su dobrodošli.

Autor

Sadržaj

1

Uvod u projektovanje baza podataka

Tradicionalni pristup razvoju sistema za čuvanje i obradu podataka do kraja šezdesetih godina prošlog veka je u suštini bio *ad hoc*. Podaci su se čuvali u datotekama ili skupovima datoteka, a programi za obradu tih datoteka su se pisali u skladu sa aplikacijama. Nedostaci tog pristupa bili su mnogobrojni, a neki od njih su [10]:

- ponavljanje istih podataka za različite aplikacije,
- nekonzistentnost podataka,
- programi za obradu podataka zavisili su od načina struktuiranja podataka,
- obrada podataka je bila skupa,
- otežano korišćenje istih podataka od strane većeg broja korisnika,
- neadekvatna realizacija oporavka od pada sistema.

Šta je baza podataka?

- Pojam baza podataka pojavio se krajem šezdesetih godina 20. veka.
- Ovaj pojam je označavao skup međusobno povezanih podataka koji se čuvaju zajedno, i među kojima ima samo onoliko ponavljanja koliko je neophodno za njihovo optimalno korišćenje pri višekorisničkom radu.
- Podaci se pamte tako da budu nezavisni od programa koji ih koriste, i struktuiraju se tako da je omogućen porast baze.
- Posle svake aktivnosti nad bazom, koja predstavlja logičku celinu posla, stanje baze mora biti konzistentno (valjano).
- Dakle, podaci u bazi i odnosi među njima moraju zadovoljavati unapred zadate uslove koji oslikavaju deo realnosti modelirane podacima u bazi.
- Obrada podataka nije vezana za programski jezik opšte namene, već za viši ,,upitni“ jezik. Postoji dakle mogućnost deklarativnog postavljanja upita (opisujemo ,,šta“ a ne ,,kako“).

Osnovni pojmovi

- Podaci
- Baza podataka
- Sistem za upravljanje bazama podataka (SUBP)
- Administrator baze podataka (engl. database administrator – DBA)

Sistem za upravljanje bazama podataka (SUBP)

Sistem za upravljanje bazama podataka predstavlja softverski proizvod koji se nalazi između korisnika i fizičkih podataka u bazi. On omogućava efikasno formiranje, korišćenje i menjanje baze podataka.

Neke od usluga koje obezbeđuje SUBP su [8]:

- *Zaštita korisnika baze od detalja na hardverskom nivou.* Korisnik ne sme da trpi nikave posledice u slučaju promene fizičke strukture podataka.
- *Upravljanje sekundarnim skladištem.* SUBP obično uključuje usluge za brzi pristup određenim podacima pomoću indeksa, grupisanje srodnih podataka i obezbeđivanje efikasnog korišćenja bafera koji u memoriji računara na kom se izvršava SUBP zadržava podatke kojima se nedovano pristupalo.
- *Kontrola uporednog pristupa.* SUBP poseduje određene zaštitne mehanizme kako bi se osiguralo održavanje integriteta baze u slučaju konkurentnog višekorisničkog rada. Podsetimo se, dva posla se izvršavaju *konkurentno* ako pri radu upotrebljavaju iste resurse (procesor, memoriju, disk, podatke,...). Za ove poslove nije unapred poznata njihova međusobna vremenska lociranost (jedan posao može da počne i završi se pre početka drugog, može da počne pre a završi se posle završavanja drugog posla itd.). Ako postoji period vremena u kome su dva posla (doslovno) aktivna, onda kažemo da se izvršavaju *paralelno*.
- *Rekuperacija.* SUBP mora da bude fleksibilan, odnosno tolerantan na greške u softveru i otkazivanje hardvera. Zbog toga se često radi privremena replikacija podataka na više mesta kako bi se omogućila rekuperacija baze, odnosno, automatsko pronalaženje i ispravljanje različitih vrsta grešaka.
- *Bezbednost.* SUBP obezbeđuje mehanizme koji omogućavaju administratoru baze podataka da kontroliše pristup bazi (ko i gde može da pristupi kao i šta može da uradi u bazi podataka).
- ...

SUBP je obično zasnovan na nekom teorijskom modelu i podržava programske jezike za:

- definisanje strukture i uslova integriteta baze podataka (DDL, Data Definition Language)

- selekciju i izmene sadržaja baze podataka (DML, Data Modification Language)

Administrator baze podataka

Administrator baze podataka je IT profesionalac odgovoran za održavanje i funkcionisanje baze podataka. Neke od njegovih odgovornosti su:

- odgovoran je za implementaciju odluka administratora podataka,
- formiranje baze i implementiranje kontrolnih struktura,
- rad sistema i performanse,
- upravljanje autorizacijom i pravima,
- obezbeđivanje maksimalne raspoloživosti podataka,
- ...

Ciljevi upravljanja podacima

- Raspoloživost podataka
  - Administrator baze podataka je zadužen da uradi sve kako bi obezbedio da ,,sistem uvek bude raspoloživ".
  - Raspoloživost se praktično definiše kao odzivnost sistema u nekim garantovanim granicama (u ovom kontekstu se razmatra samo vreme odziva).
  - Sistem obično nije raspoloživ upravo kada je potreban.
    - U vreme kada je raspoloživost najpotrebnija, onda je i najteže ostvariva, zato što je tada sistem najviše opterećen.
    - Ako je sistem raspoloživ kada nije potreban, to nema značaja.
- Integritet podataka
  - Pojam integritet se u kontekstu baza podataka odnosi na preciznost, punovažnost i korektnost podataka u bazi.
  - Održavanje integriteta podataka je od velike važnosti za SUBP.
  - Ukoliko jedan ili više korisnika u isto vreme pokušava da pristupi i promeni podatke u jednoj tabeli, može doći do preplitanja akcija i mogućnosti nekonzistentne promene ili netačnosti podataka.
  - U cilju rešavanja ovog problema, SUBP podržava korišćenje transkacija kako bi se obezbedilo konkurentno izvođenje akcija kao i oporavak baze podataka u slučaju greške.
  - Transakcija je niz operacija nad bazom podataka koji odgovara jednoj ,,logičkoj jedinici posla".
  - Na primer, ako u nekom bankarskom informacionom sistemu korisnik želi da prebaci novac sa jednog računa na drugi, tada će se ova ,,logička jedinica posla" obaviti sledećim operacijama:
    - Učitaj $sumaNovcaP$ za prenos sa jednog na drugi račun;

    - Pronađi račun Rsa sa kog se skida $sumaNovcaP$;
    - Na račun Rsa upiši $sumaNovcaRsa - sumaNovcaP$;
    - Pronađi račun Rna na koji se prebacuje $sumaNovcaP$;
    - Na račun Rna upiši $sumaNovcaRna + sumaNovcaP$;
  - Korisnici mogu da posmatraju izvođenje svake transakcije kao atomske. U transakciji će se sve akcije realizovati u celini, ili neće ni jedna. Korisnici ne treba da brinu ni o efektima nekompletnih transakcija (npr. u slučaju pada sistema).
  - Ukoliko više korisnika istovremeno (konkurentno) zahteva pristup istim podacima, ovi zahtevi mogu biti konfliktni pa ih SUBP mora razrešiti kako bi sprečio narušavanje integriteta baze.
  - Prema tome, osnovna svojstva transakcije su atomičnost, trajnost, izolovanost, mogućnost isprepletanog izvršavanja.
  - Svaka transakcija mora da štiti konzistentnost baze podataka.
  - Ako je transakcija konzistentna, i ako je baza podataka bila u konzistentnom stanju, ona će ostati u tom stanju i nakon završetka transakcije.
  - Konzistentnost transakcije je odgovornost korisnika, a konzistentnost baze nakon završetka konzistentne transakcije je odgovornost SUBP.

- Zaštita podataka
  - Kako bi se baza podataka bezbedno i efikasno koristila, svaki korisnik mora imati autorizaciju za pristup SUBP, odnosno, svakom korisniku mora da bude dodeljeno korisničko ime kojim se predstavlja sistemu.
  - Pored ovog prvog nivoa sigurnosti baze, SUBP obično podržava i više nivoe bezbednosti.
  - Tako se korisnici ili grupe korisnika mogu razlikovati prema nivou autorizacije, odnosno, može im se omogućiti da obavljaju različite opšte zadatke nad bazama podataka, kao što su povezivanje sa bazom, kreiranje tabela ili administriranje sistema.
  - S druge strane, različitim korisnicima ili grupama korisnika se mogu dodeliti različita prava ili privilegije pristupa i izvršavanja specifičnih operacija nad specifičnim objektima u bazi. Tako na primer, da bi neki korisnik mogao da menja podatke u nekoj tabeli, on mora da ima pravo (koje mu je dodeljeno eksplicitno ili implicitno) za promenu podataka u toj tabeli.

- Nezavisnost od aplikacija
  - skrivenost fizičke strukture
    - ako dođe do promena u fizičkoj strukturi baze, to neće zahtevati promene u aplikacijama
  - skrivenost logičke strukture
    - ako dođe do promena u logičkoj strukturi baze, to neće zahtevati promene u aplikacijama

1.1 Nivoi apstrakcije

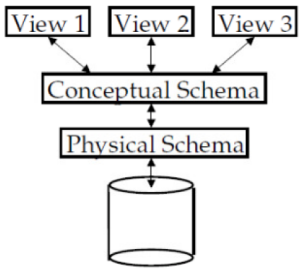
U opštem slučaju razlikujemo 3 nivoa apstrakcije podataka u okviru SUBP [13]:

- Fizička shema. Opisuje datoteke i indekse koji su korišćeni pri implementaciji na nekom fizičkom uređaju. Ovaj nivo je najbliži fizičkoj reprezentaciji baze podataka, koja u računarskom sistemu jedina zaista postoji.

- Konceptualna (logička) shema. Na ovom ,,srednjem“ nivou apstrakcije definiše se logička struktura baze ili način na koji se podaci iz fizičke baze podataka predstavljaju korisniku u opštem slučaju. To znači da se na konceptualnom nivou mogu ,,videti“ svi podaci iz fizičke baze podataka, ali na način pogodniji za korisnika (na višem nivou apstrakcije).

- Spoljašnja shema. Ovo je najviši nivo apstrakcije koji predstavu o podacima iz baze prilagođava potrebama korisnika ili grupe korisnika. Ova shema dakle opisuje kako korisnici vide podatke.

Korisnik svoje zahteve nad bazom definiše na konceptualnom ili spoljašnjem nivou, i to interaktivno, na posebnom ,,jeziku podataka“, ili programski, kada je jezik podataka ugnježden u neki drugi pogramski jezik opšte namene, tzv. matični jezik [10].



Slika 1.1: Različiti nivoi apstrakcije kod SUBP

Logičko i fizičko projektovanje baze

Struktuiranje i organizacija imaju poseban značaj u radu sa bazama podataka. Podatke je na nivou korisnika potrebno struktuirati, a na fizičkom nivou organizovati i to tako da njihovo održavanje bude najlakše, a operisanje njima najefikasnije.

- *Logičko projektovanje baze* podatka predstavlja skup postupaka kojima se dolazi do dobro struktuiranih podataka (pravilno grupisanih atributa) u bazi podataka

- *Fizičko projektovanje baze* podatka predstavlja skup postupaka kojima se podaci fizički organizuju u bazi, tako da im je pristup i održavanje najefikasnije.

1.2 Modeli podataka

Reprezentacija koja se nalazi na ,,srednjem“ konceptualnom nivou apstrakcije naziva se *model podataka*. Modelom podataka se predstavlja logička struktura svih podataka u bazi kao i skup svih operacija koje korisnik može izvršiti nad tim podacima.

U razvoju SUBP može se uočiti nekoliko generacija, koje su ili koegzistirale na tržištu ili smenjivale jedna drugu. Fundamentalna razlika između sistema ovih generacija je razlika u modelu podataka, koja se u implementaciji odgovarajućih SUBP reflektuje na efikasnost pristupa podacima i obrade podataka, produktivnost korisnika, funkcionalnost sistema i podršku raznovrsnim aplikacijama. Tako se u prve dve generacije svrstavaju mrežni i hijerarhijski sistemi, koji su osamdesetih godina prošlog veka gotovo u potpunosti prevaziđeni relacionom tehnologijom, kao trećom generacijom SUBP. Sve tri generacije namenjene su pre svega poslovno-orijentisanim aplikacijama [10].

Neki od najvažnijih modela podataka su:

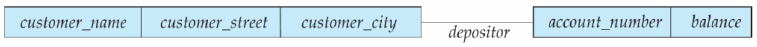
- mrežni model,
- hijerarhijski model,
- relacioni model,
- model entiteta i odnosa,
- objektni ,,model“.

1.2.1 Mrežni model podataka

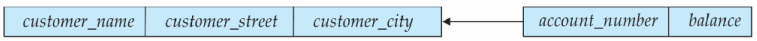
- Prvi sistem za upravljanje bazama podataka projektovao je Čarls Bekman (Charles Bachman) u kompaniji General Electic ranih 1960-ih godina. Taj sistem je nazvan Integrated Data Store. On je predstavljao osnovu mrežnog modela podataka koji je standardizovan od strane Conference on Data Systems Languages (CODASYL) i imao je jak uticaj na razvoj baza podataka u to vreme.
- Prvi standard na području baze podataka je izdat od strane grupe za standardizaciju CODASYL DBTG (Data Base Task Group Conference On DAta SYstems Languages) 1971. godine. Ista grupa je ranije standardizovala COBOL.
- Zadnji važeći standard mrežnog modela datira iz 1978. godine.
- Predlog iz 1981. nikada nije službeno prihvaćen.
- 1973. godine Čarls Bekman je dobio ACM-ovu Tjuringovu nagradu (u računarskim naukama to je ekvivalent Nobelovoj nagradi) za rad u oblasti baza podataka. Bio je prvi dobitnik ove prestižne nagrade.

Koncepti mrežnog modela podataka [14]

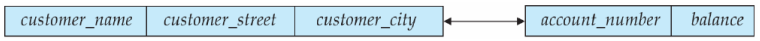
- Strukture podataka su nalik na slogove u programskim jezicima. Slog sadrži podatke jedne pojave entiteta i sastoji se od polja koji odgovaraju atributima. Svako polje sadrži jednu vrednost atributa.
- Strukture podataka se u mrežnom modelu povezuju „vezama“ (link), slično pokazivačima u prgoramskim jezicima. Ovo je nalik na dijagramsko povezivanje podataka.
- Slogovi se označavaju pravougaonicima, a veze linijama između dva sloga.
- Tipovi veza su:
  - više-više



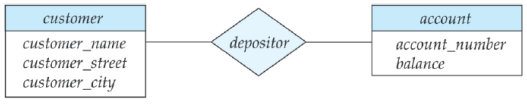
  - jedan-više



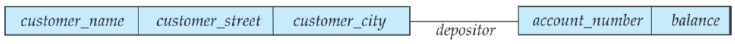
  - jedan-jedan



- *Primer 1*: U ovom primeru povezujemo klijenta i bankovni račun (videti sliku 1.1). Za svaki ER dijagaram (biće kasnije više reči o ER notaciji), postoji odgovarajući dijagram mrežnog modela. Tip entiteta zamenjujemo slogom.



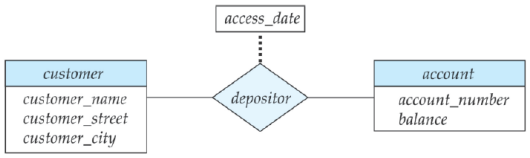
(a) E-R diagram



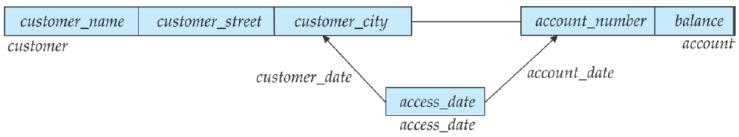
(b) Data structure-diagram

Slika 1.2: Veze bez atributa

- *Primer 2*: Dijagaram iz prvog primera dopunjavamo poslednjim datumom korišćenja.



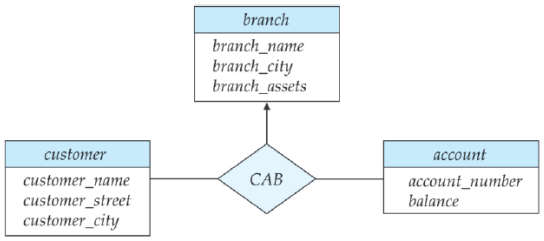
(a) E-R diagram



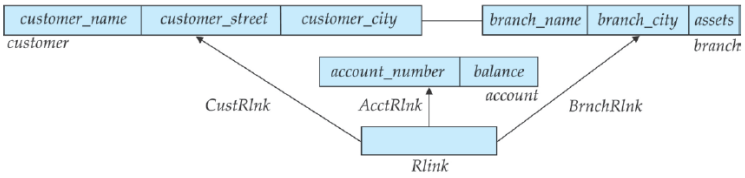
(b) Network diagram

Slika 1.3: Veze sa atributima

- *Primer 3*: Ternarna veza – povezujemo klijenta, bankovni račun i filijalu. Uvodimo novi (prazan, dummy) slog iz kog vode veze ka sva tri sloga.



(a) E-R diagram



(b) Data structure diagram

- Standard CODASYL DBTG (1971) propisuje samo veze tipa više-jedan. Veze tipa više-više nisu dopuštene zbog jednostavnosti implementacije modela. Veze jedan-jedan se implementiraju kao dve veze više-jedan.

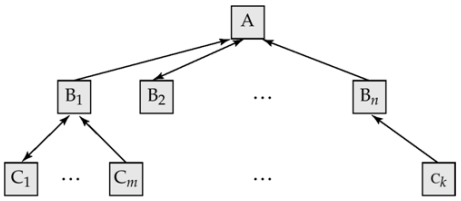
- Za implementaciju u računaru koriste se usmereni grafovi.

1.2.2 Hijerarhijski model podataka

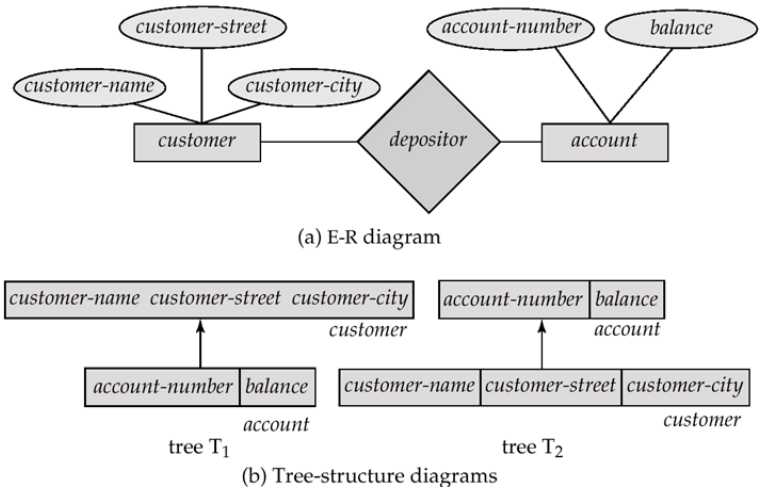
U kasnim 1960-im godinama IBM je razvio Information Management System (IMS) SUBP. IMS je predstavljao osnovu za alternativni hijerarhijski model podatka. Inicijalno IMS je razvijen kao SUBP za svemirski program Apolo. SABRE sistem za izradu avio rezervacija je razvijen od strane American Airlines i IBM-a u isto vreme i on je imao mogućnost da više ljudi pristupi istim podacima preko interneta.

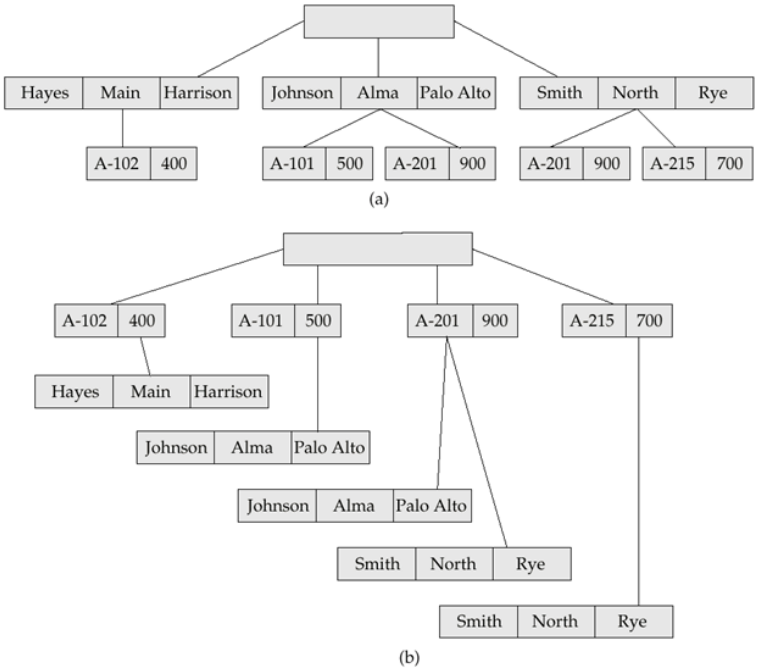
Koncepti hijerarhijskog modela podataka [14]

- Skup ,,slogova“ povezanih ,,vezama“ tako da grade ,,hijerarhiju“
  - u osnovi slično mrežnom modelu, s tim da se zahteva hijerarhija
- Skup slogova u kolekciji započinje ,,praznim“ (dummy) čvorom
- *Primer 3.* Primer podataka u hijerarhijskom modelu. Roditelj može da nema decu, može imati jedno dete ili više dece, ali dete mora imati tačno jednog roditelja.

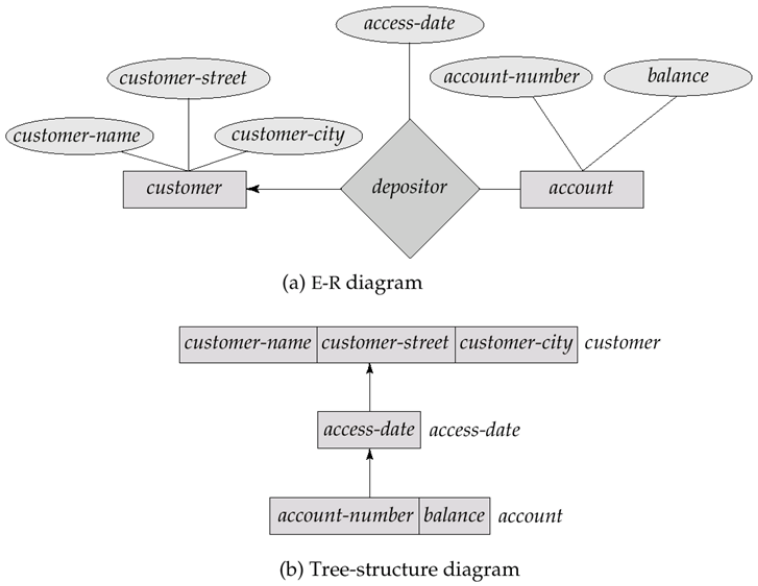


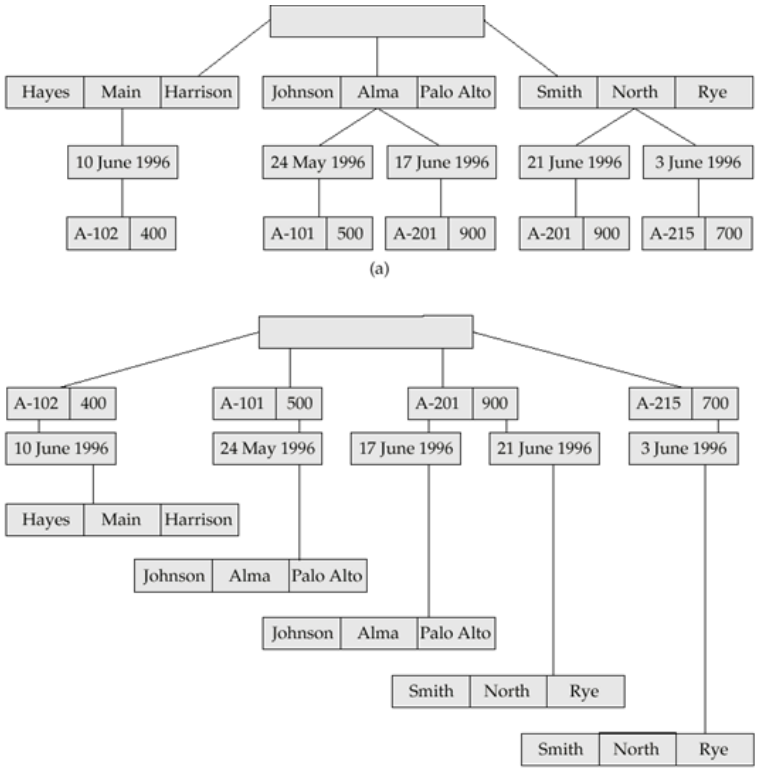
- Složeni odnosi mogu da zahtevaju više hijerarhija. Različiti načini pristupanja takođe zahtevaju više hijerarhija. Posledica svega ovoga je uvećana redundantnost podataka.
- *Primer 4.* Primer veze bez atributa.





- *Primer 5.* Primer veze sa atributima. Umećemo i datum pristupa između dva entiteta.





1.2.3 Relacioni model podataka

- Autor relacionog modela podataka je Edgar Codd, 1970.
- Objedinjeno modeliranje:
  - I entiteti i odnosi se modeliraju relacijom.
  - Relacija se sastoji od atributa koji imaju imena i domene.
  - Skup imena i domena atributa jedne relacije predstavlja shemu relacije.
  - Torka je skup imenovanih vrednosti.
  - Torka koja ima isti broj vrednosti, nazive i domene tih vrednosti kao odgovarajući atributi neke relacije, predstavlja instancu domena (sheme) te relacije.
  - Vrednost (sadržaj) relacije je skup instanci njenog domena.
  - ...
- Integritet se obezbeđuje ključevima
- ...

1.2.4 Model entiteta i odnosa

Sistemi zasnovani na tradicionalnim modelima kao što su hijerarhijski, mrežni i relacioni, imaju tendenciju da u značajnoj meri razgraničavaju unutrašnji

(fizički) nivo podataka od konceptualnog (logičkog) i spoljašnjeg, ali ne u dovoljnoj meri i spoljašnji nivo od konceptualnog. Osim toga, ovi modeli ne podržavaju semantiku podataka, već je ona prepuštena korisniku. Tako, na primer, relacioni sistemi ,,ne prepoznaju“ da su ,,naslov knjige“ i ,,država“ semantički različiti atributi, pa je moguće, ako su definisani na istom domenu (niske simbola), izvršiti operaciju poređenja nad njima. Ovi modeli ,,ne prepoznaju“ ni da je, na primer, tip entiteta "programer" specijalni slučaj tipa entiteta ,,radnik“, a da je ovaj specijalni slučaj tipa entiteta ,,osoba“. Tako se dolazi do potrebe za proširenjem postojećih modela semantičkim aspektom podataka, tj. do stvaranja novih modela koji uključuju semantičku komponentu. Predstavljanje značenja podataka modelom podataka zove se semantičko modeliranje. [10]

- Model entiteta i odnosa je definisao Peter Chen 1976. godine
- U ovom modelu se podaci modeliraju skupovima entiteta
  - ,,entiteti“ su podaci koji postoje nezavisno od drugih entiteta u bazi podataka
  - entiteti su nalik na slogove, imaju ,,atribute“
  - jedan ili više atributa može da predstavlja ,,ključ“
  - među entitetima mogu da postoje ,,odnosi“
  - ,,odnosi“ mogu da budu 1-1, 1-N, N-1 ili M-N
  - i odnosi mogu da imaju atribute
- Model entiteta i odnosa je primer jednog semantičkog modela. Doživeo je veći broj različitih proširenja imodifikacija, tako da je i danas u upotrebi veći broj verzija tog modela. Uz model je razvijena i dijagramska tehnika za predstavljanje entiteta i odnosa među njima, sa svim njihovim karakteristikama.
- Posebnu primenu model entiteta i odnosa ima u realizaciji efikasne i produktivne semantičke metode logičkog projektovanja. Nije zaživeo kao poseban model podataka, ali jeste kao alat za modeliranje relacionih baza podataka.
- Shema baze podataka koja se dobije ovom metodom obično se (automatski) preslikava u shemu relacione baze podataka, na koju se primenjuje neki od relacionih jezika za definisanje i manipulisanje podacima. Ako je proces logičkog projektovanja zasnovan na modelu entiteta i odnosa korektno sproveden, tj. ako su uočeni svi bitni entiteti i pravilno uspostavljeni odnosimeđu njima, onda dobijena relaciona baza podataka ima svojstvo da su joj, gotovo uvek, sve relacije u nekoj od viših normalnih formi (bar u 3NF). [10]

1.2.5 ,,Objektni“ model podataka

Relacione baze podataka su se pokazale veoma uspešnim u obradi poslovnih podataka i aplikacija. Ipak, s obzirom na ograničen broj tipova podatka (integer, date, string, itd.) koji su pogodni za tradicionalne sisteme koji obuhvataju dominantno alfanumeričke (tekstualno orijentisane) podatke i s obzirom na zahtev

za normalizovanošću relacija, one se pokazuju kao neadekvatan izbor za široke klase aplikacija koje manipulišu podacima kompleksnije strukture. Zbog toga proizvođači relacionih sistema proširuju svoje proizvode novim funkcionalnostima, kao što su:

- podrška bogatijem skupu tipova i pravila,
- nasleđivanje atributa u odnosima tipa generalizacija/specijalizacija,
- učaurenje podataka i operacija

Ovakvim proširenjima relacioni sistemi izlaze iz okvira relacionog modela, i prerastaju u sisteme novije generacije [10].

Kako neke od pomenutih funkcija kojima se proširuju relacioni sistemi predstavljaju sastavni deo programske paradigme poznate kao objektno-orijentisana paradigma, to je jedan pravac razvoja oblasti baza podataka definisanje novog modela podataka, objektno orijentisanog modela, i implementacija sistema za upravljanje bazama podataka nad tim modelom.

Za razliku od relacionog modela, koji je precizno definisan, ne postoji jedinstven objektno orijentisani model podataka, već su objektni sistemi dosta šaroliki. Tako, u odsustvu jedinstvenog objektno orijentisanog modela podataka, pogodno je identifikovati minimalni skup objektnih koncepata koje podržava svaki objektni sistem (jezgro OO modela podataka) i prepoznati moguće koncepte nadogradnje tog minimalnog skupa.

Jezgro objektno orijentisanog modela podataka, sa aspekta baza podataka, predstavljaju sledeći koncepti [10]:

- objekat,
- klasa,
- poruka i metoda,
- učaurenje ili enkapsulacija i
- nasleđivanje.

1.3 Evolucija zahteva i tehnika

Količina podataka iz dana u dan raste velikom brzinom. Smatra se da se svakih devet meseci količina podataka duplira [8]. Samim tim, javlja se stalna potreba za proširivanjem baze, što dovodi do otežanog pristupa podacima, dok potreba za brzim dobijanjem odgovara na pitanja i dalje ostaje. Dakle, neophodne su efikasnije tehnike za čuvanje i obradu podataka. Administrator baze podataka mora da ima slobodu da promeni fizičku reprezentaciju ili pristupne tehnike radi boljih performansi bez promena postojećih aplikacija. Da bi se obezbedile visoke performanse neophodno je vršiti manuelnu i automatsku optimizaciju. Takođe, baza treba da bude sposobna da se širi i da bude skalabilna. Skalabilnost je sposobnost sistema da podnese povećanje zahteva i broja korisnika na predvidiv način, tako da sistem ne postane previše kompleksan, skup i nepraktičan. U tom smislu više na značaju dobijaju distribuirane baze podataka — baze kod kojih se njihovi fizički delovi nalaze na različitim čvorovima mreže. Bitna je i mogućnost integracije sa drugim izvorima podataka.

1.4 Nerelacione baze

- U savremenom razvoju softvera termin nerelacione baze podataka se odnosi na sisteme za upravljanje kolekcijama podataka koje imaju sledeća svojstva:
  - nisu relacione,
  - distribuirane su,
  - otvorenog su koda i
  - horizontalno skalabilne.
- Često imaju i sledeće karakteristike:
  - nemaju strogu statičku strukturu podataka,
  - jednostavna podrška replikaciji,
  - jednostavni API,
  - ogromne količine podataka,
  - ...
- Naziv NoSql nastao je 1998. godine upravo zato što je Sql simbol relacionih baza podataka.
- Bilo je pokušaja da se zovu i NonRel baze, ali je NoSql preovladao.
- Danas se tumači kao ,,Not only Sql“. Bez obzira na bukvalno značenje, NoSql se koristi danas kao opšti termin za sve baze podataka i skladišta podataka koje ne prate popularne i dobro-utvrđene principe relacionih baza podataka.
- Često se odnose na velike skupove podataka kojima je potrebno brzo i efikasno pristupiti i menjati ih na vebu.
- Vrste NoSql baza podataka prema modelu podataka
  - Ključ/vrednost baze podataka
  - Baze podataka orijentisane ka dokumentima
  - Uređeno sortirana skladišta orijentisana ka kolonama
  - Grafovske baze podataka
  - Više o tome kasnije...

2

Arhitektura i projektovanje baza podataka

Životni ciklus baze podataka se može podeliti u sledeći niz koraka (slično kao kod razvoja softvera):

- prikupljanje i analiza zahteva,
- logički dizajn podataka,
- fizički dizajn podataka,
- projektovanje implementacije (distribucije),
- implementacija,
- testiranje,
- održavanje.

U poslednjih desetak godina dolazi do intenzivnih promena koje se dešavaju u poslovnom okruženju organizacija što utiče na promene:

- u životnom ciklusu i
- u arhitekturi.

Arhitektura sistema baza podataka je apstraktni opis njegovih komponenti i njihovih interakcija [8].

- Vrši se identifikacija komponenti sistema.
- Specifikuju se funkcije svake komponente.
- Definišu se međusobni odnosi komponenti.
- Definišu se interakcije između komponenti.

2.1 Standardizacija arhitekture

Arhitektura baza podataka predstavlja jedan od predmeta standardizacije u oblasti baza podataka, zbog tesne veze između arhitekture sistema i njegovog referentnog modela. Referentni model predstavlja konceptualni okvir čija je svrha podeliti rad na standardizaciji na manje (upravljive) delove i pokazati kako se ti delovi međusobno povezuju. Referentni model je zapravo idealizovani model arhitekture sistema.

Referentni model i arhitektura se mogu posmatrati iz tri ugla:

- ugla komponenti,
- ugla funkcija i
- ugla podataka.

2.1.1 Komponente sistema

Podsetimo se, osnovne komponente sistema baza podataka su [8]:

- podaci (integrisani i deljivi),
- hardver (spoljašnji memorijski uređaji, procesori i glavna memorija),
- softver (SUBP, alati za razvoj aplikacija, pomoćni programi,...) i
- korisnici (aplikativni programeri, krajnji korisnici, administratori).

Komponente sistema se definišu zajedno sa njihovim međusobnim odnosima.

- SUBP se sastoji od skupa komponenti, koje obavljaju određene funkcije.
- Putem definisanih interakcija komponenti ostvaruje se puna funkcionalnost sistema.

Ovakav pristup je neophodan pri implementaciji. Nije dovoljno posmatrati samo komponente da bi se odredila funkcionalnost sistema kao celine.

2.1.2 Funkcije sistema

- Prepoznaju se različite klase korisnika i funkcije koje sistem za njih obavlja.
- Uobičajeno se prave hijerarhije korisnika.
- Prednost pristupa je u jasnoći predstavljanja funkcija i ciljeva sistema.
- Slabost je u nedovoljnom uvidu u način ostvarivanja funckija i ciljeva.

2.1.3 Podaci sistema

- Prepoznaju se različite vrste podataka.
- Arhitektura se određuje tako da definiše funkcionalne jedinice koje koriste podatke na različite načine.
- Kako su podaci centralna tema SUBP-a, ovaj pristup se često preporučuje kao najpoželjniji.
- Prednost je u isticanju centralne pozicije podataka u sistemu.
- Slabost je u nemogućnosti da se arhitektura u potpunosti odredi ako nisu opisane i funkcionalne celine.

2.1.4 Objedinjavanje pristupa

Svi pristupi se moraju kombinovati kako bi se dobio model arhitekture koji u svakoj posmatranoj tački daje dovoljno informacija o odgovarajućim aspektima. Svaki pristup donosi različit pogled i služi za fokusiranje na različite aspekte modela.

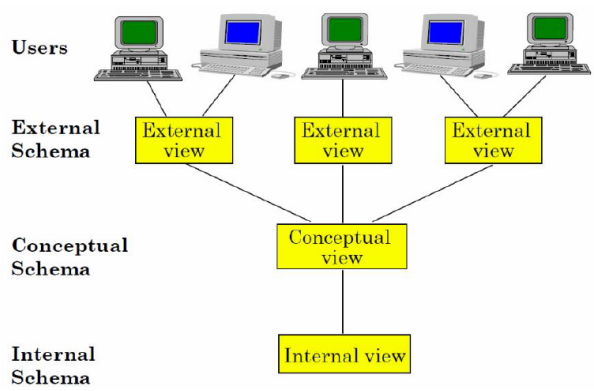
2.2 ANSI/SPARC arhitektura

Jedan od najvažnijih standarda baza podataka je nastao 1977. godine

- nakon 5 godina rada
- ANSI/SPARC (American National Standards Institute/Standard Planning and Requirements Committee)
- poznat kao ,,ANSI/SPARC arhitektura“.
- načelno počiva na podacima ali zapravo je objedinjen pogled

U okviru ove arhitekture prepoznaju se tri nivoa (pogleda) podataka:

- spoljašnji nivo
  - kako korisnici (uključujući programere) vide podatke
  - može biti više domena – svaki obuhvata samo podatke značajne jednoj klasi korisnika
- konceptualni nivo
  - kako podaci čine celinu iz ugla poslovnog okruženja
  - apstraktna definicija baze podataka
- interni (unutrašnji) nivo
  - kako su podaci implementirani na računaru/računarima
  - obuhvata elemente fizičke implementacije na konkretnom hardveru



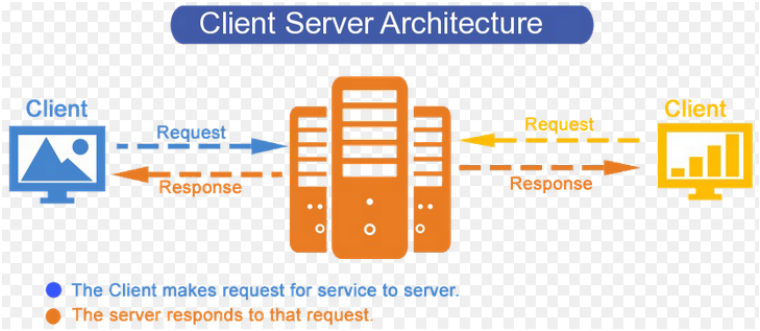
Slika 2.1: ANSI/SPARC arhitektura

2.2.1 Relacione baze i ANSI/SPARC

- Na primeru relacionih baza podataka nivoi se mogu (okvirno) predstaviti na sledeći način:
  - eksterni nivo čine pogledi koji pružaju denormalizovanu sliku dela domena
  - konceptualni nivo čini shema baze podataka
    - obuhvata opise atributa, ključeva i ograničenja, uključujući i strane ključeve
  - interni nivo čine fizički aspekti
    - indeksi
    - prostori za tabele
    - razne optimizacije
- Primer: Baza podataka univerziteta
  - Konceptualni nivo (konceptualna shema)
    - `Studenti(sid:string, ime:string, prezime:string, godine:integer, prosek:real)`
    - `Kursevi(kid:string, kime:string, kESPB:integer)`
    - `Upisao(sid:string, kid:string, godina:integer)`
  - Interni nivo (fizička shema)
    - Relacije su smeštene kao neuređene datoteke
    - Indeks je postavljen na prvu kolonu relacije Studenti
  - Eksterni nivo (spoljašnja shema, pogledi)
    - `Informacije_o_kursu(kid:string, kime:string, upisali:string)`

2.3 Arhitektura klijent/server

Osnovna ideja kod ove arhitekture je relativno jednostavna: Razdvojiti funkcionalnosti servera i klijenta. Na serveru (prvobitno u engleskoj terminologiji backend, pozadinski), je smeštena baza podataka i SUBP, i on je odgovoran za definisanje podataka, manipulisanje podacima, bezbednost, integritet, oporavak i drugo. Na klijentu (prvobitno je korišćen engleski termin frontend, prednji sistem), je smeštena aplikacija, i on je odgovoran za rad te aplikacije. Ovakva podela posla ne zahteva nužno razdvajanje odgovornosti na različite računare. Ipak, termin ,,klijent/server“ se danas gotovo uvek odnosi na slučaj kada su klijent i server na odvojenim računarima. Osnovni klijent/server sistemi podrazumevaju veći broj klijenata koji dele isti server. Međutim, moguće je da jedan klijent pristupi većem broju servera, pri čemu korisnik mora da zna na kom serveru se nalaze koji podaci (nije ostvarena nevidljivost lokacije). Zbog svega toga, često se kaže da je klijent/server arhitektura specijalan slučaj distribuirane arhitekture [10].



Slika 2.2: Klijent/server arhitektura

2.4 Distribuirane arhitekture

Neformalno, distribuirana baza je baza podataka koje se ne nalazi na jednoj fizičkoj lokaciji (na jednom računaru), već je razdeljena na više lokacija povezanih komunikacionom mrežom. Svaka lokacija (čvor komunikacione mreže) poseduje autonomni SUBP, sa sopstvenom kontrolom, upravljačem transakcija, oporavka od pada i ostalim značajnim funkcijama. Osnovna pretpostavka kod distribuiranih sistema je nevidljivost lokacije, što znači da ovaj sistem treba da obezbedi još jedan nivo fizičke nezavisnosti podataka jer korisnik ne treba da zna na kojoj lokaciji u distribuiranom sistemu se nalaze podaci koji mu trebaju [10].

DSUBP može da bude:

- homogen, kada su SUBP na svim lokacijama isti i
- heterogen, kada lokalni SUBP mogu da budu različiti.

Prednosti koje donosi distribuirani koncept baze podataka su:

- lokalna autonomija podataka, upravljanja i kontrole,

- veći kapacitet i postupni rast,
- pouzdanost i raspoloživost,
- efikasnost i fleksibilnost.

Radi postizanja efikasnosti, sve komponente DSUBP treba realizovati tako da se što više smanji mrežna komunikacija. Najznačajniji problemi koje tom prilikom treba rešiti su:

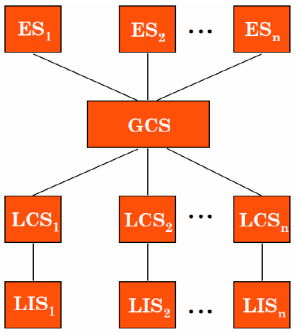
- fragmentacija podataka,
- distribuirana obrada upita,
- distribuirano (preneto) ažuriranje,
- upravljanje katalogom,
- distribuirano izvršenje skupa transakcija, što uključuje konkurentnost, integritet, oporavak i protokole kompletiranja transakcija.

Primeri distribuiranih arhitetkura:

- Arhitektura ravnopravnih čvorova
- Federativne baze podataka

Arhitektura ravnopravnih čvorova

- Svaki od čvorova može imati sopstvenu fizičku organizaciju podataka, koja se naziva lokalna interna shema (LIS, local internal schema)
- Poslovno viđenje tih podataka na konceptualnom nivou je opisano globalnom konceptualnom shemom (GCS – global conceptual schema)
- Zbog fragmentacije i replikacije podataka, na svakom čvoru je potrebno da postoji logički opis podataka, koji se naziva lokalna konceptualna shema (LCS – local conceptual schema)
  - globalna konceptualna shema je praktično unija svih lokalnih konceptualnih shema
- korisnici na spoljašnjem nivou imaju odgovarajuće spoljašnje sheme (ES – external schema)

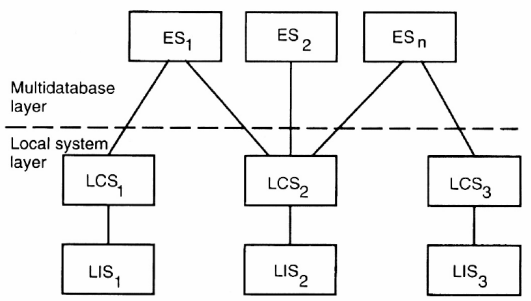


Slika 2.3: Arhitektura ravnopravnih čvorova

- Ova arhitektura je prirodno proširenje ANSI/SPARC modela

Federativne baze podataka

- Federativne (višestruke) baze podataka predstavljaju virtuelnu integraciju različitih baza
- Ove baze ne koriste globalnu konceptualnu shemu
  - Mogu da koriste tzv. izvozne sheme, koje za svaki čvor posebno opisuju koje podatke čvor želi da deli.
  - Tada je globalna baza podataka unija izvoznih shema.
  - Svaka aplikacija koja koristi globalnu bazu podataka to radi posredstvom odgovarajuće uvozne sheme.



Slika 2.4: Federativne (i višebazne) arhitekture

3

Relacioni model baza podataka

Svaki model podataka može da se posmatra kao kolekcija sledeća tri skupa:

- skup tipova objekata,
- skup operacija nad objektima definisanih tipova i
- skup pravila integriteta.

Prema tome, relacioni model podataka može da se definiše preko tri bitne komponente modela:

- *Strukturni deo relacionog modela*, koji opisuje način modeliranja podataka;
- *Manipulativni deo relacionog modela*, koji opisuje način rukovanja modeliranim podacima;
- *Integritetni deo relacionog modela*, koji opisuje način obezbeđivanja valjanosti podataka.

U relacionom modelu baza podataka relacija modelira sav sadržaj baze podataka — i entitete i odnose.

3.1 Strukturni deo relacionog modela

Dva osnovna tipa objekata koji karakterišu strukturu relacionog modela su:

- *domen* i
- *relacija*.

Domen predstavlja skup vrednosti istog tipa (na primer, skup naslova knjiga, skup imena gradova ili skup datuma nekih događaja). Neka je $D_1 \times D_2 \times \cdots \times D_n$ Dekartov proizvod domena D_1, D_2, ..., D_n. Tada *relacija* R *stepena* n (n-arna relacija), definisana na domenima D_1, D_2, ..., D_n predstavlja proizvoljni konačni podskup navedenog Dekartovog proizvoda. Za razliku od matematičkog pojma relacije, ova relacija je dinamičkog sadržaja jer se neke njene n-torke tokom „života“ relacije brišu, neke se dodaju a neke se menjaju.

Matematičke osnove

Relacioni model je u potpunosti formalno matematički zasnovan. To je neprotivrečan i nedvosmislen model. Ovde ćemo navesti samo okvire te formalizacije u širini i dubini koje omogućavaju dalji rad.

Model relacije

Svakoj matematičkoj relaciji odgovara tačno jedan skup objekata koji zadovoljavaju relaciju. Taj skup se često upotrebljava kao matematički model relacije.

Ako je ρ n-arna relacija i njen domen označen sa $Dom(\rho)$, onda je jedan model relacije ρ skup:

$$model(\rho) = \{(a_1, a_2, ..., a_n) | (a_1, a_2, ..., a_n) \in Dom(\rho) \wedge \rho(a_1, a_2, ..., a_n)\}$$

Primetimo da važi:

$$model(\rho) \subset Dom(\rho)$$

Predstavljanje relacija skupovima

Za sve n-torke iz $Dom(\rho)$ važi:

$$\rho(a_1, a_2, ..., a_n) \iff (a_1, a_2, ..., a_n) \in model(\rho)$$

Zbog navedene ekvivalencije često se umesto oznake $model(\rho)$ upotrebljava samo relacija ρ u istom kontekstu:

$$\rho = \{(a_1, a_2, \ldots, a_n) | (a_1, a_2, \ldots, a_n) \in Dom(\rho) \wedge \rho(a_1, a_2, \ldots, a_n)\}$$

Relacije često predstavljamo upravo pomoću njihovog modela.

Predstavljanje relacija skupovima — primer

Neka imamo:

- crvenu, plavu, belu i zelenu kutiju
- i u njima lopte, kocke i pločice.

Neka je data binarna relacija $kutijaSadrzi(K, P)$ koja je zadovoljena ako kutija K sadrži P. Njen domen je tada definisan na sledeći način:

$$Dom(kutijaSadrzi) = \{crvena, plava, bela, zelena\} \times \{lopte, kocke, plocice\}$$

Neka naredni model relacije opisuje šta se nalazi u kojoj kutiji:

$$kutijaSadrzi = \{(plava, lopte), (plava, kocke), (zelena, plocice), (crvena, kocke)\}$$

Tada:

- iskaz $kutijaSadrzi(plava, kocke)$ je tačan,
- iskaz $kutijaSadrzi(zelena, lopte)$ nije tačan,
- iskazi $kutijaSadrzi(zuta, kocke)$ i $kutijaSadrzi(zelena, knjige)$ nisu definisani, zato što argumenti nisu u domenu relacije.

Relacije (odnosno njihovi modeli) sa konačnim brojem elemenata mogu da se predstave i pomoću tabela:

kutijaSadrži	
BOJA	**SADRŽAJ**
plava	lopte
plava	kocke
zelena	pločice
crvena	kocke

Beskonačne relacije i skupovi

Relacije sa beskonačnim brojem elemenata možemo da predstavljamo formalnim zapisivanjem skupa koji predstavlja model relacije. Na primer, relaciju *neparan* možemo da predstavimo, pa i da definišemo, skupom:

$$neparan = \{n | n \in N \wedge (\exists k \in N)(n = 2k - 1)\}$$

Međutim, u oblasti baza podataka beskonačne relacije nisu posebno zanimljive, tako da ćemo se nadalje fokusirati samo na konačne relacije.

Entiteti i atributi

Entitetima nazivamo neke objekte „stvarnog“ sveta koje modeliramo i opisujemo nekim skupom podataka. Neka je E neki *skup entiteta*. Tada kažemo da se skup entiteta karakteriše konačnim skupom atributa $A_1, A_2, \ldots A_n$, u oznaci $E(A_1, A_2, \ldots A_n)$, akko:

- Svaki atribut A_i predstavlja funkciju koja slika entitete u odgovarajući domen atributa D_i:
$$A_i : E \to D_i$$
- Svaki atribut A_i i ima jedinstven naziv t_i

Za svaki entitet $e \in E$, vrednost funkcije $A_i(e) \in D_i$ predstavlja njegovu vrednost atributa A_i.

Entiteti i atributi — primer

Neka je E skup radnika koji se karakteriše atributima:

- *ime* (niska od 20 znakova),
- *prezime* (niska od 25 znakova),
- *osnova_plate* (broj oblika 7.2).

IME	PREZIME	OSNOVA_PLATE
Dragana	Pantić	45000
Marko	Marković	40000
Dragana	Pantić	45000
Jelena	Popović	43000
Dragiša	Đukić	47000

Napomena: Primetimo da za različite entitete možemo da dobijemo jednake slike.

3.1.1 Modeliranje entiteta relacijama

U opštem slučaju, skup svih atributa $A_1, \dots A_n$ entiteta $e \in E$ određuje funkciju α na sledeći način:

$$\alpha(e) = (A_1(e), \dots A_n(e))$$

Slika skupa entiteta funkcijom α je skup:

$$R = \alpha(E)$$

Podsetimo se, funkcija α je injektivna akko za sve različite originale daje različite slike, tj. akko važi:

$$\alpha(e) = \alpha(u) \; akko \; e = u$$

Kažemo da skup atributa *dobro karakteriše* skup entiteta ako, pored navedenih uslova, funkcija predstavlja injektivno preslikavanje.

Ako je α injektivna funkcija, tada kažemo da je slika $R = \alpha(E)$:

- relacija R sa atributima $A_1, \dots A_n$,
- domenom relacije $Dom(R) = D_1 \times \dots \times D_n$,
- i nazivima atributa $Kol(R) = (t_1, \dots t_n)$.

Tada skup entiteta E sa atributima $A_1, \dots A_n$ modeliramo relacijom R i relacija R je dobra karakterizacija skupa entiteta E. Formalno, definišemo R tako da je $model(R) = \alpha(E)$.

Zapisivanje

Ako je $\alpha(e) = (a_1, \dots a_n)$, tada n-torku $(a_1, \dots a_n)$, koja formalno predstavlja ,,model entiteta e“, često nazivamo i ,,entitetom e“. Atribut $A_i(e)$ zapisujemo i kao $e.t_i$. Namerno ne pravimo razliku između relacije R i odgovarajućih relacija dobijenih permutovanjem njenih atributa.

Predstavljanje relacije tabelom

Relaciju $R = \alpha(E)$ često predstavljamo, pa i nazivamo, *tabelom*. Pri tome:

- Kolone tabele odgovaraju atributima $A_1, \dots A_n$;
- Nazivi kolona odgovaraju nazivima atributa $t_1, \dots t_n$;
- Vrste tabele odgovaraju torkama relacije, tj. entitetima.

Predstavljanje entiteta – primer

Relacija ne sme da ima jednake elemente:

RADNIK		
IME	**PREZIME**	**OSNOVA_PLATE**
Dragana	Pantić	42000
Marko	Marković	40000
Dragana	Pantić	45000
Jelena	Popović	43000
Dragiša	Đukić	47000

Ako je

$$\alpha(x) = (ime(x), prezime(x), osnova\_plate(x))$$

onda ćemo n-torku $(ime(x), prezime(x), osnova\_plate(x))$ u zapisima obično da izjednačavamo sa x. Pri tome ćemo $ime(x)$ da zapisujemo kao $x.ime$ (i slično za ostale atribute). Nećemo razlikovati relaciju $RADNIK$ od relacija dobijenih permutovanjem atributa ove relacije.

Relaciona baza podataka

Relaciona baza podataka je skup relacija. Opis relacije čine domen relacije i nazivi atributa. Oznaka $R(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$ koristi se za predstavljanje relacije R čiji atribut A_i uzima vrednost iz domena D_i, za $i = 1, \ldots, n$, i zove se *relacijska shema relacije* R. Skup relacijskih shema relacione baze podataka je *shema relacione baze podataka.*

Primer baze podataka – tipovi

Uvedimo opisno sledeće skupove:

- $N(a)$ = „skup svih celih brojeva sa najviše a dekadnih cifara“;
- $N(a, b)$ = „skup svih brojeva koji u dekadnom zapisu sa leve strane decimalne zapete imaju najviše a, a sa desne strane najviše b cifara“
- $C(a)$ = „skup svih niski dužine do a znakova“
- D = „skup svih datuma (u nekom podrazumevanom opsegu)“

Navedeni skupovi se najčešće upotrebljavaju kao domeni (tj. tipovi) atributa relacija. Svi atributi u narednom primeru relacione baze podataka imaju za domen neki od navedenih skupova.

Primer baze podataka – shema

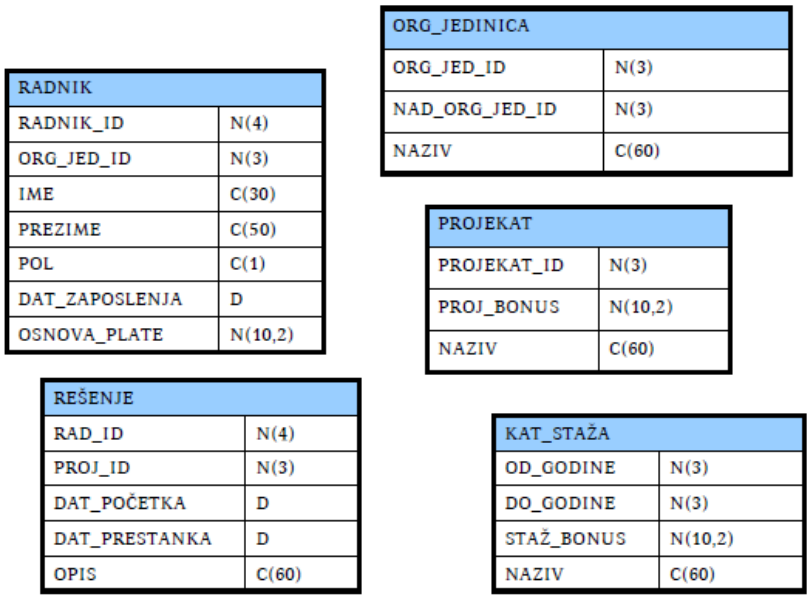
$Kol(ORG\_JEDINICA) = (\text{'}org\_jed\_id\text{'}, \text{'}nad\_org\_jed\_id\text{'}, \text{'}naziv\text{'})$
$Dom(ORG\_JEDINICA) = N(3) \times N(3) \times C(60)$

$Kol(RADNIK) = (\text{'}radnik\_id\text{'}, \text{'}org\_jed\_id\text{'}, \text{'}ime\text{'}, \text{'}prezime\text{'}, \text{'}pol\text{'}, \text{'}dat\_zaposlenja\text{'}, \text{'}osnova\_plate\text{'})$
$Dom(RADNIK) = N(4) \times N(3) \times C(30) \times C(50) \times C(1) \times D \times N(10, 2)$

$Kol(KAT\_STAZA) = ('od\_godine','do\_godine','staz\_bonus','naziv')$
$Dom(KAT\_STAZA) = N(2) \times N(2) \times N(10,2) \times C(60)$

$Kol(RESENJE) = ('rad\_id','proj\_id','dat\_pocetka','dat\_prestanka','opis')$
$Dom(RESENJE) = N(4) \times N(3) \times D \times D \times C(60)$

$Kol(PROJEKAT) = ('projekat\_id','proj\_bonus','naziv')$
$Dom(PROJEKAT) = N(3) \times N(10,2) \times C(60)$



Slika 3.1: Primer baze podataka – shema

RADNIK						
RADNIK_ID	ORG_JED_ID	IME	PREZIME	POL	DAT_ZAPOSLENJA	OSNOVA_PLATE
100	10	Petar	Perić	M	26.03.1991.	42000
101	11	Marko	Marković	M	30.03.1990.	43000
102	-	Lazar	Lazić	M	21.04.1981	28000
103	11	Milica	Milić	Ž	30.06.1979.	45000
104	30	Dragica	Dragić	Ž	10.07.1988.	44000
105	30	Gorica	Bojić	Ž	20.07.2003.	39000
106	30	Vladana	Mladić	Ž	05.01.2007.	29000
107	40	Đurđa	Perić	Ž	29.03.1978.	35000
108	40	Đura	Lazić	M	18.03.1983.	42000

ORG_JEDINICA		
ORG_JED_ID	NAD_ORG_JED_ID	NAZIV
40	-	Softver
30	40	Projektovanje
11	10	Dokumentacija
10	40	Planiranje
50	40	Analiza
60	40	Dizajn i modeliranje
70	40	Kodiranje

KAT_STAZA			
OD_GODINE	DO_GODINE	STAŽ_BONUS	NAZIV
1	5	2000	Početna
6	20	5000	Srednja
21	50	8000	Završna

PROJEKAT		
PROJEKAT_ID	PROJ_BONUS	NAZIV
100	-	Crveni
150	-	LJubičasti
200	-	Plavi
250	1500	Zeleni
300	2000	Žuti
250	1500	Narandžasti
900	-	Crni

REŠENJE				
RAD_ID	PROJ_ID	DAT_POČETKA	DAT_PRESTANKA	OPIS
100	200	01.01.1991.	-	privremeno učešće
101	200	01.01.1991.	-	-
108	200	01.01.1984.	-	-
104	100	01.01.1989.	31.12.1998.	-
105	100	01.01.2004.	-	-
106	100	01.01.2008.	-	-
107	100	01.01.1981.	-	-
100	150	01.01.1999.	31.12.1998.	Koordinator
101	150	01.01.1972.	31.12.1998.	-
102	150	01.01.1972.	31.12.1998.	-
108	150	01.01.1972.	31.12.1998.	-
104	150	01.01.1986.	31.12.1998.	-
105	150	01.01.1998.	31.12.1998.	-
107	150	01.01.1999.	-	Projektant
108	250	01.01.1999.	-	-
108	300	01.01.1999.	-	-
108	350	01.01.1999.	-	-
105	250	01.01.1998.	-	-
107	350	01.01.1998.	-	-

Slika 3.2: Primer baze podataka – sadržaj

3.1.2 Modeliranje odnosa relacijama

U relacionom modelu, odnosi se modeliraju na isti način kao i entiteti – relacijama. Bez mnogo formalizovanja, pogledajmo prethodni primer i videćemo da već imamo neke odnose:

- Svaki *RADNIK* ima dodeljen tačno jednan entitet *ORG_JEDINICA*.

- Svako $RESENJE$ predstavlja odnos tačno jednog entiteta RADNIK i tačno jednog entiteta $PROJEKAT$, uz još neke dodatne atribute.

Osnovna ideja je sasvim jednostavna:

- Ako su dva entiteta $e \in E$ i $f \in F$ u nekom odnosu, onda to možemo opisati novom relacijom $\rho(e, f)$, čiji je domen $Dom(\rho)$
- Ako su entiteti $e \in E$ i $f \in F$ modelirani kao

$$e = (x_1, \ldots, x_n), f = (y_1, \ldots y_m)$$

onda njihov odnos može da se modelira kao

$$\rho(e, f) = (x_1, \ldots, x_n, y_1, \ldots y_m)$$

Ako postoji podskup atributa $A_{i1}, \ldots, A_{ink}$ takav da postoji preslikavanje k, koje za svaki entitet e iz E jednoznačno preslikava izabrani podskup atributa $x_{i1}, \ldots x_{ink}$ entiteta e u kompletan model entiteta $e(x_1, \ldots, x_n)$, onda u modelu relacije ρ taj skup atributa može da se koristi umesto punog skupa atributa $A_1, \ldots, A_n$.

3.2 Manipulativni deo modela

Ključno mesto u manipulativnom delu modela ima pojam upita. *Upit* se može definisati kao definicija nove relacije na osnovu već postojećih i poznatih relacija baze podataka.

3.2.1 Formalni upitni jezici

Definicija relacionog modela uključuje dva formalna jezika za rad sa relacijama:

- Relaciona algebra i
- Relacioni račun

Po pitanju izražajne moći, dokazana je ekvivalentnost ovih formalizama.

Relaciona algebra

Relaciona algebra je skup operacija nad relacijama i relacioni izraz u relacionoj algebri se sastoji od niza operacija nad odgovarajućim relacijama. Relaciona algebra u stvari predstavlja proširenje skupovne algebre (koju čine uobičajene skupovne operacije na relacijama). Osnovne operacije relacione algebre su:

- *projekcija* – izdvajanje podskupa atributa:

$$\{(x.ime, x.prezime) | x \in RADNIK\}$$

- *restrikcija* – izdvajanje podskupa redova

$$\{x | x \in RADNIK \wedge x.org\_jed\_id = 40\}$$

- *proizvod* – uparivanje svih torki jedne sa svim torkama druge relacije:

$$\{(x, y)|x \in RADNIK \wedge y \in ORG\_JEDINICA\}$$

Kombinovanjem osnovnih operacija dobijaju se složeniji upiti, kao:

- *prirodno spajanje* – proizvod, pa restrikcija po uslovu jednakosti atributa sa istim imenom, pa projekcija na različite atribute:

$$\{(x.ime, y.naziv)|x \in RADNIK \wedge y \in ORG\_JEDINICA$$

$$\wedge\ x.org\_jed\_id = y.org\_jer\_id\}$$

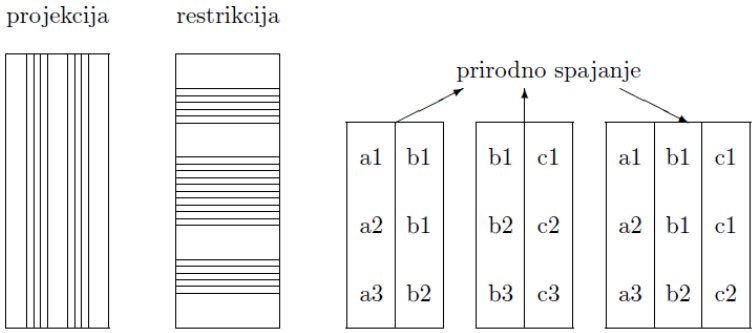
- *slobodno spajanje* — spajanje sa restrikcijom po slobodnom uslovu:

$$\{(x, y)|x \in RADNIK \wedge y \in KAT\_STAZA$$

$$\wedge\ y.od\_godine \leq staz(x) \leq y.do\_godine\}$$

uz pretpostavku da je

$$staz(x) = godina(danasnji\_datum - x.dat\_zaposlenja)$$



Slika 3.3: Operacije projekcije, restrikcije i prirodnog spajanja

Relacioni račun

Relacioni račun je zasnovan na predikatskom računu prvog reda.

Primeri upita

Primer 1 Naredni upit izdvaja muškarce koji imaju osnovu plate manju od 32000:

$$\{x|x \in RADNIK \wedge x.pol =' M' \wedge x.osnova\_plate < 32000\}$$

Primer 2 Ako nas zanimaju imena, prezimena i datumi zaposlenja radnika koji zarađuju bar 40000, onda je rešenje naredni upit:

$\{(x.ime, x.prezime, x.dat\_zaposlenja) | x \in RADNIK \wedge x.osnova\_plate \geq 40000\}$

Primer 3 Imena i prezimena zaposlenih u „Planiranju“, čiji staž ulazi u kategoriju „Srednja“:

$$\begin{aligned}
&\{(x.ime, x.prezime) | \\
&x \in RADNIK \\
&\wedge(\exists y \in ORG_JEDINICA) \\
&\quad (y.naziv = \ 'Planiranje' \\
&\quad \wedge x.org_jed_id = y.org_jed_id) \\
&\quad \wedge(\exists z \in KAT_STAZA) \\
&\quad (z.naziv = \ 'Srednja' \\
&\quad \wedge z.od_godine \leq staz(x) \leq z.do_godine) \\
&\}
\end{aligned}$$

Primer 4 Napisati upit koji izdvaja imena, prezimena i kategoriju staža zaposlenih koji imaju najveća primanja u svom odeljenju ili među radnicima istog pola:

$$\begin{aligned}
&\{(x.ime, x.prezime, y.naziv) | \\
&x \in RADNIK \wedge y \in KAT_STAZA \\
&\wedge y.od_godine \leq staz(x) \leq y.do_godine \\
&\wedge(\forall z \in RADNIK) \\
&\quad (z.org_jed_id = x.org_jed_id \\
&\quad \implies z.osnova_plate \leq x.osnova_plate) \\
&\quad \vee(\forall z \in RADNIK) \\
&\quad (z.pol = x.pol \implies z.osnova_plate \leq x.osnova_plate) \\
&\}
\end{aligned}$$

Optimizacija upita i baze podataka

Iz ugla teorije, ključno je da se utvrdi da je upitni jezik dovoljno dobar da se njime može iskazati bilo koji upit. U praksi je potrebna i efikasnost.

Optimizacija upita obuhvata poslove analize upita i pronalaženja najefikasnijeg puta za njegovo izračunavanje. Relacioni upitni jezici omogućavaju relativno jednostavno automatsko modifikovanje, pa time i optimizaciju, imajući u vidu formalnu zasnovanost.

Optimizacija baze podataka obuhvata poslove analize (stvarne ili procenjene) upotrebe i projektovanja fizičkog modela baze podataka u cilju omogućavanja sveukupno efikasnijeg rada sistema. Pored ažuriranja upita, važno mesto zauzima ažuriranje baze podataka. Ima više pristupa, ali suština je da zahteva proširenja i nove pojmove:

- relaciona promenljiva
- relacija
  = relaciona vrednost
  = sadržaj relacione promenljive
- promenljiva baze podataka

- vrednost baze podataka

Ažuriranje baze podataka je zamenjivanje vrednosti promenljive baze podataka novom vrednošću baze podataka.

3.3 Integritetni deo modela

Integritetni deo modela čine koncepti i mehanizmi koji omogućavaju da se automatizuje proveravanje zadovoljenosti određenih uslova. Stanje baze je konzistentno (tj. ispravno) ako sadržaj baze podataka ispunjava sve uslove integriteta. Promena sadržaja (vrednosti) baze podataka je dopuštena akko prevodi bazu iz jednog u drugo konzistentno stanje.

Baza podataka (tj. njena vrednost) se opisuje relacijama. Uslovi integriteta u relacionom modelu se opisuju predikatima (istinitosnim formulama) nad relacijom ili bazom podataka. Formalnost modela olakšava formulisanje uslova integriteta.

Vrste uslova integiriteta

- Opšti uslovi integriteta (implicitni)
  - Oni koji moraju da važe za svaku relaciju i svaku bazu podataka.
  - Podrazumevaju se na nivou modela ili implementacije SUBP.
  - Ne definišu se eksplicitno za svaku relaciju ili bazu podataka.
- Specifični uslovi integriteta (eksplicitni)
  - Oni koji se odnose na pojedini atribut, relaciju ili bazu podataka.
  - Određuju se pri određivanju strukture baze podataka.

3.3.1 Specifični uslovi integriteta

- Integritet domena
- Integritet ključa (integritet jedinstvenosti)
- Referencijalni integritet
- Integritet stranog ključa
- Opšti uslovi integriteta
- Aktivno održavanje integriteta

Integritet domena

Integritet domena određuje da svaki atribut može da ima samo neku od vrednosti iz unapred izabranog domena. Domen je neki od tipova podataka, a može da obuhvata i:

- dužinu podatka,
- opcionu deklaraciju jedinstvenosti,

- opcionu deklaraciju podrazumevane vrednosti,
- većina implementacija omogućava i deklarisanje da li domen obuhvata i tzv. nedefinisane vrednosti.

Jedan od opštih uslova integriteta je: „Svaka relacija mora da ima tačno određen domen“.

Integritet ključa

Odnosi se na jednu relaciju i određuje se uslovom ključa:

- Uslov ključa određuje minimalan podskup atributa relacije koji predstavlja jedinstveni identifikator torke relacije

Skup atributa $X = \{A_{i1}, \dots A_{ink}\}$ koji predstavlja podskup svih atributa relacije R je ključ (ili ključ kandidat) relacije R akko su ispunjeni sledeći uslovi:

- X funkcionalno određuje sve atribute relacije R i
- nijedan pravi podskup od X nema prethodno svojstvo.

Jedna relacija može da ima više ključeva. Jedan od ključeva se proglašava za primarni ključ i upotrebljava za jedinstveno identifikovanje torki relacije. Podskup X skupa atributa relacije R predstavlja natključ ako zadovoljava samo prvi od uslova ključa:

- X funkcionalno određuje sve atribute relacije R

Jedan od opštih uslova integriteta je: „Svaka relacija mora da ima primarni ključ“. U praksi neke implementacije omogućavaju postojanje relacija bez primarnog ključa, ali uz umanjenu funkcionalnost.

Torke relacije se po pravilu referišu pomoću primarnog ključa, iako može da se koristi i bilo koji drugi ključ. Ovakav pristup se preporučuje zbog efikasnosti.

Integritet jedinstvenosti („integritet entiteta“)

Jedan od opštih uslova integriteta *nedoslednih* implementacija, koje podržavaju nedefinisane vrednosti, je: „Primarni ključ ne sme da sadrži nedefinisane vrednosti, niti dve torke jedne relacije smeju imati iste vrednosti primarnih ključeva“.

U *doslednim* (ne obuhvataju nedefinisane vrednosti) relacionim modelima integritet jedinstvenosti je isto što i integritet ključa. Jedinstvenost vrednosti primarnog ključa je implicirana jedinstvenošću torki u relaciji.

U praksi je često integritet jedinstvenosti jači od integriteta ključa zato što postoji podrška za nedefinisane vrednosti. To znači da mogu da postoje dve identične torke u relaciji. Pored primarnog ključa, mogu da se eksplicitno deklarišu i jedinstveni ključevi, koji su po svemu isti kao primarni ključevi, ali nije uobičajeno da se koriste pri referisanju. Osnovna namena je implementacija dodatnih uslova jedinstvenosti torki.

Referencijalni integritet

Referencijalni integritet predstavlja uslove o međusobnim odnosima koje moraju da zadovoljavaju torke dveju relacija:

- ne sme se obrisati torka na koju se odnosi neka torka neke relacije u bazi podataka, niti se sme tako izmeniti da referenca postane neispravna,
- ne sme se dodati torka sa neispravnom referencom (takvom da ne postoji torka na koju se odnosi).

U praksi se ostvaruje putem integriteta stranog ključa. U slučaju nedoslednih implementacija, koje podržavaju nedefinisane vrednosti, uslovi referencijalnog integriteta su izmenjeni:

- ne sme se obrisati torka na koju se odnosi neka torka neke relacije u bazi podataka, niti se sme tako izmeniti da referenca postane neispravna,
- ne sme se dodati torka sa neispravnom referencom (takvom da ne postoji torka na koju se odnosi),
- referenca koja sadrži nedefinisane vrednosti je ispravna (i ne referiše ništa) akko je u potpunosti nedefinisana (sve vrednosti njenih atributa su nedefinisane).

Integritet stranog ključa

Skup atributa FK (foreign key) bazne relacije B je njen strani ključ koji se odnosi na relaciju P akko važe sledeći uslovi:

- relacija P ima primarni ključ PK
- domen ključa FK je identičan domenu ključa PK
- svaka vrednost ključa FK u torkama relacije B je identična ključu PK bar jedne torke relacije P (integritet ključa implicira jedinstvenost)
- Za relaciju B se kaže da je zavisna od relacije P

U slučaju nedoslednih implementacija, koje podržavaju nedefinisane vrednosti, strani ključ mora da ispunjava izmenjen skup uslova:

- relacija P ima primarni ključ PK
- domen ključa FK je identičan domenu ključa PK
- za svaku vrednost ključa FK u torkama relacije B važi da ili sve vrednosti atributa imaju nedefinisanu vrednost, ili nijedna vrednost atributa nije nedefinisana
- svaka vrednost ključa FK u torkama relacije B je ili u potpunosti nedefinisana ili je identična ključu PK bar jedne torke relacije P

Poštovanje integriteta stranog ključa pri menjanju sadržaja baze podataka se određuje pravilima koja mogu da budu:

- pravila brisanja (bira se tačno jedno)

- pravila ažuriranja (bira se tačno jedno)

Pravila brisanja:
Ako se pokuša brisanje torke relacije P, za koju postoji zavisna torka relacije B (ona čiji je strani ključ jednak primarnom ključu torke koja se pokušava obrisati) onda onda se može definisati:

- Aktivna zabrana brisanja – RESTRICT
  - zabranjuje se brisanje torke iz relacije P;
- Pasivna zabrana brisanja – NO ACTION
  - slično kao aktivna zabrana, ali se odlaže provera do samog kraja brisanja, zato što možda postoji linija kaskadnih pravila koja će obrisati zavisnu torku;
- Kaskadno brisanje – CASCADE
  - brišu se sve odgovarajuće zavisne torke B;
- Postavljanje nedefinisanih vrednosti – SET NULL
  - u svim zavisnim torkama relacije B atributi stranog ključa se postavljaju na nedefinisane vrednosti;
- Postavljanje podrazumevanih vrednosti – SET DEFAULT
  - u svim zavisnim torkama relacije B atributi stranog ključa se postavljaju na podrazumevane vrednosti.

Pravila ažuriranja:
Ako se pokuša menjanje primarnog ključa torke relacije P, za koju postoji zavisna torka relacije B (ona čiji je strani ključ jednak primarnom ključu torke koja se pokušava obrisati) onda se može definisati:

- aktivna zabrana menjanja – RESTRICT
  - zabranjuje se menjanje torke relacije P;
- pasivna zabrana menjanja – NO ACTION
  - slično kao aktivna zabrana, ali se odlaže provera do samog kraja menjanja, zato što možda postoji linija kaskadnih pravila koja će promeniti zavisnu torku;
- kaskadno menjanje – CASCADE
  - na isti način se menjaju atributi stranog ključa svim zavisnim torkama relacije B;
- postavljanje nedefinisanih vrednosti – SET NULL
  - u svim zavisnim torkama relacije B atributi stranog ključa se postavljaju na nedefinisane vrednosti;
- postavljanje podrazumevanih vrednosti – SET DEFAULT
  - u svim zavisnim torkama relacije B atributi stranog ključa se postavljaju na podrazumevane vrednosti.

3.3.2 Opšti uslovi integriteta

Pored opisanih posebnih uslova, postoje i opšti uslovi integriteta

- na atributu
- na relaciji
- na bazi podataka

Opšti uslov integriteta na atributu

Pri definisanju domena atributa mogu da se propišu i dodatni uslovi integriteta atributa (prisutno u praktično svim implementacijama):

- lokalnog karaktera, odnosi se na vrednost jednog atributa jedne torke
- uslov može da zavisi samo od vrednosti atributa,
- može da se koristi za dodatno sužavanje domena,
  - na primer, atribut mora da ima jednu od nekoliko vrednosti,
- može da se koristi za proveru ispravnosti složenih tipova podataka,
  - na primer, datum se predstavlja tekstom, ali se onda proverava da li tekstualni zapis predstavlja ispravan zapis datuma.

Na primer:

```
create table ... (
 attr1 int check (attr1 in (1,2,3)),
 ...)
```

Opšti uslov integriteta na torki

Pri definisanju relacije mogu da se propišu i dodatni uslovi integriteta torke (prisutno u praktično svim implementacijama):

- lokalnog karaktera, odnosi se na vrednost jedne torke,
- uslov može da zavisi od vrednosti svih atributa torke,
- koristi se za proveru ispravnosti složenijih saglasnosti atributa u okviru jedne torke,
  - na primer, ako se navodi neki interval, može da se proverava da li je početna vrednost intervala manja od krajnje vrednosti intervala.

Na primer:

```
create table ... (
 attr1 ...,
 attr2 ...,
 constraint check (attr1 < attr2)
 ...)
```

Opšti uslov integriteta na relaciji

Pri definisanju relacije mogu da se propišu i dodatni uslovi integriteta relacije (podržavaju ih samo neke naprednije implementacije):

- globalnog karaktera, može da se odnosi na sve torke jedne relacije,
- uslov može da zavisi od vrednosti atributa različitih torki relacije,
- koristi se za proveru ispravnosti složenijih saglasnosti vrednosti u okviru jedne relacije,
  - na primer, može da se proverava da li broj torki ili suma po nekom atributu zadovoljava određene uslove.

Obično su implementirani slično kao i uslovi integriteta na torkama.
Na primer:

```
create table T1(
 attr1 ...,
 attr2 ...,
 constraint check (
 select sum(attr1*attr2) from T1 < 1000
 )
 ...)
```

Opšti uslov integriteta na bazi podataka

Na nivou baze podataka mogu da se propišu i dodatni uslovi integriteta između više relacija (nisu podržani u svim savremenim implementacijama):

- globalnog karaktera, odnosi se na vrednosti torki u različitim relacijama,
- koristi se za proveru složenijih uslova integriteta,
  - na primer, ako sadržaj jedne relacije mora da bude u skladu sa sadržajem druge relacije, ali na način koji prevazilazi mogućnosti običnog referencijalnog integriteta.

Na primer:

```
create assertion asrt_1 check (
 select count(*) from T1 < select count(*) from T2
)
```

Sreću se pod nazivima constraints i assertions.

Aktivno održavanje integriteta

Do sada pomenuti oblici uslova integriteta osim tvrđenja uključuju, eksplicitno ili implicitno, i akciju koju sistem treba da preduzme u slučaju da tvrđenje nije tačno. Na primer, kod uslova integriteta na domenu, pokušaj unosa vrednosti atributa koja nije iz odgovarajućeg domena biće odbijena. To odbijanje zapravo predstavlja akciju koju sistem preduzima u slučaju da uslov integriteta nije zadovoljen. Slično, pokušaj da se izbriše neka n-torka iz bazne relacije, koja

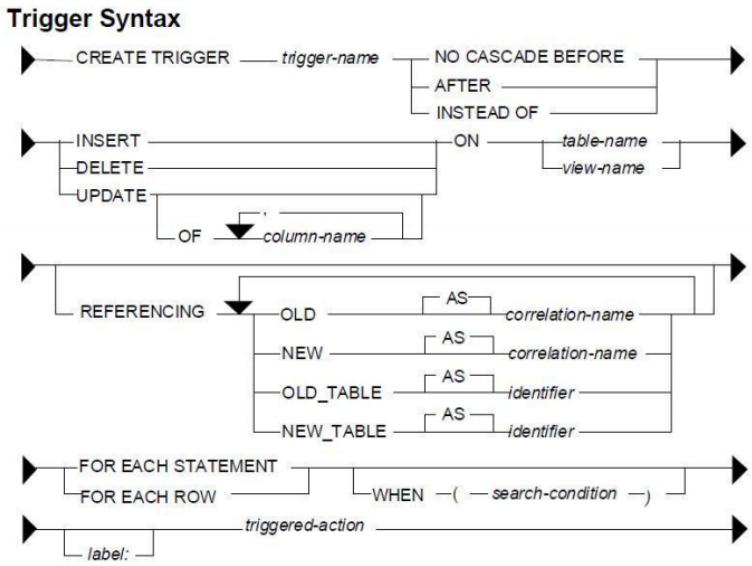
ima vrednost primarnog ključa jednaku vrednosti stranog ključa n-torke neke zavisne relacije, biće ili odbijen ili će proizvesti kaskadno brisanje odgovarajućih n-torki zavisne relacije, ili postavljanje markera nedostajućih ili podrazumevanih vrednosti stranog ključa u odgovarajućim n-torkama zavisnih relacija.

Opštija forma uslova integriteta, kojom se može definisati proizvoljna akcija u slučaju nekog događaja nad objektom u bazi, zove se *okidač ili triger* (engl. trigger). Triger je proceduralni mehanizam koji specifičnoj operaciji modifikovanja podataka (triger operaciji), nad specifičnom baznom relacijom ili pogledom, pridružuje niz SQL iskaza (triger proceduru) koja se ,,aktivira“ tj. izvrši kadgod se izvrši triger operacija [10].

Neke od primena trigera su: validacija ulaznih podataka, automatsko generisanje vrednosti unesene vrste, čitanje ili upis u druge tabele u cilju uspostavljanja međusobnih veza i drugo. Trigeri su osnovno sredstvo aktivnog održavanja integriteta.

Uobičajeno je da za svaku od operacija unošenja, izmene i brisanja postoje po dve vrste trigera:

- *pre-triger* — triger čija se triger procedura aktivira pre izvršenja same operacije
- *posle-triger* — triger čija se triger procedura aktivira posle izvršenja same operacije



Slika 3.4: Kreiranje trigera

Okidači na relacijama

Izvršavaju se pre (engl. BEFORE) ili posle (engl. AFTER) naredbe za dodavanje (engl. INSERT), menjanje (engl. UPDATE) ili brisanje (engl. DELETE) torki. Na jednoj relaciji može da bude više okidača. Nekad se mora reagovati

pre ili posle, a nekad je svejedno.

Primer okidača na relaciji: Automatski odrediti osnovnu platu novog službenika na osnovu nivoa obrazovanja (DB2).

```
CREATE OR REPLACE TRIGGER insert_set_salary
 NO CASCADE BEFORE INSERT ON employee
 REFERENCING NEW AS N FOR EACH ROW
 BEGIN
 SET n.salary = CASE n.edlevel WHEN 18 THEN 50000
  WHEN 16 THEN 40000
  ELSE 25000
 END;
END
```

Okidači na pogledima

Savremeni SUBP nude okidače na pogledima. Oni se izvršavaju umesto (enlg. INSTEAD OF) naredbe za dodavanje, menjanje ili brisanje torki iz pogleda. Dodatno, omogućavaju preusmeravanje izmena na relacije na kojima počiva pogled, ali i dalje od toga, na sasvim druge relacije.

Primer okidača na pogledu: Ako pogled V1 počiva na relacijama T1 i T2, okidačem se rešava kako se menjaju tabele ,,kroz pogled“.

```
CREATE TRIGGER V1_UPDATE INSTEAD OF UPDATE ON V1
 REFERENCING NEW AS n OLD AS o
 FOR EACH ROW MODE DB2SQL
 UPDATE T1 SET (c1, c2) = (n.c1, n.c2)
 WHERE c1 = o.c1 AND c2 = o.c2
```

Okidači na pogledima omogućavaju skrivanje veoma složenih operacija kojima se spoljašnja shema razdvaja od konceptualne, ili konceptualna od interne.

Primer okidača na pogledu: Implementira skriveno implicitno enkriptovanje lozinki pri zapisivanju u bazu podataka.

```
CREATE TRIGGER UPDATE_LOGINS INSTEAD OF UPDATE
 ON LOGINS REFERENCING OLD AS o NEW AS n
 FOR EACH ROW MODE DB2SQL
 UPDATE USERS U
 SET system = n.system,
 login = n.login,
 password = encrypt(n.password)
 WHERE system = o.system
 AND login = o.login
 AND U.user = USER
```

Razdvajanje nivoa modela

Osnovno sredstvo razdvajanja nivoa shema u relacionom modelu su pogledi

- Na nižem nivou (interna ili konceptualna shema) relacije su normalizovane radi stabilnosti i neredundantnosti
- Na višem nivou (konceptualna ili spoljašnja shema) relacije mogu da budu denormalizovane, tj. spajaju se radi lakšeg postavljanja upita i pristupanja podacima
- Privilegije na pogledima mogu da se potpuno razlikuju od privilegija na relacijama

Osnovno ograničenje pogleda:

- Pogledi nad više relacija ne mogu da se ažuriraju
- To značajno ograničava primenu pogleda u razdvajanju nivoa modela

Okidači na pogledima potpuno prevazilaze to ograničenje:

- Omogućavaju potpunu kontrolu nad upotrebom pogleda za razdvajanje nivoa modela
- Upotpunjuju i zaokružuju relacioni model

4

Model entiteta i odnosa (ER model)

Model entiteta i odnosa je predložio Piter Čen (Peter Chen) 1976. godine, kao naredni korak posle mrežnog, hijerarhijskog i relacionog modela. Pojednostavljeno rečeno, model entiteta i odnosa je konceptualni model podataka koji realni svet ,,vidi“ kroz entitete i njihove odnose.

Neke osnovne kritike postojećih modela (mrežnog, hijerarhijskog i relacionog) bile su da ne čuvaju meta informacije o entitetima i odnosima među njima, da relacioni model može da izgubi neke važne semantičke informacije o stvarnom svetu, i drugo. Model entiteta i odnosa se mogao posmatrati kao okvir iz kog se mogu razviti sva tri postojeća modela.

Model entiteta i odnosa prepoznaje dva različita osnovna koncepta:

- entitete i
- odnose.

Pun naziv ovog modela je ,,Model entiteta i odnosa“, ili na engleskom ,,Entity-Relationship Model“, zbog čega se skraćeno naziva i ,,ER Model“. To nije baš u duhu srpskog jezika i ,,Model EO“ bi bilo ispravnije, ali bi malo korazumeo da se radi baš o ovom modelu.

ER model teži da u samom modelu očuva sve bitne semantičke informacije o domenu koji se modelira. Cilj ER modeliranja je da stvori verni odraz realnosti u bazi podataka. Naglasimo to da je ER modeliranje u stvari konceptualno modeliranje. ER model ne daje opis baze podataka ali proizvod ER modeliranja predstavlja međukorak iz kojeg se lako može definisati baza podataka.

U ovom modelu možemo da razlikujemo 4 nivoa pogleda na podatke:

- *Konceptualni model*, kojim se predstavljaju apstraktne semantičke informacije, odnosno informacije koje se tiču entiteta i odnosa i koje postoje u našem umu;
- *Logički model*, kojim se predstavljaju strukture podataka koje čuvamo u bazi podataka. U ovom modelu se entiteti i odnosi predstavljaju podacima.

- *Fizički model* sa određenim nivoom apstrakcije, kojim se predstavljaju strukture podataka nezavisne od pristupnog puta (indeksi ili heš algoritmi), odnosno koje na zahtevaju sheme pretraživanja, sheme indeksiranja i sl.

- *Niski fizički model*, praktično bez apstrakcije, kojim se predstavljaju strukture podataka koje zavise od pristupnog puta.

4.1 Koncepti ER modeliranja

Entiteti i odnosi

Entitet je osnovni pojam koji se ne definiše, a čije je značenje stvar, biće, pojava ili događaj od značaja koji se može jednoznačno identifikovati. *Skup entiteta* (zvaćemo ga još i *tip entiteta*) se može definisati kao kolekcija entiteta sa zajedničkom definicijom. Skupovi entiteta ne moraju da budu disjunktni.

Odnos je neko međusobno pridruživanje skupova entiteta (na primer, otac-sin, radnik-preduzeće). Uloga entiteta u odnosu je funkcija koju on obavlja u tom odnosu.

Atributi

Informacije o entitetima i odnosima se izražavaju skupom parova atribut-vrednost. Vrednosti se klasifikuju u skupove vrednosti koji odgovaraju domenima atributa. Atribut može da se definiše kao funkcija koja preslikava skup entiteta u skup vrednosti ili Dekartov proizvod skupova vrednosti (sve ovo veoma liči na relacioni model). Atributi su, dakle, karakteristike tipa entiteta za koje smo zainteresovani. Oni su deskriptori čije su vrednosti pridružene individualnim entitetima specifičnog tipa. Vrednost atributa jednog entiteta je jedinstvena u svakom trenutku, a može se menjati u vremenu [10].

Svaki tip entiteta ima identifikator („primarni ključ“). On je sredstvo za razlikovanje i jednoznačno referisanje elemenata skupova entiteta, odnosno za referisanje instanci tipova entiteta. Vrednost identifikatora se ne menja u vremenu.

Atribute je moguće klasifikovati na sledeći način [10]:

- *osnovni* – to su vrednosti koje se obezbeđuju poslovanju. Takve smo atribute razmatrali do sada – na primer ime, adresa, itd. Ove vrednosti ne mogu da se izvedu iz drugih atributa.

- *definisani* – to su izmišljeni atributi i postoje samo u bazi podataka, na primer jedinstveni identifikator sektora. Jednom dodeljena vrednost se ne menja.

- *izvedeni* – ovo je vrednost koja može da se izračuna iz vrednosti drugih atributa u bazi podataka. Na primer, starost zaposlenog kada je datum rođenja u bazi. Ovi atributi u opštem slučaju ne bi trebalo da se pamte u bazi podataka već da se izračunavaju po potrebi.

Nemaju svi entiteti vrednost za svaki atribut; međutim, neki atributi moraju da imaju vrednost za sve entitete. Zato se atributi mogu klasifikovati u

- *opcione* i
- *obavezne*

Na primer, tip entiteta *Zaposleni* ima atribute *datum zaposlenja* i *datum prestanka*. Datum zaposlenja je svakako obavezni atribut jer ako nema datuma zaposlenja, nije zaposleni. Datum prestanka je opcioni atribut. Za očekivanje je da mnogi zaposleni u bazi podataka nemaju datum prestanka dok drugi, bivši zaposleni, imaju vrednost tog atributa.

Opcionost atributa zavisi od poslovne situacije, od toga kako se informacija prikuplja i kako poslovanje ažurira svoju bazu podataka. Isti atribut može da bude različito klasifikovan od strane raznih kompanija.

Primer: Razmotrimo atribut prodajna cena tipa entiteta kataloški proizvod kompanije za naručivanje e-poštom. Kompanija A ima politiku da ne stavlja proizvod u katalog dok nema cenu; dakle, oni ne kreiraju entitet kataloški proizvod dok ne dodele vrednost njegovom atributu prodajna cena. Za ovu kompaniju, prodajna cena je obavezni atribut.

S druge strane, kompanija B ima politiku da stavlja proizvod u katalog čim odluči da ga skladišti. Na taj način prikazuju što je moguće širu lepezu proizvoda. Oni stavljaju u katalog oznaku ,,Pozovite nas za najnovije informacije“ umesto cene. Dakle, za ovu kompaniju prodajna cena je opcioni atribut.

Slabi i jaki entiteti

Entiteti mogu da postoje na različitim nivoima samostalnosti. ER model razlikuje:

- *slabe entitetske relacije*, kod kojih u identifikovanju entiteta učestuju odnosi sa drugim entitetima, odnosno u praksi u primarnom ključu učestvuje referenca na drugi entitet i
- *obične (regularne) entitetske relacije*, kod kojiih u identifikovanju entiteta ne učestuju odnosi sa drugim entitetima, odnosno, u primarnom ključu ne učestvuje referenca na drugi entitet.

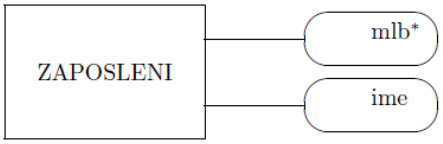
To često u literaturi prerasta u jednostavnije koncepte:

- *jaki entiteti*, koji se identifikuju sami za sebe i koje karakteriše njihova egzistencijalna nezavisnost od drugih entiteta u bazi,
- *slabi (opisni, zavisni) entiteti*, koje karakteriše njihova egzistencijalna zavisnost od drugih entiteta u bazi, odnosno oni opisuju, kao svojstvo, neki drugi entitet.

4.2 ER dijagrami

ER dijagrami su sastavni deo ER modela i služe za vizuelno predstavljanje podataka. Oni predstavljaju osnovni način zapisivanja semantičkih znanja o informacijama. Tipovi entiteta se predstavljaju pravougaonicima, atributi se predstavljaju elipsama, a primenjivost atributa na tip entiteta se predstavlja linijom.

Izbor atributa reflektuje nivo detalja na kom želimo da predstavimo informacije o tipovima entiteta. Na primer, entitet *Zaposleni* može se predstaviti svojim sledećim atributima: matični lični broj (mlb) i ime. U ovom slučaju nemamo informacije o, na primer, adresi zaposlenog, ili polu i godinama.



Slika 4.1: Primer ER dijagrama - tip entiteta i atributi

ER dijagrami – kardinalnost

Odnosi se predstavljaju rombovima i mogu da imaju opisne atribute. Odnosi mogu da uključuju više tipova entiteta.

Kardinalnost odnosa opisuje u koliko različitih odnosa tog tipa može da učestvuje svaki od entiteta. Formalnije, kardinalnost preslikavanja tipa entiteta E1 u tip entita E2 možemo definisati kao par (DG, GG), pri čemu je DG (donja granica) najmanji mogući, a GG (gornja granica) najveći mogući broj dodeljenih entiteta tipa E2 jednom entitetu tipa E1. Pri tome, $DG \in \{0, 1, broj > 1\}$ i $GG \in \{1, broj > 1, M\}$, gde je M neka celobrojna promenljiva [10].

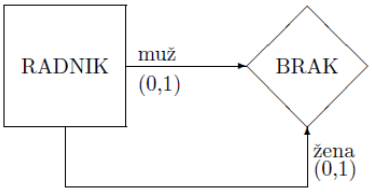
Stepen odnosa

Odnosi mogu da se klasifikuju prema broju tipova entiteta koji u njima učestvuju. Taj broj se naziva stepenom odnosa. Do sada smo se bavili najčešćom vrstom odnosa – binarnim odnosom tj. odnosom između dva tipa entiteta. Uobičajeni stepeni odnosa su:

- *binarni* – odnos između dva tipa entiteta,
- *ternarni* – odnos između tri tipa entiteta,
- *rekurzivni* – odnos koji uključuje samo jedan tip entiteta.

Slika 4.2: Primer ER dijagrama - binarni odnos [10]

Odnosi mogu da budu između više instanci istog tipa entiteta. U tom slučaju moraju da se imenuju uloge koje entiteti imaju u odnosima.



Slika 4.3: Primer ER dijagrama - rekurzivni odnos [10]

Ako na osnovu entiteta može da se jednoznačno odredi odnos u kome učestvuje, onda je to *ključni entitet odnosa* i označava se strelicom od tipa entiteta prema odnosu.
Ako svaka instanca tipa entiteta učestvuje u bar jednom odnosu, onda je to *puno učešće u odnosu*, a inače je *parcijalno učešće u odnosu*.

Primer punog učešća:

- svaki zaposleni radi u bar jednom odeljenju,
- svako odeljenje ima bar jednog zaposlenog,
- svako odeljenje ima tačno jednog rukovodioca.

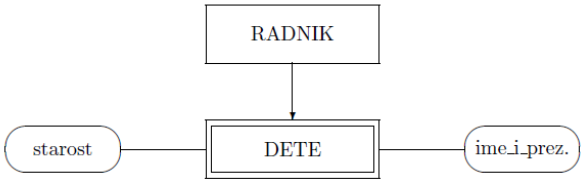
Primer parcijalnog učešća:

- ne mora svaki zaposleni da rukovodi nekim odeljenjem.

Puno učešće u odnosu se može označavati debljom linijom.

ER dijagrami – slabi entiteti

Elementi slabog tipa entiteta se identifikuju samo u sklopu odnosa sa nekim drugim entitetom. Taj odnos mora biti identifikujući odnos i slab entitet mora imati puno učešće u tom odnosu. Na primer, ako su deca zaposlenog osigurana preko roditelja, dete nije u potpunosti identifikovano imenom, nego tek odnosom sa zaposlenim roditeljem. Identifikujući odnos i učešće slabog entiteta u njemu mogu da se označavaju debljim (ili dvostrukim) linijama.



Slika 4.4: Primer ER dijagrama - slabi entiteti [10]

ER dijagrami – hijerarhije tipova entiteta

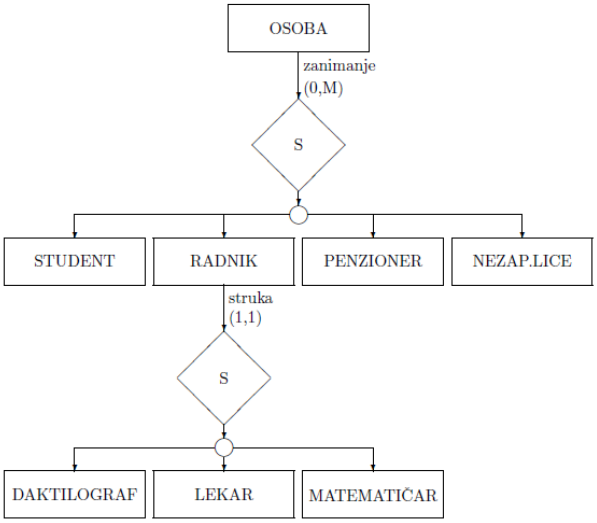
Između dva tipa entiteta X i Y važi odnos podtip/nadtip ako je svaki entitet tipa X obavezno i entitet tipa Y. U ovom odnosu, tip entiteta X je podtip,

a tip entiteta Y je nadtip. Pri tome, tip entiteta Y može da bude nadtip za veći broj tipova entiteta u datom odnosu podtip/nadtip, dok je tip entiteta X podtip samo tipa entiteta Y u tom odnosu podtip/nadtip. Na ovaj način, među tipovima entiteta uspostavlja se tzv ,,ISA“ hijerarhija.

Od značaja su sledeća dva uslova:

- *Uslov preklapanja:* da li isti entitet može da bude pripadnik dva podtipa ili ne?
- *Uslov prekrivanja:* da li svaki entitet nadtipa pripada nekom od podtipova ili ne?

Kasnije ćemo videti da hijerarhije u relacionom modelu mogu da se modeliraju na više načina, a izbor može da zavisi i od prethodna dva uslova.

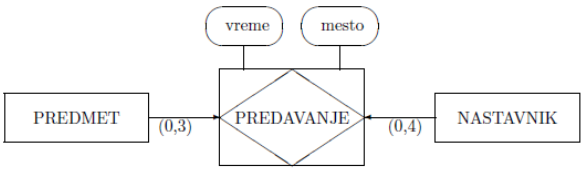


Slika 4.5: Primer ER dijagrama - hijerarhije tipova entiteta [10]

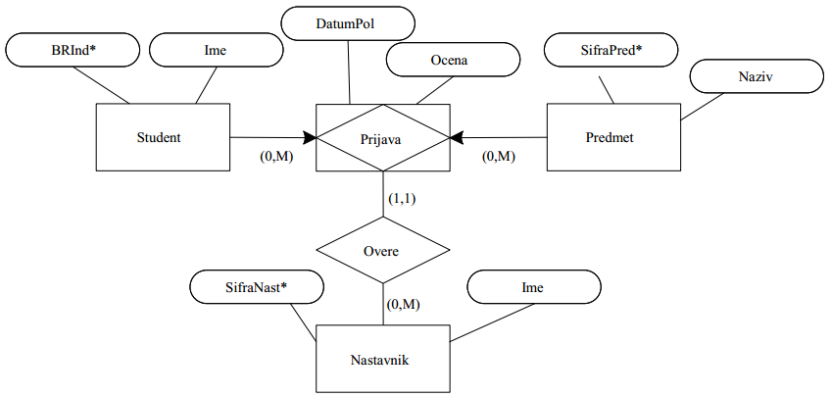
Pogled na bazu podataka u kome se neki detalji namerno zanemaruju zove se *apstrakcija* te baze podataka. Apstrakcija kojom se skup sličnih tipova entiteta posmatra kao jedan generički tip entiteta zove se *generalizacija*, pri tome su ti slični tipovi entiteta zapravo podtipovi generičkog tipa. Apstrakcija koja je ,,inverzna“ generalizaciji zove se *specijalizacija* [10].

ER dijagrami – agregacija

Asocijacija predstavlja odnos više-više dva tipa entiteta (ili više tipova entiteta), koji može imati i svoja sopstvena svojstva. Na primer, između tipova entiteta PREDMET i NASTAVNIK može se uspostaviti odnos (asocijacija) PREDAVANJE, sa svojstvima kao što su vreme i mesto. Osim što je asocijacija odnos između različitih tipova entiteta, ona se može posmatrati i kao entitet specifičnog tipa – *asocijativni entitet*. Apstrakcija kojom se odnos između dva ili više tipova entiteta definiše kao entitet nove vrste zove se *agregacija* [10].

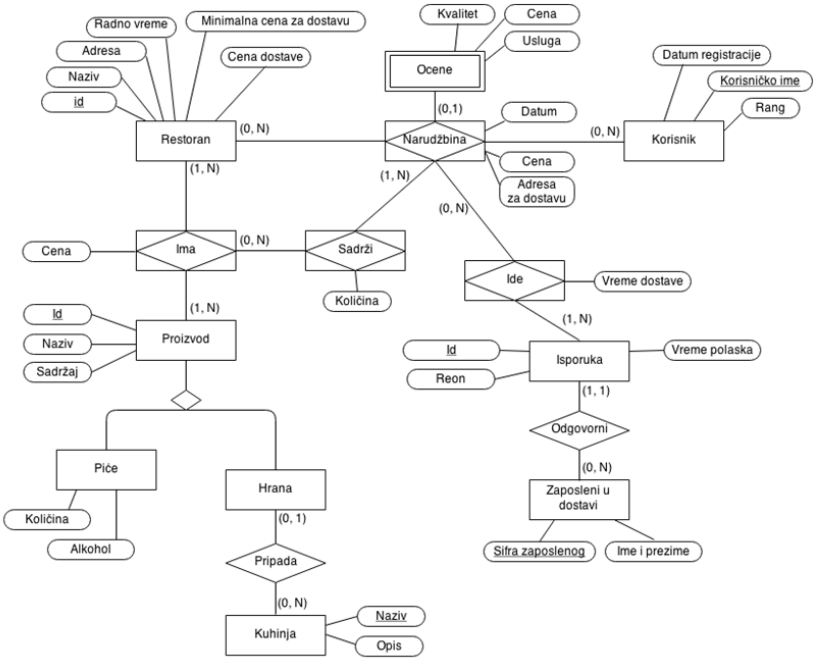


Slika 4.6: Primer 1: ER dijagram - agregacija [10]

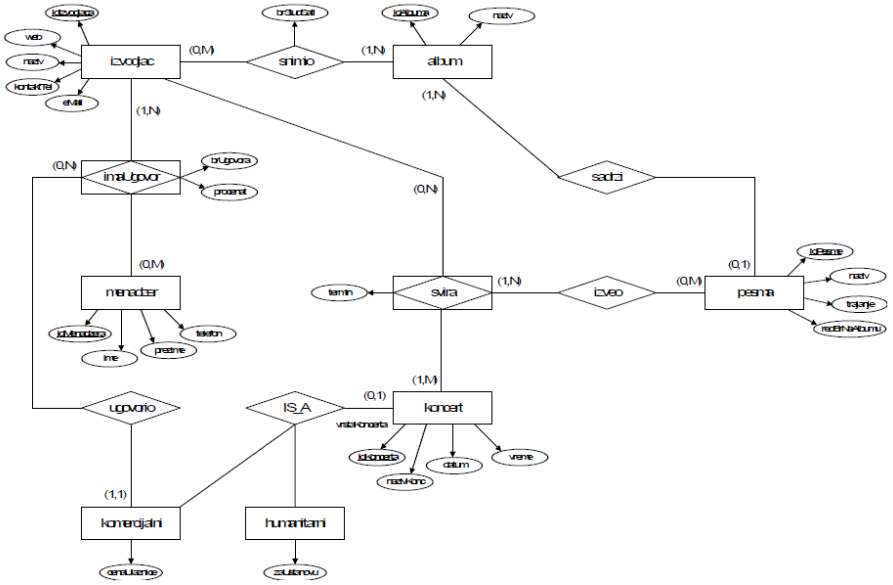


Slika 4.7: Primer 2: ER dijagram - agregacija

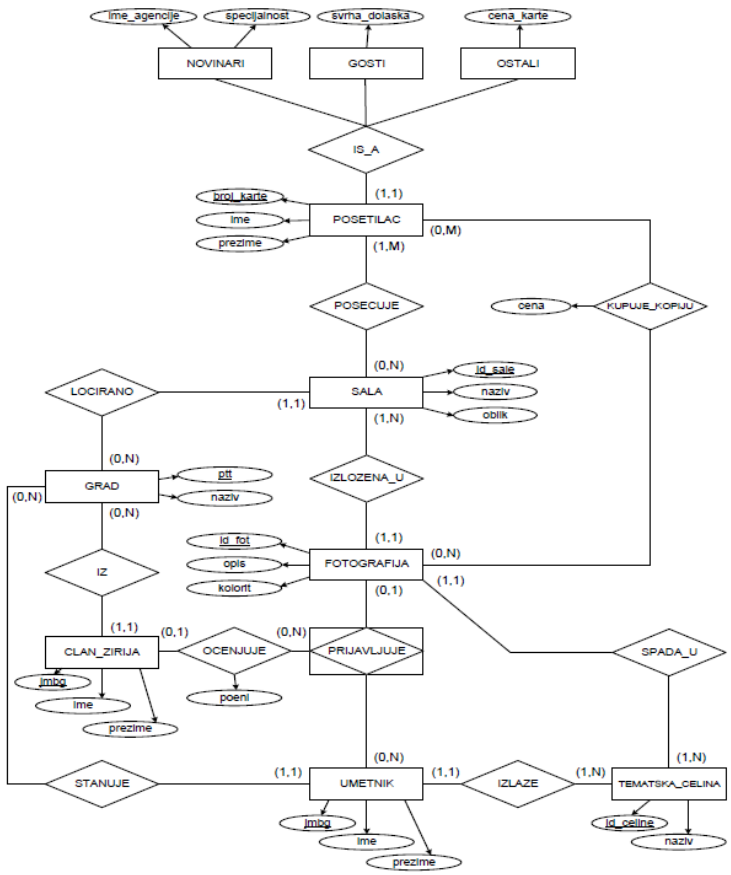
Primeri ER dijagrama:



Slika 4.8: ER dijagram online restorana



Slika 4.9: ER dijagram koncerta



Slika 4.10: ER dijagram izložbe

Alternativna notacija

Ovde treba biti oprezan. Notacija nije ujednačena i razni izvori koriste razne notacije.

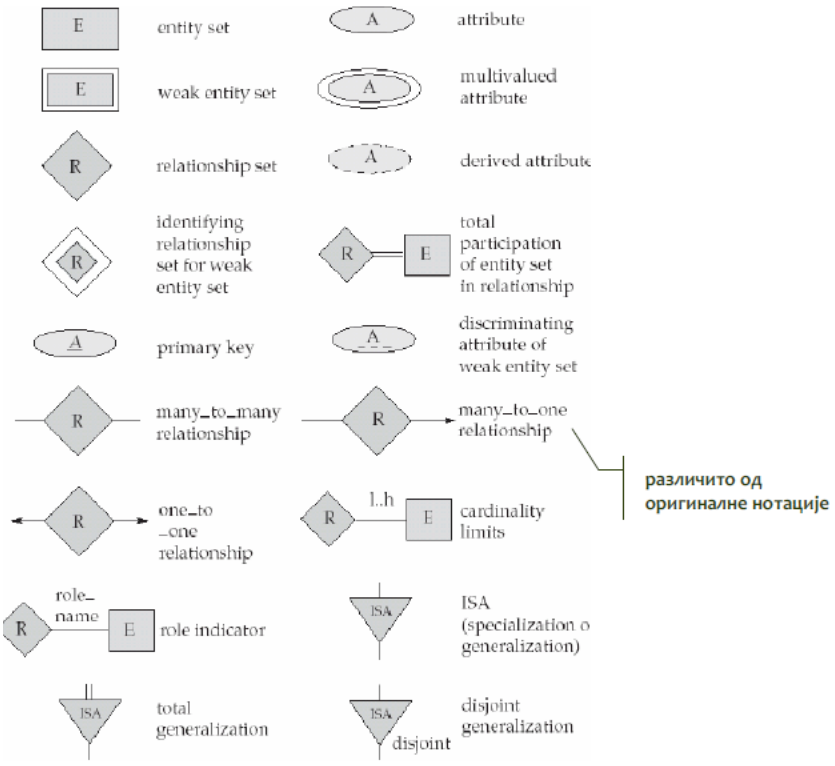
Na primer, kardinalnost se može predstaviti i na sledeći način:

- Označava se po brojem na svakoj od linija koje vode od entiteta prema odnosu. Pri tom broj označava sa koliko različitih entiteta tog tipa mogu biti u odnosu ostali učesnici;
- Umesto broja može da stoji opseg vrednosti u oznaci $A..B$, gde je $A \leq B$;
- Umesto broja može da stoji i $*$ koja označava ,,neki prirodan broj“.

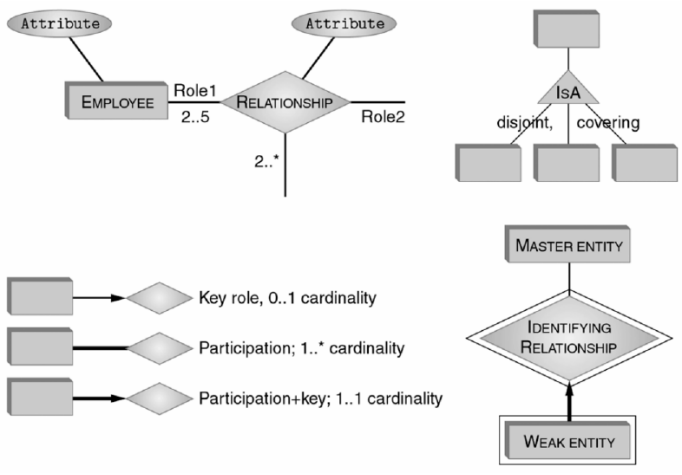
Ispravne oznake kardinalnosti u ovoj notaciji su, na primer: 1, 4, 0..1, 2..5, 1..*, 0..*.

Slika 4.11: ER dijagrami - alternativna notacija

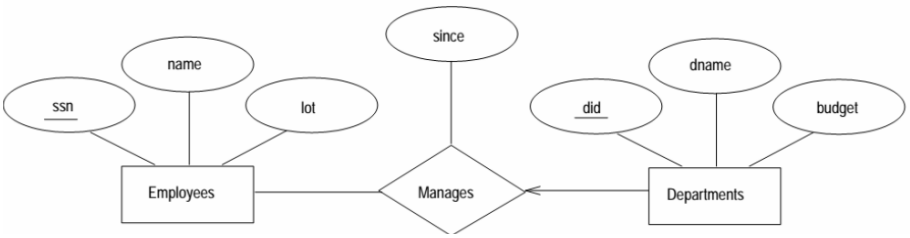
Važno je napomenuti da u originalnoj notaciji oznake kardinalnosti stoje na suprotnim stranama u odnosu na oznake u dijagramima klasa u UML-u. Neki autori to zanemaruju i rade kao u UML-u!



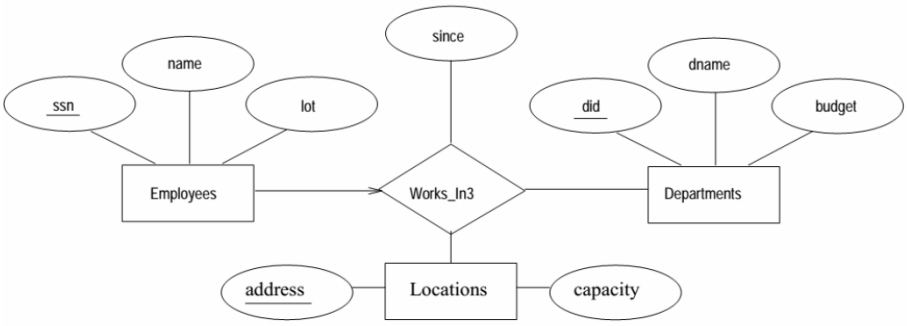
Slika 4.12: ER dijagrami - alternativna notacija



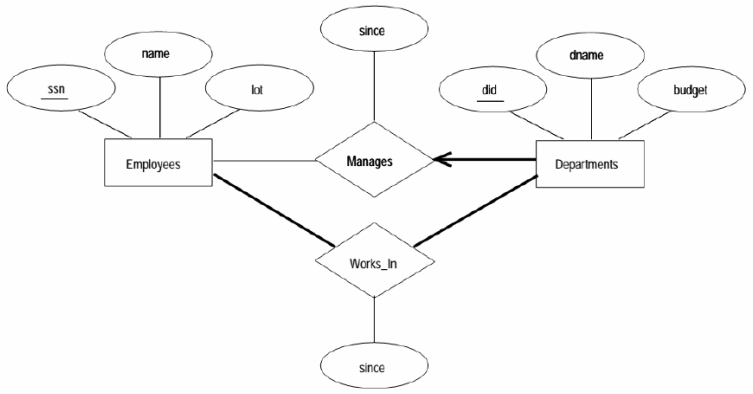
Slika 4.13: ER dijagrami - alternativna notacija



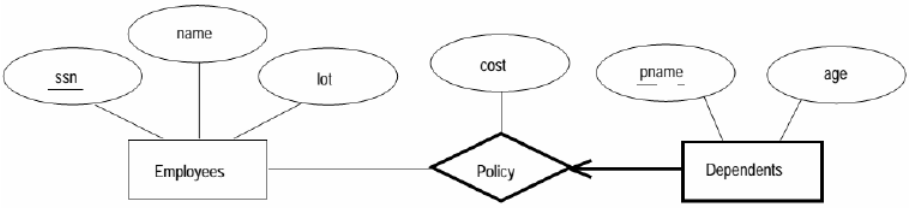
Slika 4.14: ER dijagrami (alternativna notacija) - odeljenje ima najviše jednog rukovodioca [13]



Slika 4.15: ER dijagrami (alternativna notacija) - zaposleni radi u najviše jednom odeljenju [13]

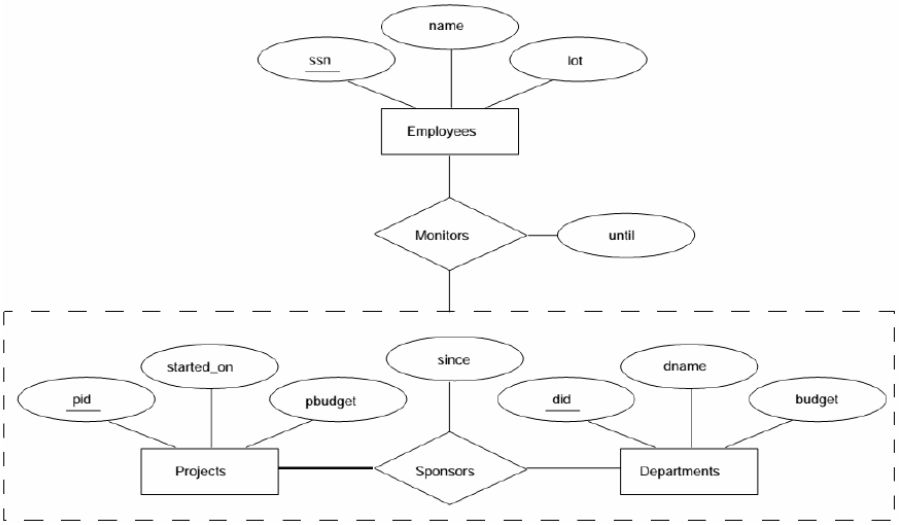


Slika 4.16: ER dijagrami (alternativna notacija) - uslovi učešća [13]



Slika 4.17: ER dijagrami (alternativna notacija) - slabi entiteti [13]

Agregacija nije isto što i ternarna relacija. Na primer, projekti mogu da se sponzorišu i bez pridruženih posmatrača. Agregaciju možemo da uokvirimo isprekidanom linijom.



Slika 4.18: ER dijagrami (alternativna notacija) - agregacija [13]

Doprinos ER modela

U odnosu na relacioni model ER model zadržava semantička znanja o informacijama u značajno većem obimu. Ovaj model obuhvata dijagramsku tehniku koja predstavlja jednostavno i izražajno sredstvo komunikacije i sadrži deo semantičkih znanja. On je bolje prilagođen najvišem nivou posmatranja informacija, odnosno semantičkom i konceptualnom modeliranju.

Konceptualni problem ER modela

U ovom modelu postoje određene nedoslednosti. Počiva na pretpostavci da postoje dva različita koncepta entiteta i odnosa, ali ne nudi objektivne kriterijume za njihovo razlikovanje. Način predstavljanja agregacija na dijagramima dodatno potvrđuje da odnosi mogu da imaju neke karakteristike entiteta. Složeni atributi su dodatni problem. Postavlja se pitanje da li složeni atributi imaju osobine entiteta?

Osnovni konceptualni problem ER modela uviđa i sam autor modela, navodeći ga u fusnoti rada kao komentar kome ne pridaje poseban značaj: „ Moguće je da neki ljudi vide nešto (npr. brak) kao entitet, dok neki drugi ljudi to vide kao odnos. Mišljenja smo da odluku o tome mora da donese administrator preduzeća. On bi trebalo da definiše šta su entiteti a šta odnosi tako da razlika bude odgovarajuća za njegovo okruženje.“ Sam autor ovim priznaje da su osnovni koncepti modela samo subjektivno a ne i suštinski različiti.

Bez objektivnih kriterijuma za njihovo razlikovanje, dva osnovna koncepta se praktično stapaju u jedan. Time nestaje suštinska razlika (u delu koji se odnosi na nivoe 2 i 3) između ER modela i relacionog modela. Zadržavanjem razlika samo na nivo 1, ER model se u teoriji svodi na alat za projektovanje, umesto na sveobuhvatni model podataka. Štaviše, danas se ER najčešće i ne koristi kao model, nego samo kao dijagramska tehnika relacionog modela. To svakako ne umanjuje njegov značaj u domenu konceptualnog modeliranja.

5

ER model u procesu projektovanja baza podataka

5.1 Koraci u projektovanju BP

Osnovni koraci u projektovanju baza podataka su:

- Analiza zahteva
- Konceptualno projektovanje
- Logičko projektovanje
- Prečišćavanje sheme
- Fizičko projektovanje
- Projektovanje bezbednosti

Analiza zahteva

Analiza zahteva podrazumeva pre svega razumevanje podataka, i to strukture podataka, obima podataka i odnosa među podacima. Analiza zahteva podrazumeva i razumevanje aplikacija, i to potrebe aplikacije za podacima, učestalost upita i transakcija i performanse.

Konceptualno projektovanje

Ovaj korak podrazumeva pravljenje modela podataka visokog nivoa. Podrazumeva i opisivanje struktura podataka, odnosa među strukturama podataka i uslova integriteta. U realizaciji ovog koraka često se koristi ER model ili bar ER dijagrami.

Logičko projektovanje BP

Bira se konkretna vrsta, a često i implementacija SUBP. Konceptualni model se prilagođava konkretnom implementacionom modelu podataka. Na primer, ukoliko se izabere relacioni SUBP, ključni posao ovog koraka je prevođenje konceptualnog modela na relacioni model.

Prečišćavanje sheme

Ovaj zadatak podrazumeva analizu logičkog modela kao i prepoznavanje i rešavanje problema u implementaciji. U slučaju relacionog modela, ovaj zadatak se obično svodi na zadatak normalizacije.

Fizičko projektovanje

Razmatra se očekivano opterećenje. Dodatno se popravlja model BP tako da zadovolji postavljene kriterijume vezano za performanse.

Uobičajeni koraci:

- projektovanje indeksa
- denormalizacija
- projektovanje distribuiranja
- i drugo

Projektovanje bezbednosti

Prepoznaju se vrste/uloge korisnika i vrste aplikacija. Za svaku ulogu i za svaku vrstu aplikacija definišu se korisnička grupa i odgovarajući minimalan skup privilegija za obavljanje posla. Prepoznaju se delovi baze podataka koji su posebno osetljivi i kojima je potrebno dodatno redukovati pristup. Definišu se i implementiraju odgovarajući bezbednosni mehanizmi.

5.2 ER model u projektovanju BP

ER model se najviše koristi u sledećim koracima:

- Konceptualno projektovanje, koje često za rezultat ima upravo ER model,
- Logičko projektovanje, čiji je često glavni posao prevesti konceptualni projekat iz ER modela na relacioni model.

Ukratko ćemo razmotriti specifičnosti koje se odnose na ER modele.

ER model i konceptualno projektovanje BP

Glavni koraci u ER modelovanju su [6]:

- Odrediti tipove entiteta
- Odrediti koji tipovi entiteta grade odnos
- Profiniti definiciju odnosa

Najčešći scenario:

- Poći od tekstualnog opisa problema
- Analizirati ga i uvesti neophodne pretpostavke
- Kreirati ER dijagram koji odslikava situaciju i eksplicira odnose među tipovima entiteta
- Korak od ER dijagrama ka implementaciji baze podataka biće pravolinijski
- Kreiranje baze podataka i jeste naš glavni cilj

Primer 1: Potrebno je kreirati bazu podataka za sledeću situaciju:

- Svaki sektor u okviru kompanije pripada tačno jednom odeljenju. Svako odeljenje ima veše sektora. Nema gornje granice broja sektora u okviru jednog odeljenja. Na primer, *New Business Development* — sektor kojim upravlja *Mackenzie* — i *Higher Education* sektor su u odeljenju *Marketing*.
- Mnoge stvari se podrazumevaju u vezi sa ovim opisom. Na primer, svako odeljenje ima *ime*, ime je jedinstveno u okviru kompanije, i sl.
- Prvi korak u ER modeliranju je uočiti stavke od značaja u opisanoj situaciji.

Problem 1.1: Koje su stavke od značaja u opisanoj situaciji?

Shodno prethodim definicijama pojma entiteta i tipa (skupa) entiteta, možemo da zaključimo da je *Sektor* ime tipa entiteta. Jedna instanca ovog tipa entiteta je *New Business Development* sektor. *Marketing* odeljenje je instanca tipa entiteta *Odeljenje*. *Mackenzie* je instanca tipa entiteta *Zaposleni*. Tipovi entiteta omogućuju da se informacija može formulisati u formi iskaza o odnosima između tipova entiteta, a ne odnosima među entitetima (npr. „svaki sektor je u tačno jednom odeljenju“).

Problem 1.2: Šta su tipovi entiteta a šta entiteti?

U prethodnom opisu uviđamo da postoji jedna vrsta veze između Sektora i Odeljenja i druga vrsta veze između Sektora i Zaposlenog. Prva veza je veza sadržavanja – svako odeljenje ima jedan ili više sektora, a jedan sektor pripada tačno jednom odeljenju. Drugi tip veze govori da je jedan zaposleni u izvesnom odnosu prema nekom sektoru, i to da taj zaposleni upravlja tim sektorom, a da se sektorom upravlja od (sektorom je upravljano od) strane tog zaposlenog. Dakle, drugi važan korak u ER modeliranju je određivanje odnosa između tipova entiteta.

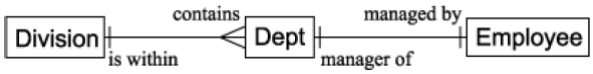
Problem 1.3: Kako biste nazvali ove dve vrste veza?

Slika 5.1: Veze u ER modelu [6]

Karakteristika odnosa je da uključuje nekoliko tipova entiteta. Zato *ime* ili *datum rođenja* nisu odnosi već atributi.
Objašenjenje dijagrama: Sektor (Department) / Odeljenje (Division), Sektor / Zaposleni. Zaposleni nisu u direktnoj vezi sa odeljenjima. Znamo i više: Znamo da u svakom (jednom) odeljenju može da bude više sektora.

Problem 1.4: Kakav je odnos između sektora i direktora (jedan/više)? Koliko u svakom sektoru može da bude najviše direktora? Za svakog direktora, koliko može da bude najviše sektora?

Informacija o kardinalnosti može da se predstavi ER dijagramom:



Slika 5.2: Kardinalnost u ER modelu [6]

Sada ER dijagram predstavlja više informacija:

- Svako odeljenje može da sadrži više sektora, a svaki sektor pripada najviše jednom odeljenju.
- Svaki sektor može da ima najviše jednog direktora. Svaki zaposleni može da upravlja (da bude direktor) najviše jednim sektorom

Druga pitanja:

- Koliki je najmanji broj sektora u odeljenju?
- Da li sektor mora da bude pridružen odeljenju?
- Da li sektor mora da ima direktora?

I ova pitanja moraju da dobiju odgovor u ER modelu.

Egzistencija odnosa definiše ono što znamo o postojanju entiteta na drugoj strani odnosa, i zadaje se kao *opciona*, *obavezna* ili *nepoznata*.

Razmotrimo, na primer, odnos *upravlja* između sektora i zaposlenog. Znamo da je kardinalnost odnosa 1:1. To nam govori da je jedan zaposleni - direktor najviše jednog sektora i da jedan sektor ima najviše jednog zaposlenog kao svog direktora. Egzistencija ovog odnosa govori o najmanjem broju sektora kojim može da upravlja jedan zaposleni, i najmanjem broju zaposlenih koji mogu da upravljaju jednim sektorom. Mogućnosti su:

Za sektor:

- Opciona: Sektor ne mora da ima direktora, ili
- Obavezna: Sektor mora da ima bar jednog direktora, ili
- Nepoznata: ne zna se da li sektor mora da ima direktora

Slično, za zaposlenog:

- Opciona: Radnik ne mora da upravlja nijednim sektorom
- Obavezna: Radnik mora da upravlja bar jednim sektorom
- Nepoznata: ne zna se da li radnik mora da upravlja sektorom

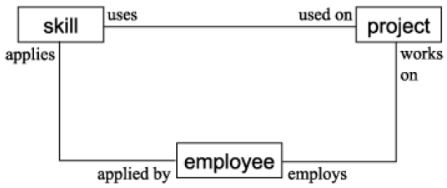
Problem 1.5: Šta biste izabrali iz svakog od prethodna dva skupa?

Za proizvoljni sektor, mora da postoji zaposleni na drugoj strani odnosa *upravlja*, pa je odnos obavezan u tom smeru. Predstavlja se crticom uspravnom na liniju odnosa, bližom sredini te linije. Iz sličnih razloga, odnos je opcioni u drugom smeru i predstavlja se kružićem. Možemo da pretpostavimo da je odnos *sadrži* obavezan u oba smera, pa ER dijagram može da se modifikuje tako da postane:

Slika 5.3: Egzistencija u ER modelu [6]

Dijagram se čita sleva na desno i zdesna na levo. Unutrašnje oznake predstavljaju minimalni broj u odnosu, spoljašnje maksimalni broj u odnosu. Crtica označava obaveznost (najmanji broj 1), kružić opcionost (najmanji broj je 0); ako nema oznaka, minimalni broj je nepoznat. Oznake bliže pravougaonicima predstavljaju maksimalni broj u odnosu. Viljuška označava kardinalnost ,,više“, crtica ,,1“.

Primer 2 (ilustracija ternarnog odnosa). Želimo da prikažemo koje veštine (skill) zaposleni (employee) koriste na kojim projektima (project). Mogu da se koriste tri binarna odnosa (veštine – projekti, projekti – zaposleni, zaposleni – veštine):



Slika 5.4: Tri binarna odnosa u ER modelu [6]

Odnos *primenjuje (apply)* ukazuje na to koji zaposleni koristi koje veštine. Odnos *korišćena na (used on)* ukazuje koja se veština koristi na kom projektu. Odnos *radi na (works on)* ukazuje koji zaposleni radi na kom projektu. Ali to još uvek nije dovoljno da se prikaže koji zaposleni koristi koje veštine na kom projektu.

- Primer za odnos *radi na*: Petar i Ana rade na projektima A i B.
- Primer za odnos *primenjuje*: Petar primenjuje veštine *interface design* i *database design* a Ana samo veštinu *database design*.
- Primer za odnos *korišćena na*: Obe veštine se koriste na oba projekta.

Šta je odgovor na sledeća pitanja: Na kojim projektima Petar koristi koje veštine? Možda *interface design* na projektu B a *database design* na projektu A — ili obratno? Ili obe na oba? Nema dovoljno informacija u bazi.

Potreban je ternarni odnos, *esp* (employee, skill, project), između zaposlenih, veština i projekata.

Slika 5.5: Ternarni odnos u ER modelu [6]

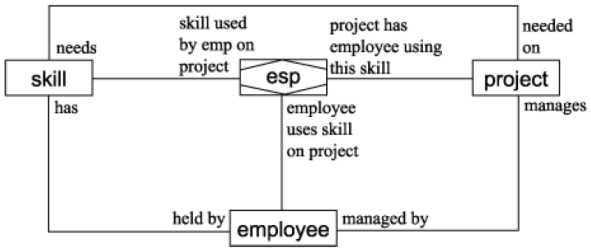
Odnos *esp* odražava činjenice kao što su:

1. Petar koristi *interface design* veštinu na projektu A.
2. Ana koristi *database design* veštinu na projektu A.
3. Petar koristi *interface design* veštinu na projektu B.
4. Petar koristi *database design* veštinu na projektu B.
5. Ana koristi *database design* veštinu na projektu B.

Ternarni odnos *esp* sadrži i informacije predstavljene binarnim odnosima:

- Petar radi na projektu A (iz 1.).
- Petar radi na projektu B (iz 3. i 4.).
- Ana radi na projektu A (iz 2.).
- Ana radi na projektu B (iz 5.).
- Petar koristi veštinu *interface design* (iz 1. i 3.).
- Petar koristi veštinu *database design* (iz 4.).
- Ana koristi veštinu *database design* (iz 2. i 5.).
- Obe veštine koriste se na oba projekta (iz svih).

Ternarni odnos ne isključuje potrebu za binarnim. Binarni odnosi su suvišni samo kada odražavaju informaciju koja je podskup informacije koju sadrži ternarni odnos. Ako binarni odnos sadrži informaciju koja se razlikuje od one sadržane u ternarnom, binarni odnosi se zadržavaju.



Slika 5.6: Ternarni i binarni odnos u ER modelu [6]

Odnos *esp* ostaje isti kao i ranije. Binarni odnosi su drugačiji.

- odnos *poseduje/posedovan od (has/held by)* – zaposleni poseduje neke veštine. Ovaj odnos se razlikuje od *esp* odnosa zato što mogu postojati veštine koje zaposleni poseduje ali koje nije koristio u okviru nekog projekta.

- odnos *zahteva/biti potreban (needs/needed on)* – neki projekat zahteva specifične veštine. Ovaj odnos se razlikuje od *esp* odnosa zato što mogu da postoje veštine za koje nekom projektu još nisu pridruženi zaposleni koji ih poseduju.

- odnos *upravlja/upravljan od (manages/managed by)* – zaposleni upravlja projektom. To je potpuno drugačija dimenzija od veštine i zato se ne može obuhvatiti *esp* odnosom.

Metod određivanja kardinalnosti odnosa višeg reda razlikuje se od određivanja kardinalnosti binarnog odnosa:

- Prekriti sve linije koje idu od ternarnog odnosa (npr. esp) prema nekom entitetu, osim jedne. (Na primer, prekriti linije iz esp prema employee i project.)

- Za preostali entitet, zapitati se: da li može da postoji najviše jedan primerak tog entiteta, za svaku pojedinačnu kombinaciju preostalih entiteta, ili ih može biti više? (U našem primeru, zapitati se: da li može da postoji najviše jedna veština za nekog zaposlenog i neki projekat, tj. koju neki zaposleni primenjuje na nekom projektu, ili zaposleni može da primenjuje veći broj veština na jednom projektu?)

- Ako je odgovor ,,više“, staviti viljušku na liniju koja ulazi u taj entitet; ako je odgovor ,,1“, staviti crticu preko te linije.

- Ponoviti proces dok se ne ispita svaki entitet u odnosu.

Problem 2.1: Za *esp* odnos, koje su kardinalnosti pojedinačnih tipova entiteta?
Problem 2.2: Imamo tri entiteta: automobil, kupac, prodavac. Razmotrimo sledeću situaciju:
Neka postoji ternarni odnos *prodaja* koji povezuje sva tri entiteta: „Kupac K kupio je automobil A od prodavca P.“

prodaja		
A	K	P
Cobra	Sam	Don
Mustang	Sam	Don
Mustang	Sam	Sharon
Cobra	Jenn	Sharon
Mustang	Jenn	Sharon

Pitanje: Ko je prodao automobil Cobra kupcu Sam? (Don)

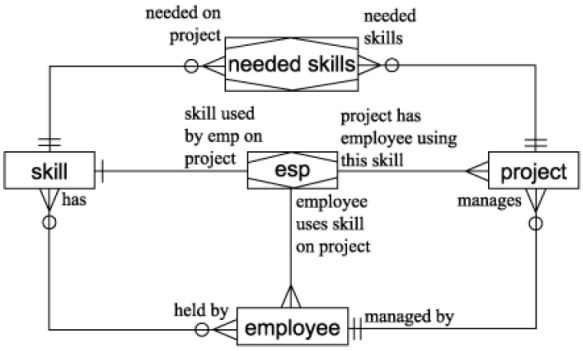
Neka sada imamo tri binarne relacije:

- *kupuje* – kupac K kupuje automobil A.
- *kupuje od* – kupac K kupuje od prodavca P.
- *prodaje* – prodavac P prodaje automobil A.

kupuje		kupuje od		prodaje	
K	A	K	P	P	A
Sam	Mustang	Sam	Don	Don	Cobra
Sam	Cobra	Sam	Sharon	Don	Mustang
Jenn	Mustang	Jenn	Sharon	Sharon	Cobra
Jenn	Cobra			Sharon	Mustang

Pitanje: Ko je prodao automobil Cobra kupcu Sam? (Don ili Sharon)

Primer 3: Interpretirati sledeći dijagram:



Slika 5.7: ER dijagram [6]

Interpretacija odnosa:

- needed (potreban)
  - Jedna veština može da bude potrebna na većem broju projekata, a može da ne bude potrebna ni na jednom.
  - Jednom projektu može da bude potrebna nula ili više veština.
- manages (upravlja)
  - Jedan zaposleni može da upravlja većim brojem projekata a može da ne upravlja ni jednim.
  - Jednim projektom mora da upravlja (rukovodi) neki zaposleni.
- has-skill (ima-veštine)
  - Jedan zaposleni može da ima više veština, a može da nema ni jednu.
  - Jednu veštinu može da poseduje više zaposlenih. Ima i veština koje ne poseduje ni jedan zaposleni.
- used-on (korišćen):
  - Jedan zaposleni koristi jednu veštinu na više projekata.
  - Više zaposlenih može da koristi jednu veštinu na jednom projektu.
  - Jedan zaposleni može da koristi samo jednu veštinu na jednom projektu.
  - Napomena: Ovo čitanje se dobija ,,prekrivanjem“ dve od tri grane ternarnog odnosa.

5.2.1 Nedoumice u ER modeliranju

Videli smo da je ER model dobro prilagođen konceptualnom modeliranju. Ipak, tu ima ozbiljnih nedoumica:

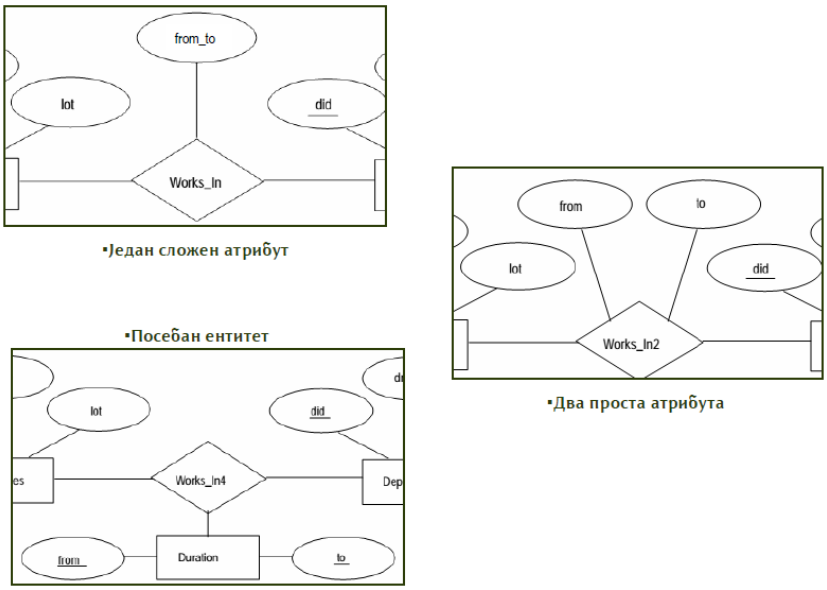
- Da li je nešto bolje modelirati entitetom ili atributom?
- Da li je nešto bolje modelirati entitetom ili odnosom?
- Koji su odgovarajući skupovi (tipovi) entiteta i skupovi (tipovi) odnosa? Da li je u nekom slučaju bolje imati binarni ili ternarni odnos?
- Da li bi trebalo da koristimo agregacije?

Entitet ili atribut?

Ako imamo složen atribut, možemo da ga modeliramo kao:

- jedan složen atribut
- više prostih atributa
- jedan entitet

Na osnovu čega donosimo odluku? Čak i za proste atribute može da se postavi pitanje da li su oni možda ipak entiteti?



Slika 5.8: Entitet ili atribut? [13]

Argumenti za „atribut“:

- Ako jednom entitetu odgovara tačno jedna vrednost „atributa“;
- Ako predstavlja jednostavnu skalarnu vrednost koja opisuje entitet;

- Ako u više entiteta može da ima iste vrednosti, a da one nisu međusobno uslovljene, odnosno ako menjanje vrednosti za jedan entitet ne povlači njeno menjanje za drugi.

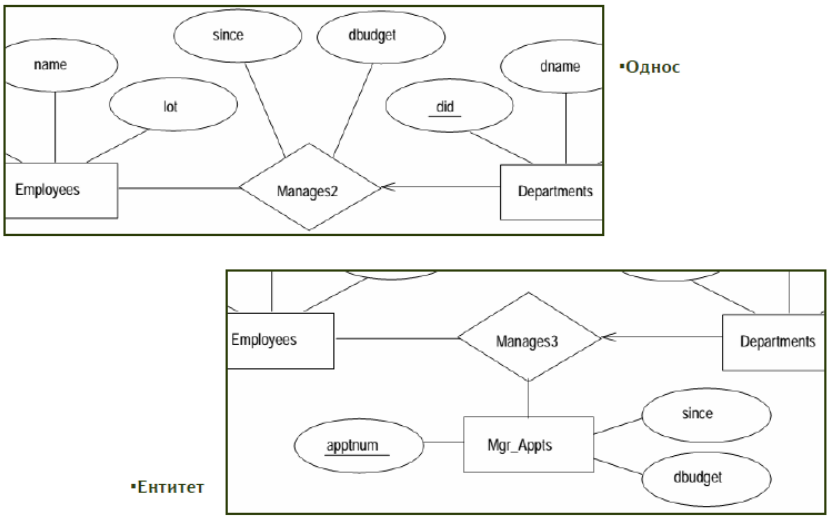
Složen atribut može da ima smisla:

- Ako želimo da sačuvamo internu strukturu, ali ona ima značaja samo interno;
- Ako u više entiteta može da ima iste vrednosti, a da one nisu međusobno uslovljene. Na primer, ako ,,adresa“ obuhvata grad, ulicu i broj, onda više zaposlenih može da ima istu adresu, a da to ne znači da žive zajedno, pa se one i dalje mogu odvojeno menjati;

,,Atribut“ mora da preraste u entitet:

- Ako nekom entitetu odgovara istovremeno više vrednosti tog atributa;
- Ako postoji međuzavisnost vrednosti atributa, na primer, ako više entiteta ima istu vrednost atributa ali tako da ako se on promeni za jedan entitet, onda mora da se promeni i za ostale.

Entitet ili odnos?



Slika 5.9: Entitet ili odnos? [13]

Odnos može da bude “odnos“:

- Ako se svi njegovi atributi odnose strikno na jednu instancu odnosa, tj. nema redundantnosti

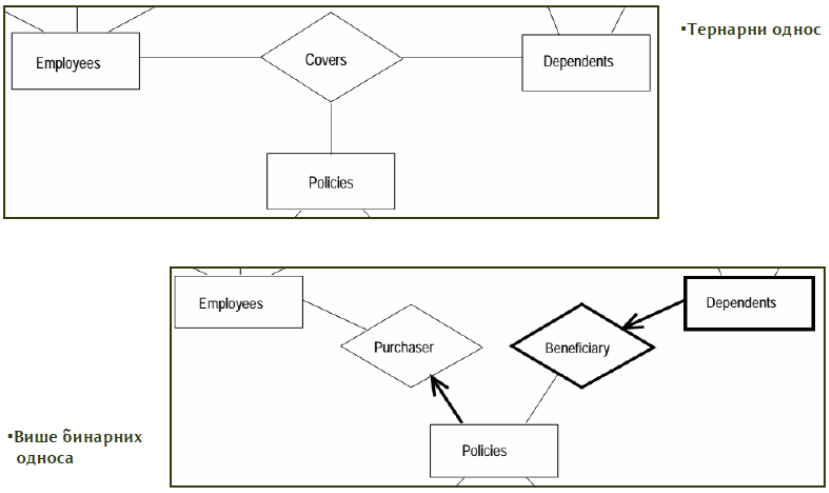
Odnos mora da preraste u ,,entitet“:

- Ako se neki od njegovih atributa odnosi istovremeno na više instanci odnosa

U velikom broju slučajeva je ,,jednako ispravno“ da nešto bude bilo odnos bilo entitet.

Ternarni odnos ili više binarnih odnosa?

Većina složenih odnosa može da se ,,podeli“ na više binarnih agregiranih odnosa. Da li je bolje podeliti ih ili ih ostaviti?



Slika 5.10: Ternarni odnos ili više binarnih odnosa? [13]

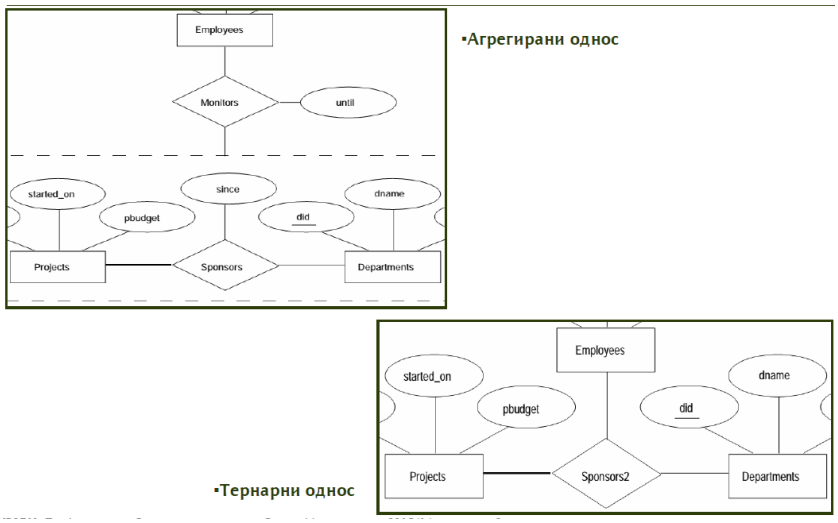
Složene odnose ima smisla podeliti na više jednostavnijih isključivo ako svaki od dobijenih odnosa ima svoj semantički smisao u posmatranom domenu. Čak i tada, ako je semantika očiglednija iz složenog odnosa, onda bi taj složeni odnos trebalo da ostane.

Agregacija ili ternarni odnos?

Ovo pitanje je slično prethodnom:

- Agregirani odnos podrazumeva da se agregirani odnos može posmatrati kao skup entiteta (ili skup odnosa)
- No, to može da bude i veštački napravljeno?

Šta je i kada bolje uraditi?



Slika 5.11: Agregacija ili ternarni odnos? [13]

Agregirani odnosi su bolje rešenje:

- Ako nekada postoji samo deo odnosa, a nekada ceo (tj. nekada samo deo koji se agregira, a nekada pun odnos);
- Ako se deo atributa odnosa ne tiče celine odnosa već samo njegovog dela, koji uključuje manje entiteta.

U ostalim slučajevima obično je bolje da se koriste složeni odnosi.

5.2.2 Prevođenje ER modela na R model

Prevođenje ER modela na relacioni model je u nekim elementima pravolinijsko, ali u nekim drugim nije. Može da zavisi i od semantike i od konteksta.

Neka jednostavna pravila su:

- Svaki entitet se prevodi u relaciju;
- Skoro svaki odnos se prevodi u relaciju;
- Atributi koji predstavljaju skupove podataka prevode se u relacije.

Entiteti

Svaki entitet se prevodi u relaciju. Pri tome:

- Atributi entiteta se prevode u atribute relacije, pri čemu se složeni atributi prevode u više prostih atributa;
- Ključni atributi se prevode u primarni ključ.

Odnosi

Skoro svaki odnos se prevodi u relaciju. Pri tome:

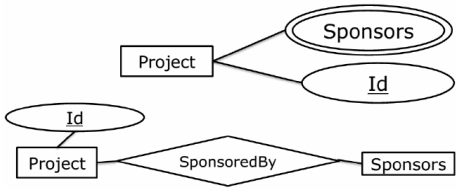
- Atributi odnosa se prevode u atribute relacije, pri čemu se dodaju i ključni atributi uključenih entiteta;
- Primarni ključ dobijene relacije čine primarni ključevi relacija koje odgovaraju uključenim entitetima;
- Primarnom ključu se dodaju i svi ključni atributi odnosa;
- Strani ključevi su primarni ključevi učesnika.

Odnosi, poseban slučaj 1 : 0..1

Ako je odnos 1 : 0..1, onda ne mora da se pravi dodatna relacija za odnos. Atributi odnosa se dodaju relaciji koja odgovara entitetu sa kardinalnošću 1. Ovde treba biti oprezan. Ako odnos nije 1 : 0..1 nego je 0..1 : 0..1, onda se dobija potencijalno redundantno rešenje.

Atributi koji su skupovi podataka

Atributi koji predstavljaju skupove podataka prevode se u entitete, pa u relacije.



Slika 5.12: Atributi koji su skupovi podataka [13]

Hijerarhije

Hijerarhije mogu da se reše na tri osnovna načina:

- Cela hijerarhija u jednu relaciju;
- Svaki entitet u posebnu relaciju, odnosno kao da se radi o „običnom“ osnosu entiteta;
- Svaki entitet - list u posebnu relaciju, ali tako da uključi sve nasleđene atribute.

Moguće je i kombinovanje metoda, za različite delove hijerarhije.

Hijerarhije — sve u jednu relaciju

Cela hijerarhija se preslikava u jednu relaciju koja obuhvata sve atribute koji postoje u hijerarhiji. U ovom slučaju se za svaki entitet koristi samo deo atributa dok ostali imaju nedefinisane (prazne) vrednosti.

Ako je hijerarhija sa preklapanjem onda mora da postoji način da se za svaki konkretan skup entiteta hijerarhije proveri da li mu entitet pripada. Ovo se rešava tako što se:

- ili doda po jedan dodatni atribut,
- ili se proverava da li su ostali odgovarajući atributi neprazni.

Ako je hijerarhija sa pokrivanjem onda je potreban uslov da svaka torka relacije pripada bar jednom konkretnom skupu entiteta. Ovo obično nije problem definisati.

Hijerarhije — svaki entitet posebno

Za svaki entitet se pravi po relacija, sa stranim ključem na neposrednu baznu relaciju. Svaka relacija obuhvata samo one atribute koji su za nju specifični, i još atribute ključa bazne relacije koji se koriste kao strani ključ. Upotreba zahteva česta spajanja i potencijalno je neefikasna, ali je na nivou konceptualnog i logičkog modela prihvatljiva. Ako je hijerarhija sa pokrivanjem onda mora da se obezbedi dodatno sredstvo za proveru pokrivenosti.

Hijerarhije — listovi posebno

Za svaki entitet u listu hijerarhije se pravi po relacija, sa stranim ključem na neposrednu baznu relaciju. Svaka relacija obuhvata atribute koji su za nju specifični, i SVE atribute SVIH baznih klasa. Atributi ključa bazne relacije se koriste kao strani ključ. Upotreba pojedinačnih entiteta je efikasna. Pretraživanje baznog skupa entiteta je neefikasno (unija). Ako je hijerarhija sa preklapanjem onda ovakvo rešenje može da napravi probleme, na primer, mogu da se redundantno ponavljaju delovi entiteta.

Hijerarhije — kombinovano

Poslednja dva metoda mogu da se kombinuju u različitim delovima hijerarhije.

Slabi entiteti

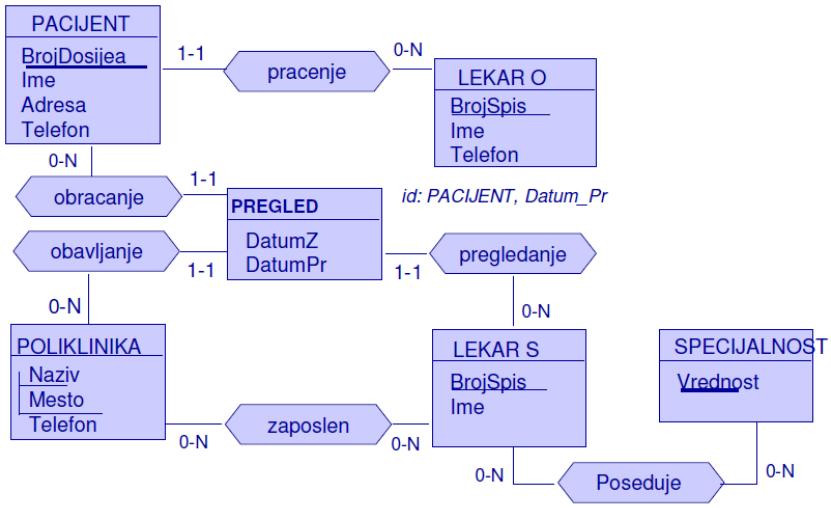
Slab entitet i odgovarajući odnos se modeliraju jednom relacijom tako što se dodaje ključni atribut vezanog jakog entiteta, kao deo primarnog ključa.

5.2.3 Primeri ER konceptualnog i logičkog modeliranja

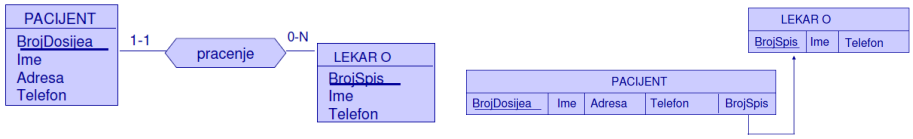
Primer 1: Poliklinika

- Medicinski centar organizuje upravljanje administriranjem pacijenata koji obavljaju preglede na poliklinikama.

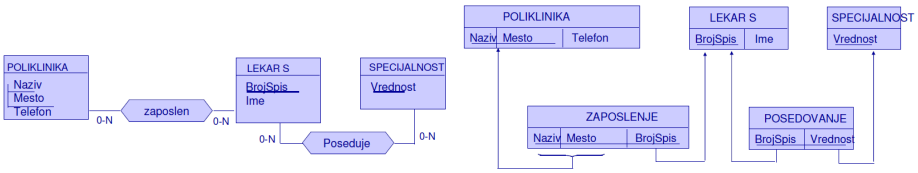
- Pacijent ima jedinstveni broj dosijea, ime, adresu i broj telefona. Jednog pacijenta prati jedan lekar opšte prakse, koji ima broj (iz poznatog spiska brojeva), ime i broj telefona.
- Jedan pacijent može da se obrati poliklinikama radi pregleda kod lekara specijalista. Pregled se obavlja odredenog datuma na odredenoj poliklinici, kod odredenog lekara - specijaliste. Pregledi se zakazuju nekog datuma koji prethodi datumu pregleda.
- Poliklinika se karakteriše imenom, mestom i brojem telefona. Ne postoje dve poliklinike sa istim imenom u istom mestu.
- Jedna poliklinika ima više lekara specijalista. Jedan specijalista može da obavlja preglede u više poliklinika, a pored broja (iz spiska) i imena, karakteriše se i svojom specijalnošcu.



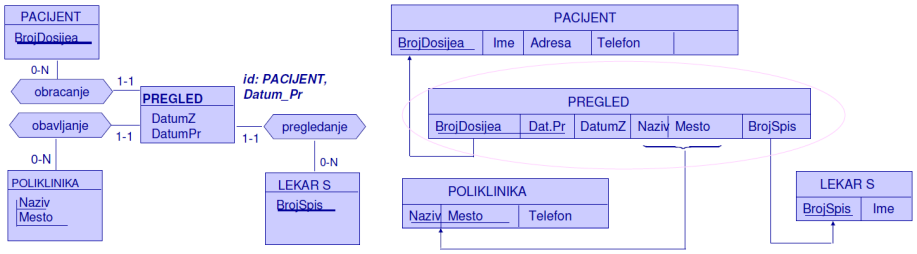
Slika 5.13: ER dijagram



Slika 5.14: Prevođenje relacije *pracenje* iz ER u R model



Slika 5.15: Prevođenje relacija *zaposlen* i *poseduje* iz ER u R model



Slika 5.16: Prevođenje dela ER modela u R model

Uslovi integriteta:

- *Datum_Pr* jednog pregleda je kasniji u odnosu na *Datum_Z*.
- Jedan *LEKAR_S* (specijalista) vrsi *PREGLED* u jednoj *POLIKLINICI* samo ako je zaposlen u toj *POLIKLINICI*.

```
PREGLED.BrojSpis = SOME
                   (select BrojSpis
                    from ZAPOSLENJE
                    where Mesto=PREGLED.Mesto and
                    Naziv=PREGLED.Naziv
                   )
```

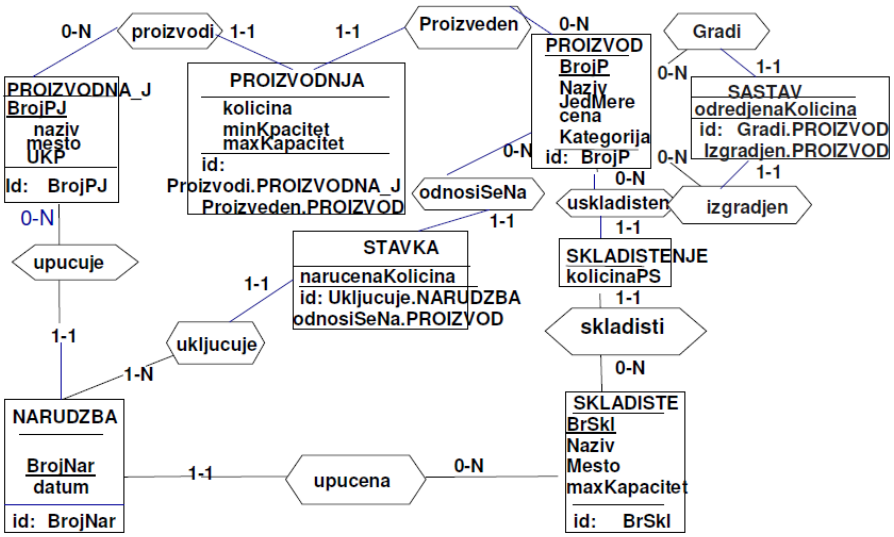
Primer 2: Proizvodnja i zalihe

- Jedna proizvodna organizacija je odlucila da uvede bazu podataka kojom bi poboljšala upravljanje svojom proizvodnjom i zalihama.
- Proizvodna organizacija se sastoji od proizvodnih jedinica.
- Proizvodna jedinica identifikuje se brojem, a ima i ime, mesto i ukupni kapacitet proizvodnje. Može da proizvodi više proizvoda. U svakom trenutku poznata je, za svaku proizvodnu jedinicu i svaki proizvod koji ona proizvodi, količina tog proizvoda koji proizvodi, kao i najmanji i najveći kapacitet proizvodnje.

- Proizvodi se odlažu u skladišta. Skladište se identifikuje brojem i ima ime, mesto i ukupni kapacitet zaliha. U jednom skladištu može da se uskladišti više vrsta proizvoda. Zna se u svakom trenutku količina svakog proizvoda u svakom skladištu. Jedan proizvod može da proizvodi više proizvodnih jedinica, i može da bude usladišten u više skladišta. Svakom proizvodu pridružen je broj koji ga identifikuje.

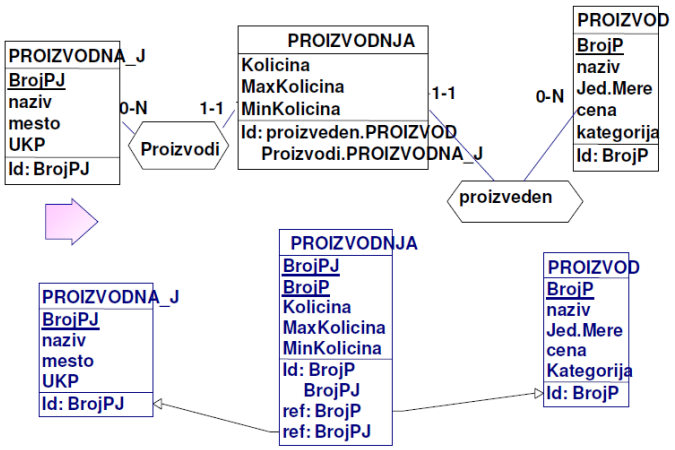
- Proizvod ima ime, jedinicu mere i cenu. Jedinična količina gotovog ili polugotovog proizvoda proizvodi se od određenih količina drugih poluproizvoda i/ili sirovina koje proizvodna jedinica naručuje iz svog skladišta.

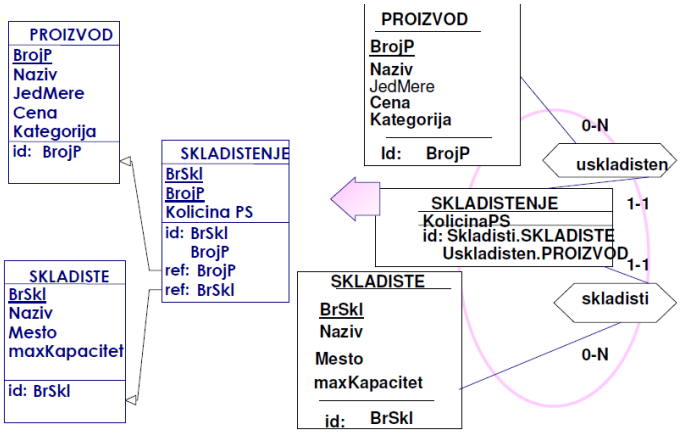
- Narudžba jedne proizvodne jedinice upućena je tačno jednom skladištu, identifikovana je brojem, ima datum i odnosi se na jedan ili više proizvoda sa navedenim količinama.

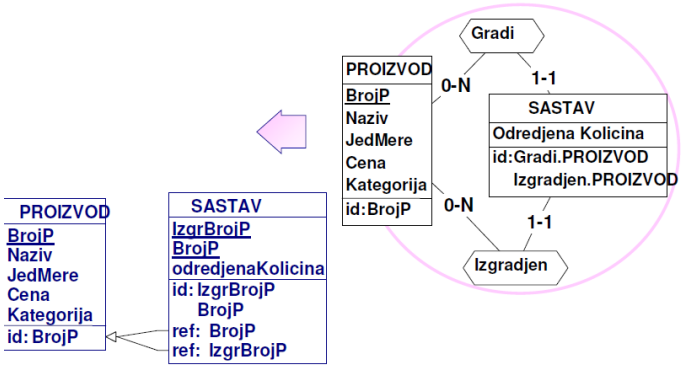


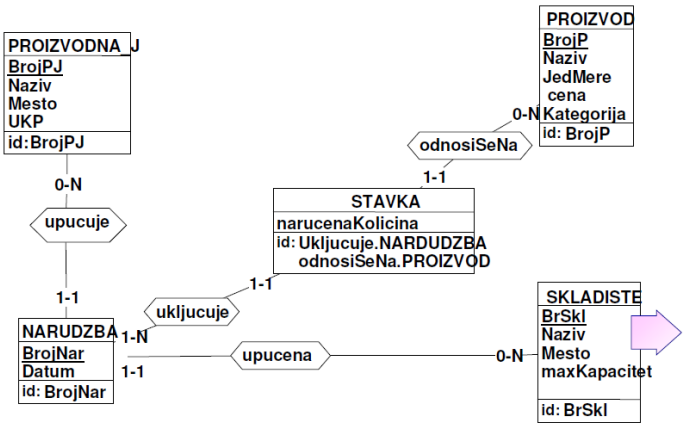
Slika 5.17: ER dijagram

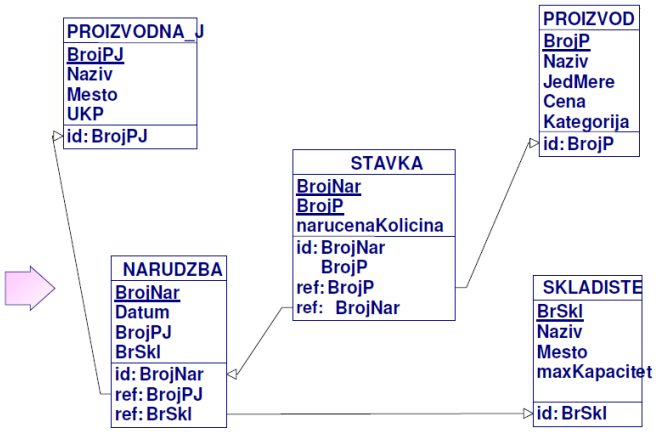
Prevođenje ER modela u R model

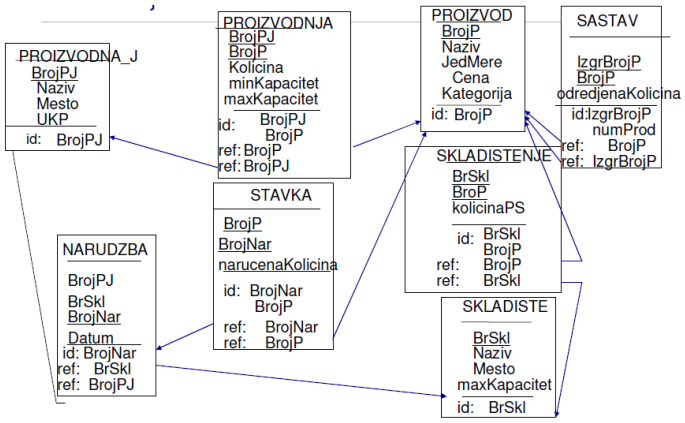







PROIZVODNA_J
BrojPJ
Naziv
Mesto
UKP
id: BrojPJ
0-N
upucuje
1-1
NARUDZBA
BrojNar
Datum
id: BrojNar
1-1
1-N
ukljucuje
1-1
STAVKA
narucenaKolicina
id: Ukljucuje.NARDUDZBA
odnosiSeNa.PROIZVOD
1-1
odnosiSeNa
0-N
PROIZVOD
BrojP
Naziv
JedMere
cena
Kategorija
id: BrojP
1-1
upucena
0-N
SKLADISTE
BrSkl
Naziv
Mesto
maxKapacitet
id: BrSkl


PROIZVODNA_J
BrojPJ
Naziv
Mesto
UKP
id: BrojPJ
PROIZVOD
BrojP
Naziv
JedMere
Cena
Kategorija
id: BrojP
STAVKA
BrojNar
BrojP
narucenaKolicina
id: BrojNar
BrojP
ref: BrojP
ref: BrojNar
NARUDZBA
BrojNar
Datum
BrojPJ
BrSkl
id: BrojNar
ref: BrojPJ
ref: BrSkl
SKLADISTE
BrSkl
Naziv
Mesto
maxKapacitet
id: BrSkl


PROIZVODNA_J
BrojPJ
Naziv
Mesto
UKP
id: BrojPJ
PROIZVODNJA
BrojPJ
BrojP
Kolicina
minKapacitet
maxKapacitet
id: BrojPJ
BrojP
ref: BrojP
ref: BrojPJ
PROIZVOD
BrojP
Naziv
JedMere
Cena
Kategorija
id: BrojP
SASTAV
IzgrBrojP
BrojP
odredjenaKolicina
id:IzgrBrojP
numProd
ref: BrojP
ref: IzgrBrojP
SKLADISTENJE
BrSkl
BroP
kolicinaPS
id: BrSkl
BrojP
ref: BrojP
ref: BrSkl
STAVKA
BrojP
BrojNar
narucenaKolicina
id: BrojNar
BrojP
ref: BrojNar
ref: BrojP
NARUDZBA
BrojPJ
BrSkl
BrojNar
Datum
id: BrojNar
ref: BrSkl
ref: BrojPJ
SKLADISTE
BrSkl
Naziv
Mesto
maxKapacitet
id: BrSkl

6

Dijagrami tabela

Dijagrami tabela (relacija) se često pogrešno nazivaju „ER dijagrami“. Oznake i termini koji se koriste su obično tabele/kolone/ključevi. Prilagođeni su logičkom i (posebno) fizičkom nivou relacionog modela. Semantika odnosa je predstavljena u meri u kojoj to dopušta relacioni model.

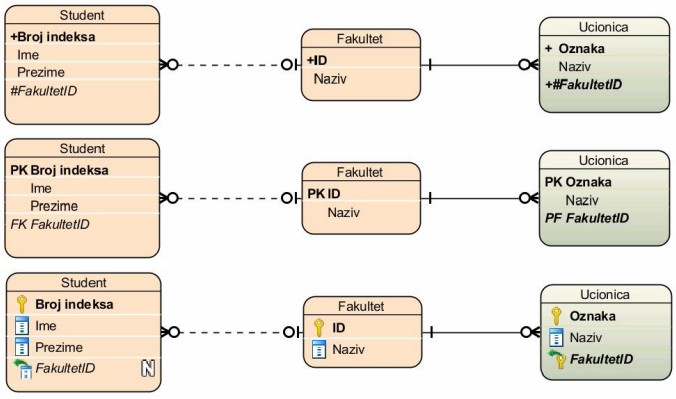
Tabele se predstavljaju pravougaonicima (obično zaobljenim). Odnosi među tabelama (strani ključevi) se predstavljaju linijama.

Tabele

Horizontalnom linijom je pravougaonik podeljen na dva dela:

- gornji deo sadrži naziv tabele,
- donji deo sadrži opise kolona.

Kolone su posebno označene simbolima, slovnim oznakama ili crtežima. Na primer, ako kolona pripada primarnom ključu, onda se koristi simbol +, oznaka PK i crtež ključa. Ako kolona pripada stranom ključu, onda se koristi simbol #, oznaka FK (ili PF, ako je i primarni i strani, tzv slabi entitet), crtež ključa sa strelicom a obično i iskošen naziv. Ako kolona može da sadrži nedefinisanu vrednost, onda se koristi simbol, crtež ili oznaka koji asociraju na slovo N.



Slika 6.1: Tabele - pimeri

Vrste tabela

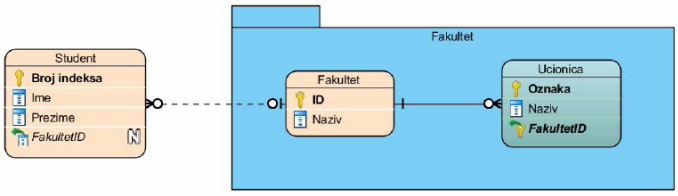
Neki alati i autori razlikuju tabele prema boji

- jednom bojom jaki entiteti
- drugom bojom slabi entiteti
- trećom bojom odnosi
- može da se uvede i više različitih boja, prema kontekstu

Neki alati i autori označavaju tabele različitim simbolima ili crtežima u desnom gornjem uglu. Obe prakse su dobre i olakšavaju razumevanje modela.

Grupisanje

Za grupisanje tabela koriste se pravougaonici koji liče na fascikle. Mogu da se koriste i boje, ali to pravi problem u odnosu na označavanje vrsta tabela.



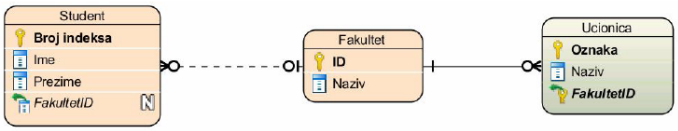
Slika 6.2: Grupisanje - pimeri

Odnosi

Predstavljaju se isprekidanim linijama (puna linija predstavlja odnos slabog entiteta sa matičnim). Na kraju linije se predstavlja kardinalnost što je u duhu UML-a (broj entiteta uz koji stoji oznaka, koji mogu da budu u odnosu sa jednim

entitetom koji je na drugoj strani). Za predstavljanje kardinalnosti koriste se grafičke oznake:

- 0 – kružić
- 1 – uspravna linija
- $*$ - linija se grana u tri kratke linije prema tabeli
- $0-1$, $0-*$, $1-*$ – kombinacije prethodnih slučajeva



Slika 6.3: Odnosi - pimeri

Nivo modela

Osnovni oblik dijagrama najviše odgovara logičkom i posebno fizičkom nivou:

- Na konceptualnom nivou se prilagođava
- Na logičkom nivou može da se prilagođava ali ne mora
- Na fizičkom nivou se kristi u osnovnom obliku

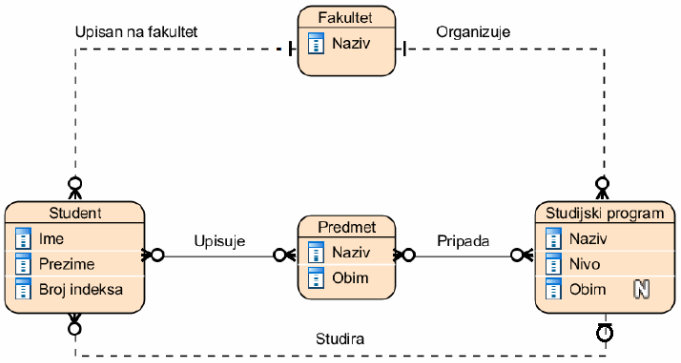
Konceptualni nivo

Dijagram sadrži samo strukturu podataka i odnose. Na ovom nivou, dijagram ne sadrži:

- dodatne kolone surogat ključeva,
- dodatne kolone stranih ključeva,
- oznake ključeva (osim možda jednog kandidat-ključa),
- tipove kolona.

U tako pojednostavljenom obliku, ovakav dijagram može da se koristi i u slučaju logičkog modela.

Podsetimo se, surogat ključevi su primarni ključevi koje generiše i kontroliše sistem. To su jedinstvene reprezentacije entiteta, odnosno nepromenljive i neponovljive vrednosti od kojih svaka odgovara pojedinačnoj n-torci relacije i koje se generišu pri unošenju n-torki kojima se predstavljaju entiteti [10].

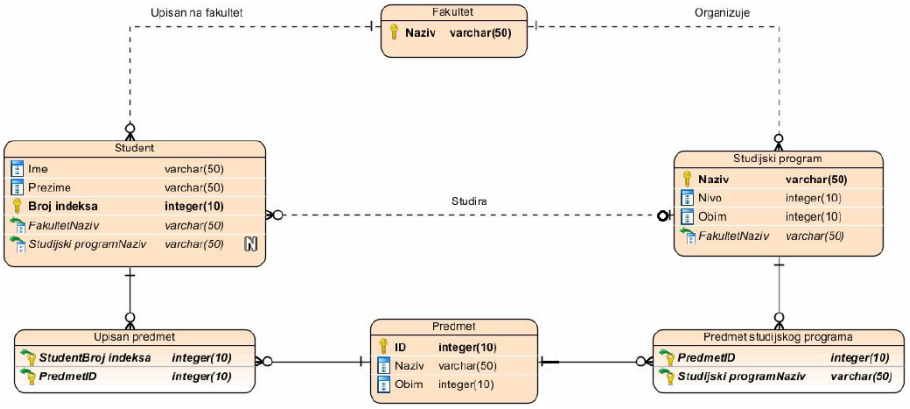


Slika 6.4: Konceptualni nivo - pimeri

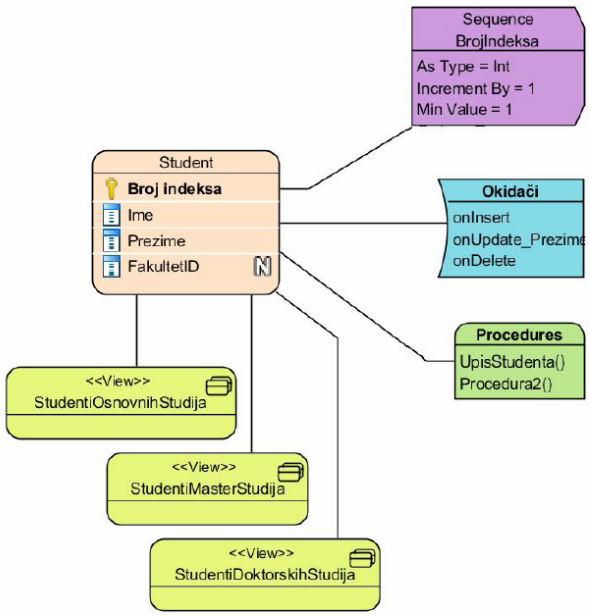
Fizički nivo

Dijagramu se dodaju svi ostali podaci neophodni za preslikavanje u relacioni model:

- kolone surogat ključevi (ako su potrebne)
- kolone vezanih tabela koji čine strane ključeve
- oznake ključeva i vrsta ključeva
- tipovi kolona
- odnosi više-više se često zamenjuju tabelama koje ih modeliraju



Slika 6.5: Fizički nivo - pimeri



Slika 6.6: Fizički nivo - pogledi i ostalo

Ostalo

Alati i autori dodaju i druge elemente kao na primer, uslove integriteta (dodaju se na različite načine, obično u vidu komentara), sekvence, okidače, prostore tabela i slično.

Modifikovani dijagrami

Mnogi alati donose modifikacije, na primer, strani ključevi kao strelice u smeru referisanja.

Sve modifikacije su dopuštene ako su promene intuitivno jasne i sve dok postoji doslednost u okviru projekta.

Rezime

Dijagrami tabela su veoma blizu fizičke implementacije pa se zato češće koriste na nižim nivoima projektovanja. Često se pogrešno nazivaju ,,ER dijagramima“. Predstavlja uobičajeno sredstvo za opisivanje relacionih baza podataka i teško da ima nekoga ko se bavi bazama podataka a da ih ne poznaje.

7

Prečišćavanje sheme

Kao što smo ranije rekli, konceptualni model opisuje prepoznatu semantiku posmatranog domena (na primer, ER model) a logički model obuhvata svođenje apstraktnog i uopštenog konceptualnog modela na konkretan model podataka implementacije (na primer, relacioni model posmatranog domena).

Prečišćavanje sheme podrazumeva analiziranje logičkog modela i doslednu primenu pravila integriteta, pri čemu su funkcionalne zavisnosti osnovno sredstvo primene.

7.1 Anomalije ažuriranja, dodavanja i brisanja

Redundantno čuvanje podataka podrazumeva da se neki podaci ponovljeno čuvaju. Usled toga se javljaju anomalije ažuriranja, anomalije dodavanja i anomalije brisanja.

Anomalije ažuriranja: ako se jedna kopija ponovljenog podatka aužurira, baza će biti nekonzistentna ako se ne ažuriraju i ostale kopije.

Anomalije dodavanja: zapisivanje nekih podataka može da bude nemoguće ako se ne zapišu još neki podaci.

Anomalije brisanja: brisanje nekih podataka može da bude nemoguće bez brisanja još nekih podataka.

Primer redundantnosti

```
UpisanKurs (
    broj indeksa,
    ime i prezime studenta,
    šifra predmeta,
    naziv predmeta,
    ime nastavnika
)
```

U ovom primeru postoji potencijalan problem sa redundantnim podacima o nastavnicima, studentima i predmetima.

211/2015	Горан Петровић	М101	Анализа 1	Милан Марић
211/2015	Горан Петровић	М102	Линеарна алгебра	Петар Симић
212/2015	Тања Митровић	М101	Анализа 1	Милан Марић
212/2015	Тања Митровић	М102	Линеарна алгебра	Петар Симић
211/2015	Тања Митровић	М101	Линеарна алгебра	Милена Савић

Primer anomalije ažuriranja: Nemamo mogućnost nezavisnog ažuriranja vrednosti atributa *ime nastavnika*. Na primer, ako je potrebno promeniti ime nastavnika na predmetu *Analiza 1*, onda se taj podatak ne može promeniti (ažurirati) u jednoj trojci relacije, već se to mora učiniti u svakoj trojci koja se odnosi na predmet *Analiza 1*, tj. onoliko puta koliko je puta neki student upisao taj predmet.

Primer anomalije dodavanja: Nemamo mogućnost unošenja podataka o broju indeksa i imenu i prezimenu studenta dok se ne unese i podatak o nastavniku bar jednog predmeta koji je taj student upisao.

Primer anomalije brisanja: Brisanjem informacije o nastavniku na nekom predmetu mogu se izbrisati, bez takve namere, i informacije o studentima koji su upisali taj predmet.

U rešavanju anomalija dodavanja i brisanja mogu da pomognu nedefinisane vrednosti:

- Ako moramo da dodamo podatak a ne znamo druge potrebne podatke, možemo da navedemo nedefinisane vrednosti (na primer, ako student upisuje kurs a ne znamo nastavnika)
- Ako hoćemo da obrišemo neke redundantne podatke, a da ne brišemo cele redove, možemo da navedemo nedefinisane vrednosti (na primer, ako brišemo podatak o nastavniku na kursu a ne želimo da obrišemo podatke o studentima koji su upisali kurs)

Dekompozicija

Redundantnost se pojavljuje na mestima na kojima postoje ne sasvim prirodne veze između atributa. Uobičajeno sredstvo za rešavanje redundantnosti je dekompozicija. Ideja je da se jedna relacija sa mnogo atributa zameni sa više relacija sa po manje atributa.

Na primer, relaciju:

```
UpisanKurs (
    broj indeksa,
    ime i prezime studenta,
    šifra predmeta,
```

```
    naziv predmeta,
    ime nastavnika
)
```

možemo da podelimo na relacije:

```
UpisanKurs (
    broj indeksa,
    ime i prezime studenta,
    šifra predmeta,
    naziv predmeta
)
```

```
NastavnikKursa(
    šifra predmeta,
    ime nastavnika
)
```

Tako smo rešili samo deo problema.

211/2015	Горан Петровић	M101	Анализа 1
211/2015	Горан Петровић	M102	Линеарна алгебра
212/2015	Тања Митровић	M101	Анализа 1
212/2015	Тања Митровић	M102	Линеарна алгебра
211/2015	Тања Митровић	M101	Линеарна алгебра

M101	Милан Марић
M102	Петар Симић
M101	Милена Савић

Svaki put kada razmišljamo o dekompoziciji pitamo se:

- Koji problem želimo da rešimo predloženom dekompozicijom?
- Da li ga zaista i rešavamo?
- Da li proizvodimo neke nove probleme?

Odgovore na ova pitanja daju:

- analiza funkcionalnih zavisnosti,
- teorijske osnove normalnih formi relacija.

7.2 Funkcionalne zavisnosti

Postojanje određene funkcionalne zavisnosti u relaciji dovoljan je uslov za zamenu te relacije, bez gubljenja informacije, njenim dvema projekcijama, određenim tom funkcionalnom zavisnošću. „Bez gubljenja informacije“ znači da se početna relacija može restaurisati primenom operacije prirodnog spajanja na dobijene projekcije [10].

Definicija: Neka je R relacija i neka su X i Y neprazni skupovi atributa relacije R. Kažemo da skup torki r relacije R zadovoljava funkcionalnu zavisnost $X \rightarrow Y$ ako za svaki par torki $t1$ i $t2$ iz r važi:

$$ako\ je\ t1.X = t2.X\ onda\ je\ t1.Y = t2.Y$$

Pri tome $t1.X$ predstavlja projekciju torke $t1$ na skup atributa X.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	e11

Slika 7.1: Skup torki koji zadovoljava funkcionalnu zavisnost $AB \rightarrow C$ [13]

Ako u relaciji R važi funkcionalna zavisnost $X \rightarrow Y$ onda se kaže da relacija R zadovoljava tu funkcionalnu zavisnost.

U prethodnom primeru skupa torki relacije $UpisanKurs$, ako izostavimo poslednju neispravnu torku, možemo da uočimo sledeće funkcionalne zavisnosti:

- *broj indeksa* $\rightarrow$ *ime i prezime studenta*
- *šifra predmeta* $\rightarrow$ *naziv predmeta*
- *šifra predmeta* $\rightarrow$ *ime i prezime nastavnika*

Integritet ključa

Uslovu ključa odgovara funkcionalna zavisnost oblika:

$$atributi\ ključa \rightarrow svi\ ostali\ atributi$$

Primetimo da je ovo problematično ako ključ sadrži atribute koji mogu da imaju nedefinisane vrednosti. Pitanje koje se u tom slučaju postavlja je: Kako se porede nedefinisane vrednosti?

Integritet jedinstvenosti

Integritet jedinstvenosti je u stvari formalniji integritet ključa.

1. Atributi primarnog ključa ne smeju da imaju nedefinisane vrednosti.
2. Uslovu ključa odgovara FZ

$$atributi\ primarnog\ ključa \rightarrow svi\ ostali\ atributi$$

Da li se problemi vide u ER modelu?

Neke zavisnosti ipak nisu očigledne. Recimo da imamo ovakav slučaj:

- *Student (indeks, ime i prezime, naziv smera)*
- *Smer (oznaka, naziv, trajanje)*

i odnos

- *Student* — *studira* — *Smer* (odnos više-jedan)

Sve zavisnosti u relacijama *Student* i *Smer* su ispravne. Ipak, redundantnost se vidi tek kada se u model uvedu i ključevi:

- *Student (indeks, ime i prezime, naziv smera, oznaka smera)*
- *Smer (oznaka, naziv, trajanje)*

Sada je očigledno da postoji redundantnost atributa *nivo studija* u relaciji *Student*. Ispravno rešenje je dakle:

- *Student (indeks, ime i prezime, oznaka smera)*
- *Smer (oznaka, naziv, trajanje)*

Da bi se redundantnosti videle u ER (tj. konceptualnom) modelu, model mora da se kritički posmatra i stalno deo po deo „prevodi“ u relacioni (tj. logički) model.

Osobine funkcionalnih zavisnosti

Osnovne osobine (Armstrongove aksiome) su kompletne i zatvorene na skupu svih funkcionalnih zavisnosti na nekoj relaciji:

- Refleksivnost
  $X \supseteq Y \implies X \to Y$
- Proširivost
  $X \to Y \implies (\forall Z) XZ \to YZ$
- Tranzitivnost
  $X \to Y \wedge Y \to Z \implies X \to Z$

Koriste se i „složenije“ osobine:

Pri zaključivanju o funkcionalnim zavisnostima koriste se i druga, izvedena pravila:

- Unija
  $X \to Y \wedge X \to Z \implies X \to YZ$
- Dekompozicija
  $X \to YZ \implies X \to Y \wedge X \to Z$

Skup zavisnih atributa

Skup zavisnih atributa $X+$ nekog skupa atributa X je skup svih atributa koji su (posredno ili neposredno) funkcionalno zavisni od atributa iz X

7.3 Normalne forme

Za neku relaciju kažemo da zadovoljava neku normalnu formu ukoliko zadovoljava određena pravila u vezi funkcionalnih zavisnosti definisana za tu normalnu formu. Ova pravila garantuju da neće biti redundantsnoti određenog tipa.
Od posebnog značaja su:

- 1. normalna forma (1NF)
- 2. normalna forma (2 NF)
- 3. normalna forma (3 NF)
- Bojs-Kodova normalna forma (BCNF)

Svaka sledeća normalna forma podrazumeva prethodnu.

1. normalna forma

Relacija je u prvoj normalnoj formi ako svaki atribut može da ima samo atomične vrednosti. Relacioni model to već pretpostavlja ali ER model omogućava i složene vrednosti atributa.

2. normalna forma

Relacija je u drugoj normalnoj formi ako je u prvoj normalnoj formi i ako nijedan atribut, koji ne pripada nijednom kanditatu ključu, nije funkcionalno zavisan od pravog podskupa atributa nekog kandidat ključa. Suština je da u slučaju složenog ključa svi ostali atributi zavise od celog ključa a ne od nekog njegovog dela.

Moguća narušavanja 2. NF:

- Imamo parcijalne zavisnosti ne-ključnog atributa od dela ključa

3. normalna forma

Ako je R relacija i X neki podskup njenih atributa i A neki atribut relacije R, onda je R u trećoj normalnoj formi ako za svaku funkcionalnu zavisnost $X \to A$ na relaciji R važi jedno od:

- A pripada skupu X (tzv. trivijalna zavisnost, $A \subseteq X$)
- X je natključ ($X \to R$)
- A je deo kandidat ključa relacije ($\exists Z(A \subset Z \wedge Z \to R)$)

Moguća narušavanja 3. NF:

- Imamo tranzitivne zavisnosti ne-ključnog atributa od dela ključa

Bojs - Kodova normalna forma

Ako je R relacija i X neki podskup njenih atributa i A neki atribut relacije R, onda je R u Bojs-Kodovoj normalnoj formi ako za svaku funkcionalnu zavisnost $X \rightarrow A$ na relaciji R važi jedno od:

- A pripada skupu X (tzv. trivijalna zavisnost, $A \subseteq X$)
- X je natključ ($X \rightarrow R$)

Odnosno

- svaki atribut relacije je ili deo ključa ili zavisi od celog ključa
- ne postoje tranzitivne zavisnosti

7.4 Dekompozicije relacione sheme

Dekompozicija relacione sheme je zamenjivanje relacione sheme dvema (ili više) relacionim shemama koje zajedno sadrže sve atribute polazne sheme.

Potpunost dekompozicije

Dekompozicija je potpuna (kompletna) ako sadrži dovoljno informacija da se na osnovu dobijenih relacija uvek može rekonstruisati polazna relacija.

Očuvanje zavisnosti

Dekompozicija čuva funkcionalne zavisnosti ako se iz funkcionalnih zavisnosti na dekomponovanim relacijama mogu rekonstruisati sve funkcionalne zavisnosti na polaznoj relaciji.

Normalizacija

Normalizacija logičkog modela baze podataka je svođenje na skup relacija koje su sve u Bojs-Kodovoj normalnoj formi (ili eventualno u 3. NF ili nekoj drugoj NF, zavisno od postavljenih ciljeva).

Efektivan algoritam:

1. Neka je R relacija koja nije u BKNF i neka je X pravi podskup njenih atributa i A jedan atribut koji zavisi od X i tako narušava BKNF. Dekompozicijom je potrebno podeliti R na relacije sa atributuma R–A (atributi iz R koji nisu u A) i XA.

2. Ako R–A ili XA nisu u BKNF, dekomponovati ih dalje rekurzivnom primenom istog algoritma.

Slabosti

U nekim slučajevima ne postoji dekompozicija u BKNF koja čuva zavisnosti.

Primer: Neka imamo relaciju:

$$PrijavaIspita(student, predmet, ispitni\ rok)$$

i dve zavisnosti:

$$student, predmet \rightarrow ispitni\ rok$$

tj. student može da prijavi ispit samo jedanput

$$ispitni\ rok \rightarrow predmet$$

tj. u svakom ispitnom roku se polaže samo jedan predmet,

tada relacija $PrijavaIspita$ nije u BKNF jer predmet zavisi od ispitnog roka, a ispitni rok nije deo ključa. Ako napravimo dekompoziciju, narušićemo prvu zavisnost...

8

Fizički model baze podataka

Fizički model baze podataka je najniži model i on opisuje konkretnu implementaciju baze podataka.Prelaz iz logičkog u fizičko projektovanje podrazumeva promenu u fokusu i veštinama koje su zahtevane. U drugom planu su aspekti važni za konceptualni i logički model. Sada su potrebna drugačija znanja i veštine – više tehnička nego poslovna. Fizičko projektovanje baze podataka se često naziva i ,,modeliranje podataka“ (engl. data modeling).

Strukture podataka

Uobičajeno je da se na fizičkom nivou govori o tabelama i kolonama, a ne o relacijama i atributima. Od značaja je interna (fizička) organizacija podataka (prostori za tabele, kontejneri, stranice, baferi) kao i pomoćne komponente (indeksi). Zadatak SUBP je da zahteve koje korisnik postavlja na konceptualnom nivou nad logičkim tipovima podataka (npr. relacijama i n-torkama) preslika, na unutrašnjem nivou, u zahteve nad internom organizacijom podataka. Terminologija, kao i rešenja pojedinih problema u ovoj oblasti bitno se razlikuju od sistema do sistema, ali principi fizičke reprezentacije i pristupa podacima uglavnom su standardni.

8.1 Procena optrećenja

Da bismo mogli da procenimo opterećenje i performanse, osim logičkog modela podataka moramo da imamo i druge informacije:

- model obrade podataka (engl. process model)
- nestrukturne zahteve
- matricu entiteta i procesa
- zahtevane performanse
- ciljne implementacije SUBP

- eventualna ograničenja prostora
- eventualni razvojni problemi

Model obrade podataka

Model obrade podataka daje detaljnije informacije o ulaznim procesima (kreiranje i ažuriranje vrsta u tabelama) i izlaznim zahtevima (dobijanje podataka iz baze), omogućavajući nam da utvrdimo sledeće:

- Uslove dodavanja novih redova:
  - Koliko redova u proseku (npr. jedna vrsta dnevno)?
  - Koliko redova pri najvećem opterećenju (npr. 100 vrsta u jednoj sekundi)?
  - Da li su primarni ključevi slični i da li zavise od vremena, odnosno, da li će vrste koje su dodate gotovo u isto vreme imati slične primarne ključeve?
- Uslove ažuriranja postojećih redova:
  - Koliko redova u proseku (npr. dnevno)?
  - Koliko redova pri najvećem opterećenju (npr. u sekundi)?
  - Kolika je verovatnoća da se redovi sa sličnim PK istovrmeno koriste/ažuriraju?
- Uslove brisanja redova:
  - Koliko redova u proseku (npr. dnevno)?
  - Koliko redova pri najvećem opterećenju (npr. u sekundi)?
  - Da li se brišu pojedinačno ili u grupama?
- Uslove čitanja redova:
  - Kolika je učestalost čitanja?
  - Koliko redova se čita jednim upitom?
  - Koje kolone se koriste za odabir redova?
  - Koje druge tabele se često koriste zajedno sa posmatranom?

Nestrukturni zahtevi

- Trajanje podataka:
  - Koliko se dugo podaci zadržavaju u tabeli pre brisanja ili arhiviranja?
- Obim podataka:
  - Koliko će redova biti u tabeli pri puštanju u rad i koliko će se broj redova menjati tokom vremena?
- Raspoloživost podataka:

  - Da li su podaci potrebni stalno ili povremeno, koliko često i dugo podaci mogu da budu nedostupni korisnicima?

- Ažurnost podataka:
  - Koliko ažurni moraju da budu podaci koji se koriste?
  - Da li mogu da se razdvoje kopije za menjanje i čitanje, ...?

- Bezbednosni zahtevi:
  - Tajnost (engl. secrecy) – treba da postoji mogućnost skrivanja nekih podataka od korisnika. Na primer, student ne bi trebalo da ima mogućnost uvida u ocene drugih studenata.
  - Integritet (engl. integrity) – treba da postoji mogućnost zabrane menjanja nekih podataka. Na primer, samo profesor može da menja ocenu nekom studentu.
  - Dostupnost (engl. availability) – korisnici bi trebalo da mogu da vide i menjaju sve podatke za koje imaju odgovarajuće dozvole.

Ostale informacije

- Matrica entiteta i procesa (CRUD matrica: create, read, update, delete):
  - Opisuje u kojim se procesima koriste koje tabele (entiteti) i na koji način.

- Zahtevane performanse:
  - Zadaju se obično u obliku vremena odziva korisniku nakon zadavanja akcije.

- Ciljne implementacije SUBP:
  - Potrebno je odrediti ne samo proizvod (na primer, DB2, SQL Server, Access, MySQL), nego i konkretnu verziju kako bi imali saznanja o specifičnim karakteristikama i opcijama koje su obezbeđene tim konkretnim SUBP.

- Eventualna ograničenja prostora:
  - Da li postoje potencijalni problemi sa skladišnim prostorom?

- Eventualni razvojni problemi:
  - Da li su potrebni neki posebni razvojni stručnjaci, na primer, za složene algoritme, izveštaje, analize i slično?

8.2 Osnovni metodi optimizacije baze podataka

Postoji nekoliko opcija u postizanju ciljanih performansi i sve one upadaju u tri široke kategorije:

- Optimizacija na nivou interne organizacije podataka

  - Ostvaruje se kroz upravljanje internom organizacijom podataka, pomoćnim komponentama i resursima
  - Ne menja se logički model (skup tabela i kolona ostaje nepromenjen)

- Optimizacija na nivou upita
  - Vrši se pisanje upita na način koji omogućava njihovo efikasnije izvršavanje
  - Ne menja se logički model

- Optimizacija na nivou strukture podataka
  - Fizička struktura podataka se menja u odnosu na logički model

8.3 Optimizacija na nivou interne organizacije podataka

Optimizacija na nivou interne organizacije podataka podrazumeva upravljanje fizičkom organizacijom podataka (prostori za tabele, stranice, baferi stranica), upravljanje pomoćnim komponentama (indeksima) i upravljanje memorijom.

8.3.1 Fizička organizacija podataka

Logička organizacija kao osnovno mesto čuvanja podataka vidi relaciju, odnosno tabelu. Fizička organizacija ide i dalje od toga. Ozbiljni sistemi omogućavaju veoma precizno upravljanje elementima fizičke organizacije podataka.

Prostor za tabele

Osnovni skladišni prostor se obično naziva prostor za tabele (engl. table space). Jedan prostor za tabele može da sadrži više tabela a u nekim sistemima jedna tabela može da bude i u više prostora za tabele.

Na nivou prostora za tabele definišu se:

- veličina fizičke stranice,
- način i uslovi baferisanja stranica,
- fizički uređaji (diskovi, particije, direktorijumi, fajlovi) koji čine taj prostor za tabele (tzv. kontejneri).

Stranica

Stranica je osnovni element fizičkog zapisa tabele. Svaka tabela, kao i svaki indeks, sastoji se od stranica. Veličina stranice je određena prostorom za tabele.

Ako su stranice velike, to znači da:

- mogu da sadrže više redova,
- manje se traži po disku,
- manja je dubina indeksa,

- redovi tabele mogu da budu veći (obično jedan red mora da stane u stranicu).

Ako su stranice male, to znači da:

- veća je iskorišćenost prostora na disku,
- manje je nepotrebnog čitanja sa diska.

Bafer za stranice

Bafer za stranice (engl. buffer pool) je memorijski prostor predviđen za čuvanje kopije dela stranica jednog prostora za tabele, radi omogućavanja bržeg pristupa podacima. Što je bafer za stranice veći, to je broj pristupa disku manji.

U idealnom slučaju cela baza podataka je u memoriji. Ako to nije moguće, teži se da u memoriji budu bar indeksi (ili bar prvih nekoliko nivoa indeksa) i manje tabele (šifarnici).

Dobro konfigurisanje prostora za tabele i bafera za stranice može da bude od presudnog uticaja na performanse.

Drugi važni koncepti

Fizička organizacija podataka počiva na još mnogo važnih koncepata kao što su:

- particionisane tabele,
- kompresija podataka,
- katanci,
- različiti drugi koncepti, često specifični za određene implementacije.

Particionisane tabele

Sadržaj tabele može da se podeli u više fizičkih celina (particija) na osnovu vrednosti ključa. Na primer, svi podaci o studijama bi mogli da se particionišu po godini upisa studenta čime se omogućava lakše arhiviranje starih podataka, različito upravljanje baferisanjem novih i starih podataka i slično.

Kompresija podataka

Savremeni SUBP obično omogućavaju da se podaci komprimuju, kako bi zauzimali manje mesta. To obično ne utiče na način upotrebe podataka. Novi procesori omogućavaju da se komprimovanje i dekomprimovanje takvih podataka vrši „u hodu“, bez vidljivog uticaja na performanse.

Katanci

Uobičajen način implementiranja izolovanosti i transkacija je pomoću mehanizma katanaca. Postavljanje katanca na objekat (zaključavanje objekta) je postupak koji obezbeđuje transakciji pristup objektu, i kojim transakcija istovremeno sprečava druge transakcije da pristupe tom objektu. Svaka transakcija na kraju svog izvršavanja otključava sve objekte koje je sama zaključala.

Svaki katanac ima:

- objekat koji se zaključava,
- trajanje,
- vrstu katanca (deljivi ili ekskluzivni; pri tom više transakcija može da postavi deljivi katanac na jedan objekat, ali najviše jedna može da postavi ekskluzivni katanac).

Objekat koji se zaključava može biti:

- vrednost (atribut),
- red tabele,
- stranica tabele,
- cela tabela,
- prostor za tabele,
- indeks.

Eskalacija katanaca

Veličina katanca može da bude različita. Tako katanac može da zaključava jedan red ili više redova ili jednu stranicu ili više stranica.

Veliki broj katanaca može da značajno uspori rad. Zato je broj katanaca ograničen (bilo brojem bilo količinom memorije koja je rezervisana za katance).

Kada se prevaziđe dopušten broj katanaca dolazi do eskalacije katanaca. Tada se više manjih katanaca zamenjuje jednim većim. Na primer, više katanaca na redovima se zamenjuje katancem na stranici. Ili, više katanaca na stranicama se zamenjuje katancem na tabeli.

8.3.2 Indeksi

Indeksi su pomoćne strukture podataka koje omogućavaju brže pristupanje podacima, odnosno brže pretraživanje po unapred izabranom ključu. Svaka tabela može da ima više indeksa sa različitim ključevima.

Definicija indeksa obuhvata sledeće:

- kolone koje čine uslov uređivanja, odnosno ključ pristupanja,
- odgovor na pitanje da li je indeks jedinstven ili nije,
- odgovor na pitanje da li je indkes uređujući (grupišući) ili ne,
- vrstu (strukturu) indeksa.

Jedinstveni indeksi

Indeksi koji ne dozvoljavaju ponavljanje istog ključa nazivaju se jedinstveni indeksi. Koriste se i kao sredstvo za implementiranje integriteta ključa (jedinstvenosti).

Grupišući indeksi

Podrazumevaju da su redovi u tabeli poređani u odgovarajućem poretku. Dozvoljeno je imati najviše jedan takav indeks po tabeli. Značajno ubrzavaju izdvajanje ,,podsekvence“ redova u datom poretku.

Na primer: Izlistati sve studente čiji je broj indeksa između 50 i 150.

Ovakvi indeksi se nazivaju još i ,,uređujući“ (engl. sorting ili clustering) indeksi.

Kandidati za grupišuće indekse su kolone:

- koje se često traže u opsezima,
- po kojima se često uređuje rezultat,
- koje pripadaju stranom ključu po kome se najčešće vrši spajanje,
- kolone primarnog ključa.

Prednosti koje donose ovi indeksi su višestruko ubrzano čitanje nizova redova po uslovu, a *slabosti* su dodatno usporeno održavanje i to što ne ubrzavaju pristupanje pojedinačnim redovima.

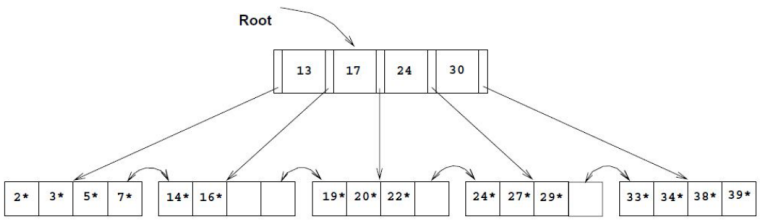
Indeksi sa strukturom B-stabla

Ideja je da se podaci čuvaju u balansiranom drvetu:

- svaki podatak je u sadržan u listu jednake dubine,
- čvorovi i listovi sadrže po više podataka,
- veličina čvorova odgovara stranici diska.

Danas uobičajeni indeksi su upravo indeksi sa strukturom B-stabla (ako se ne navede drugačije).

Prednosti ovih indeksa su jednostavni i efikasni algoritmi za održavanje, a *slabosti* su što ne rade posebno dobro ako je mnogo redova a malo različitih vrednosti ključa, odnosno ako ima mnogo ponavljanja ključa.



Slika 8.1: Indeksi sa strukturom B - stabla, ilustracija

Bit-mapirani indeksi

Indeks se sastoji od niza vrednosti ključa. Iza svake vrednosti ključa sledi niz bitova, za svaki red tabele po jedan. Pri tome bit je jedan ako odgovarajući red ima baš tu vrednost ključa.

Prednost bit-mapiranih indeksa je u slučaju postojanja relativno malo različitih vrednosti ključa, jer tada ovakva struktura postaje efikasnija od B-stabla, a *slabost* je u slučajevima sa mnogo različitih vrednosti ključeva, kada indeks postaje veoma velik, a time i slabo efikasan.

Heš tabele

Heš tabele su alterntiva klasičnim indeksima. Ovde se računaju heš vrednosti na osnovu ključnih atributa, a onda se pomoću dobijene vrednosti neposredno pristupa podacima. Alternativa ovom pristupu bi bilo da se vrši indeksiranje po heš vrednostima.

Prednost je efikasnije pristupanje pojedinačnim redovima sa tačno zadatim ključem, a *slabost* je ta što heš tabele nisu dobre za sekvencijalno pristupanje većem broju redova ili ako operator poređenja nije jednakost.

Tabele bez indeksa

U slučaju malih tabela, lakše je uvek pretražiti sve redove nego koristiti dodatne strukture. Prihvatljiva granica veličine zavisi od implementacije i upotrebe (obično je od nekoliko redova do najviše par stotina redova).

Indeksi sa dodatnim kolonama

Savremeni SUBP omogućavaju da se indeksu osim kolona ključa dodaju i još neke kolone. Ako upit zahteva samo kolone ključa i te dodatne kolone, onda ne moraju ni da se čitaju stranice tabele.

Indeksi, rezime

Prednosti indeksa

- Omogućavaju brži pristup konkretnim redovima tabele
- Ako čitanje zahteva samo kolone sadržane u indeksu, onda tabeli ne mora ni da se pristupa (tzv. indeksi sa dodatnim kolonama)

Slabosti indeksa

- Indeksi moraju da se održavaju jer svaka operacija dodavanja i brisanja, kao i operacije menjanja kolona koje čine ključ indeksa, zahtevaju ažuriranje indeksa
- Zauzimaju prostor
- Mogu da povećaju broj zaključavanja ili granularnost katanaca
- Podižu cenu u slučaju reorganizovanja ili prenošenja tabela

Koliko indeksa je idealno?

Da li neki indeks više koristi ili šteti zavisi od odgovora na sledeća pitanja:

- Da li se indeks uopšte koristi u upitima?
- Koliko upotreba indeksa doprinosi performansama upita u kojima se koristi?
- Koliko su takvi upiti česti?

Idealan broj i vrsta indeksa zavise od vrste, namene i strukture tabele i baze podataka, kao i od načina upotrebe.

8.3.3 Upravljanje memorijom

Važno je da se pri administriranju SUBP dobro upravlja memorijom. Ako se memorija ne koristi dovoljno, ili se koristi pogrešno, suviše će se pristupati disku. SUBP ne koristi virtualnu memoriju za pristupanje stranicama baze podataka, već samo bafere stranica.

Upotreba memorije

SUBP koristi memoriju na mnogo različitih načina, što uključuje:

- glavne i pomoćne bafere stranica
- liste katanaca
- keš kataloga baze podataka (podataka o strukturi i organizaciji podataka)
- keš paketa (podataka o programima)
- interni hip baze podataka (koristi se pri izračunavanju upita)
- hip pomoćnih alata (za uređivanja, pretraživanja, rezervne kopije, ...)
- memoriju agenata (svaka uspostavljena sesija (engl. connection) sa bazom ima svoj prostor)

Na primeru DB2

Memorija se deli na četiri celine:

- Deljena memorija instance (čvora)
- Deljena memorija baze podataka
- Deljena memorija grupe aplikacija
- Privatna memorija agenta

Deljena memorija instance (čvora):

- memorija instance
- hip za praćenje rada i stanja (monitori)

- audit baferi
- baferi za brzu komunikaciju među čvorovima

Deljena memorija baze podataka obuhvata većinu ranije navedenih stavki:

- glavni baferi stranica
- skriveni (pomoćni) baferi stranica
- hip za liste katanaca
- keš kataloga baze podataka (podataka o strukturi i organizaciji podataka)
- keš paketa (podataka o programima)
- hip pomoćnih alata (za uređivanja, pretraživanja, rezervne kopije, ...)
- baferi za dnevnik transakcija

8.4 Optimizacija upita

8.4.1 Plan izvršavanja upita

Da bi se neki upit izvršio nad bazom podataka, najpre se pravi ,,plan izvršavanja upita“ (engl . execution plan) Ovaj plan najčešće obuhvata sledeće:

- redosled koraka,
- operacije koje se izvršavaju u pojedinim koracima,
- strukture podataka koje se upotrebljavaju,
- način svakog pojedinačnog pristupanja podacima,
- procenjenu cenu svakog od koraka i celog posla.

Većina savremenih SUBP omogućava korisniku da sagleda plan izvršavanja pre nego što se pokrene izračunavanje upita.

Način računanja cene koraka

Pri računanju cene pojedinačne operacije uzimaju se u obzir sledeće informacije:

- cena uređaja (čitanje stranice),
- broj redova u tabeli,
- popunjenost stranica,
- selektivnost upita (procena broja redova koji zadovoljavaju uslov),
- cena same operacije.

Većina tih informacija je tačna samo ako su ažurni statistički podaci o sadržaju baze podataka.

Vrste operacija (primer DB2)

Najčešće operacije su:

- TBSCAN – čitanje (pretraživanje) redova podataka direktno iz tabele.
- IXSCAN – čitanje pokazivača na redove iz indeksa
- RIDSCN – čitanje svih pokazivača na redove iz indeksa (ili međurezultata)
- FETCH – čitanje kolona iz poznatih redova
- HSJOIN – heš spajanje, bez prethodnog uređivanja tabela po uslovu spajanja
- NLJOIN – ugneždeno spajanje, unutrašnja tabela se pretražuje za svaki red spoljašnje
- MSJOIN – merge spajanje, kada su obe tabele uređene po uslovu spajanja
- SORT – uređivanje redova, opciono bez ponavljanja

Plan izvršavanja upita, primeri

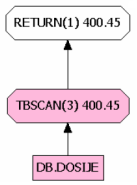
Svi primeri će biti prikazani nad sledećim tabelama baze:

```
CREATE TABLE DB.DOSIJE (
    INDEKS INTEGER NOT NULL,
    ID_SMERA INTEGER NOT NULL,
    STATUS VARCHAR(20) NOT NULL,
    IME VARCHAR(20) NOT NULL,
    PREZIME VARCHAR(25) NOT NULL,
    ...
    PRIMARY KEY (INDEKS)
)
```

```
CREATE TABLE DB.SMER (
    ID INTEGER NOT NULL ,
    OZNAKA VARCHAR(8) NOT NULL ,
    NAZIV VARCHAR(50) NOT NULL ,
    SEMESTARA SMALLINT NOT NULL WITH
    DEFAULT 6 ,
    BODOVI SMALLINT NOT NULL WITH
    DEFAULT 180 ,
    ID_NIVOA SMALLINT NOT NULL ,
    ZVANJE VARCHAR(40) NOT NULL WITH
    DEFAULT ' ',
    OPIS LONG VARCHAR,
    PRIMARY KEY (ID)
)
```

Primer 1: Plan izvršavanja upita pri čitanju cele tabele.

```
select indeks, ime, prezime
from db.dosije
```



укупна процењена цена

процењена процесорска цена

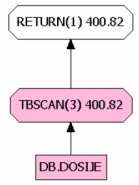
процењена цена У/И операција

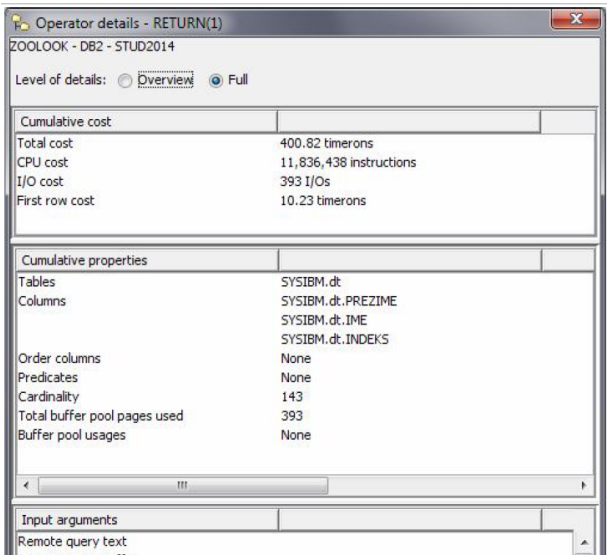
процењен број редова

процењен број страница

Primer 2: Plan izvršavanja pri pretraživanju tabele bez indeksa.

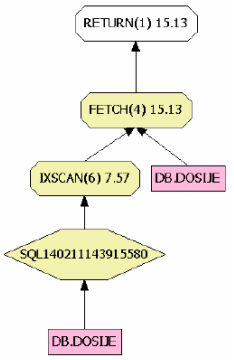
```
select indeks, ime, prezime
from db.dosije
where ime = 'Marko'
```





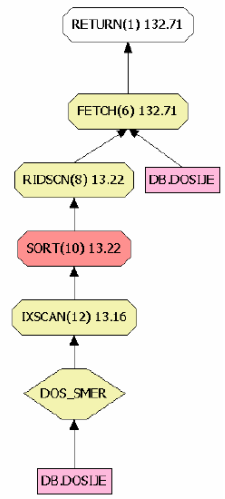
Primer 3: Plan izvršavanja pri pretraživanju tabele sa indeksom - primarni ključ.

```
select indeks, ime, prezime
from db.dosije
where indeks = 20100201
```



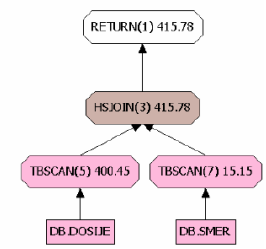
Primer 4: Plan izvršavanja pri pretraživanju tabele sa indeksom.

```
select indeks, ime, prezime
from db.dosije
where id_smera = 201
```



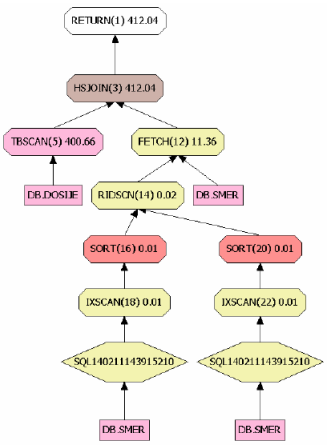
Primer 5: Plan izvršavanja pri spajanju dve tabele.

```
select ime, prezime, s.naziv
from db.dosije d
    join db.smer s
        on d.id_smera = s.id
```



Primer 6: Plan izvršavanja upita.

```
select ime, prezime, s.naziv
from db.dosije d
    join db.smer s
        on d.id_smera = s.id
where s.id in (201,205)
```

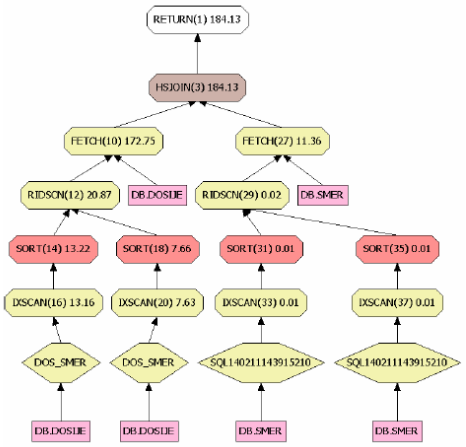


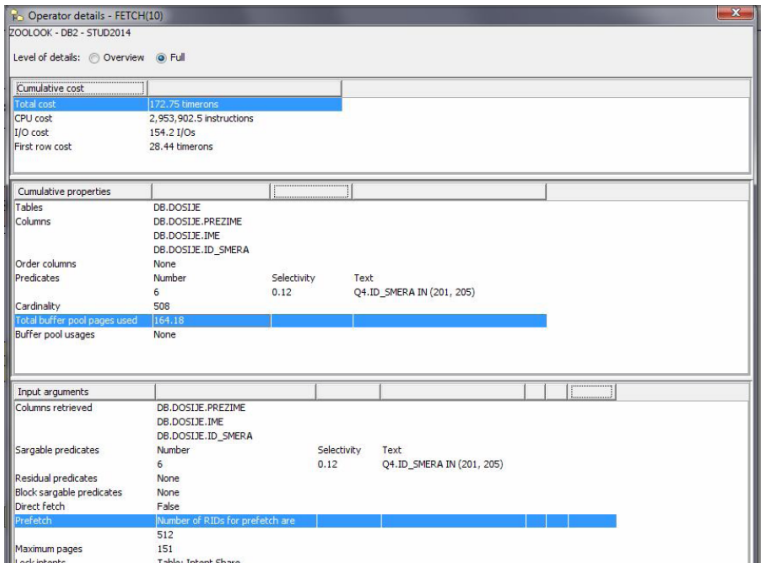
Primer 7: Plan izvršavanja upita.

```
select ime, prezime, s.naziv
from db.dosije d
    join db.smer s
        on d.id_smera = s.id
where s.id in (201,205)

Исти упит, али додајемо:

CREATE INDEX DB.DOS_SMER
ON DB.DOSIJE(
    ID_SMERA ASC,
    INDEKS ASC
)
```

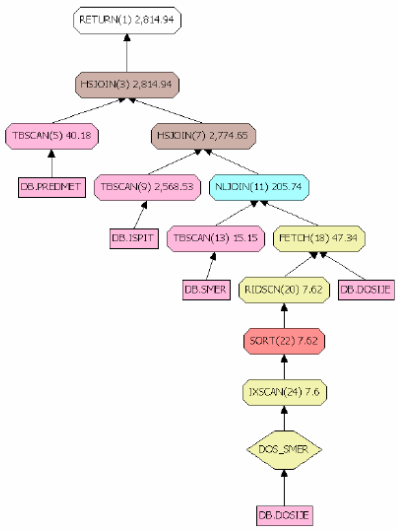




Primer 8: Plan izvršavanja upita - nekada se koriste indeksi.

Сви испити положени у школској 2012/13. години на смеру Информатика

```
select d.ime, d.prezime, p.naziv,
    ip.ocena
from db.dosije d
    join db.smer s
        on d.id_smera = s.id
        and s.naziv='Informatika'
    join db.ispit ip
        on ip.indeks = d.indeks
        and ip.ocena > 5
        and ip.godina = 2012
    join db.predmet p
        on ip.id_predmeta = p.id
```



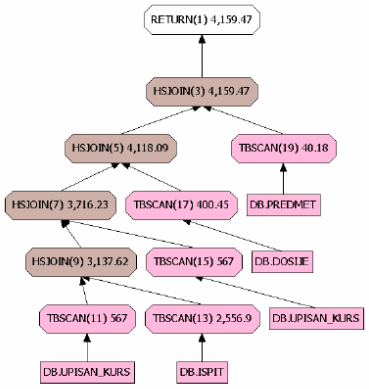
Primer 9: Plan izvršavanja upita - nekada se ne koriste indeksi iako bismo pretpostavili.

Студенти који су положили испит после поновљеног слушања предмета

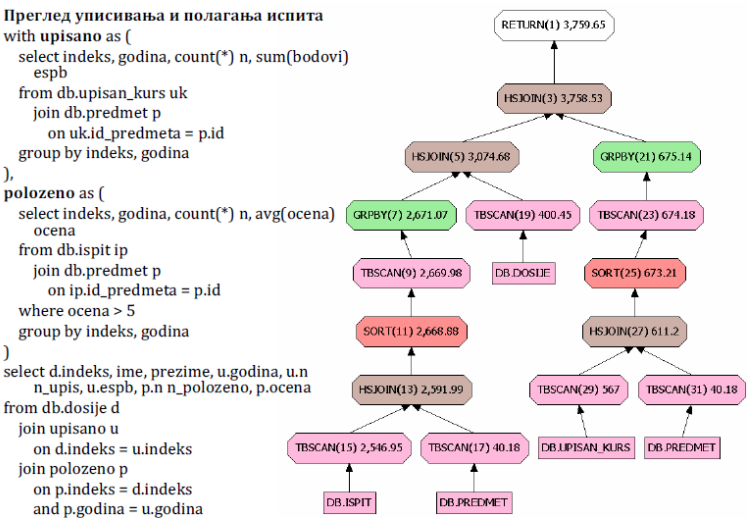
```
select d.ime, d.prezime, p.naziv
from db.dosije d
  join db.ispit ip
    on ip.indeks = d.indeks
    and ip.ocena > 5
  join db.upisan_kurs uk
    on ip.indeks = uk.indeks
    and ip.id_predmeta = uk.id_predmeta
    and ip.godina = uk.godina
    and ip.semestar = uk.semestar
  join db.predmet p
    on p.id = uk.id_predmeta
where exists (
  select *
  from db.upisan_kurs uk2
  where uk2.indeks = uk.indeks
    and uk2.id_predmeta = uk.id_predmeta
    and uk2.godina < uk.godina
)
```

На свим табелама постоје индекси, али се не користе...

Испити су најважнији, али рестрикција није индексирана...



Primer 10: Plan izvršavanja upita - nekada se ne koriste indeksi iako bismo pretpostavili.



8.4.2 Optimizacija upita

Na početku izlaganja o optimizaciji upita, biće reči o sistemskom katalogu kao sistemskoj bazi podataka o bazama podataka. Potreba za ovim izlaganjem postoji zbog važnosti uloge koju katalog ima u procesu optimizacije upita.

Katalog

Katalog, kao sistemska baza podataka (ili njen deo), sadrži informacije o raznim objektima u sistemu kao što su bazne tabele, pogledi, indeksi, baze podataka, planovi izvršavanja upita. Karakteristično za katalog je da obično uključuje tabelu koja sadrži po jednu n-torku za svaku tabelu u sistemu, uključujući i sistemske tabele, zatim tabelu svih atributa svih tabela u sistemu, tabelu svih indeksa svih tabela u sistemu, kao i tabele pogleda, baza podataka, uslova integriteta, stranih ključeva, autorizacije, optimizovanih upita, sinonima, itd. Ove tabele sadrže informacije o imenu objekta (npr. tabele, atributa, indeksa, baze, pogleda), vlasniku, broju atributa (tabele), tabeli za koju je (atribut, indeks) vezan, tipu atributa, itd. S obzirom da je sistemski katalog relacionog sistema – relaciona baza podataka, ona se može pretraživati istim upitnim jezikom kao i korisničke baze podataka. Međutim, tabele kataloga ne mogu se ažurirati iskazima ažuriranja upitnog jezika, jer to može da bude vrlo opasno po podatke u bazi i pristup podacima [10].

Optimizacija upita je manuelno ili automatsko odabiranje najpovoljnijeg plana izvršavanja upita radi postizanja boljih performansi. Većina savremenih SUBP raspolaže solidnim optimizatorima upita. Optimizacija upita je komponenta procesora upitnog jezika koja je neophodna, bar kod velikih sistema i za velike baze podataka, da bi sistem uopšte mogao da zadovolji traženu efikasnost. Posebnu pogodnost kod optimizacije upita pružaju relacioni sistemi, s obzirom na dovoljno apstraktni nivo relacionih izraza kojima se upiti zadaju.

Automatska optimizacija daje značajne prednosti u kvalitetu, jer čovek često nije u mogućnosti da postigne kvalitet optimizacije koji postiže sistem. Ova prednost sistema nad čovekom je posledica činjenice da sistem ,,ima uvid“ u vrednosti podataka (a ne samo u njihovu strukturu) koji čovek – korisnik nema, kao i, s obzirom na brzinu rada sistema, posledica mogućnosti proizvođenja većeg broja alternativnih načina izvršavanja upita i procene njihove efikasnosti. Optimizator koristi statističke informacije iz sistemskog kataloga, formule za procenu veličine međurezultata i cene operacija niskog nivoa [10].

Primer 11

Posmatrajmo sledeći upit:

```
select ime, prezime, s.naziv
from db.dosije d
  join db.smer s
    on d.id_smera=s.id
where d.ime='Marko'
```

Neka baza podataka sadrži 100 smerova (vrsta u tabeli db.smer) i 10000 dosijea (vrsta u tabeli db.dosije) od kojih se 20 odnosi na one sa imenom 'Marko'. Jedan način izvršavanja ovog upita, bez primene optimizacije, mogao bi se sastojati u sledećem nizu radnji.

1. Izračunati Dekartov proizvod tabela db.dosije i db.smer; pri tom se čita 10100 vrsta sa diska i konstruiše se privremena tabela sa 1000000 vrsta, koja, zbog veličine, mora takođe da se upiše na disk.

2. Izvršiti restrikciju dobijene privremene tabele po WHERE logičkom uslovu, tj. po logičkom izrazu *d.ime='Marko'*; ona uključuje sekvencijalno čitanje 1000000 vrsta sa diska i konstrukciju privremene tabele sa samo 20 vrsta, koja se može zadržati u unutrašnjoj memoriji.

3. Projektovati privremenu tabelu dobijenu u koraku 2 na atribute ime, prezime i s.naziv, čime se dobija najviše 60 podataka.

Jasno je da je ovaj postupak, mada neposredno diktiran sintaksom upita, najmanje efikasan. Znatno efikasnije izvršavanje upita dao bi sledeći postupak.

1'. Izvršiti restrikciju tabele db.dosije po uslovu *d.ime='Marko'*, čime se sekvencijalno čita 10000 vrsta i proizvodi privremena tabela sa 20 vrsta koja se može zadržati u unutrašnjoj memoriji.

2'. Spojiti privremenu tabelu dobijenu u koraku 1' sa tabelom db.smer po jednakim vrednostima atributa id_smera i id; time se čita 100 vrsta tabele db.smer sa diska i proizvodi privremena tabela sa 20 vrsta koja ostaje u unutrašnjoj memoriji.

3'. Projektovati privremenu tabelu dobijenu u koraku 2' na atribute ime, prezime i s.naziv.

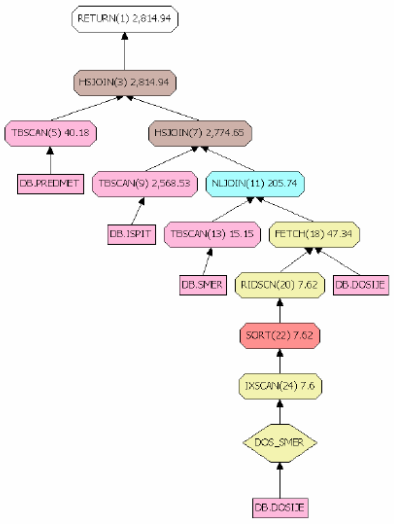
Ako se broj ulazno/izlaznih operacija nad diskom (broj čitanja stranica sa diska odnosno upisa stranica na disk) uzme kao mera efikasnosti (što je realno s obzirom da te operacije oduzimaju najveći deo vremena), onda je drugi postupak 200 puta efikasniji od prvog [10].

U naredna tri primera se koriste različiti a ekvivalentni upiti, za koje SUBP računa iste planove izvršavanja.

Primer 12: Plan izvršavanja upita.

Сви испити положени у школској 2012/13. години на смеру Информатика

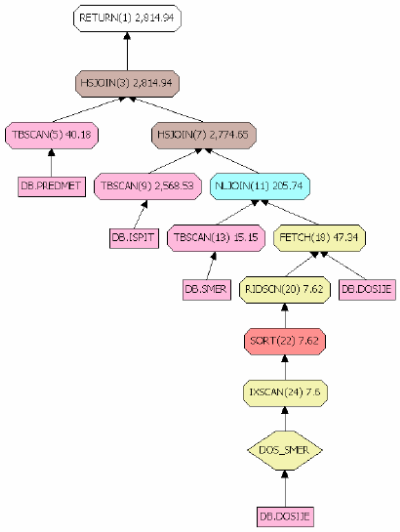
```
select d.ime, d.prezime, p.naziv,
    ip.ocena
from db.dosije d
    join db.smer s
        on d.id_smera = s.id
        and s.naziv='Informatika'
    join db.ispit ip
        on ip.indeks = d.indeks
        and ip.ocena > 5
        and ip.godina = 2012
    join db.predmet p
        on ip.id_predmeta = p.id
```



Primer 13: Plan izvršavanja upita.

Сви испити положени у школској 2012/13. години на смеру Информатика

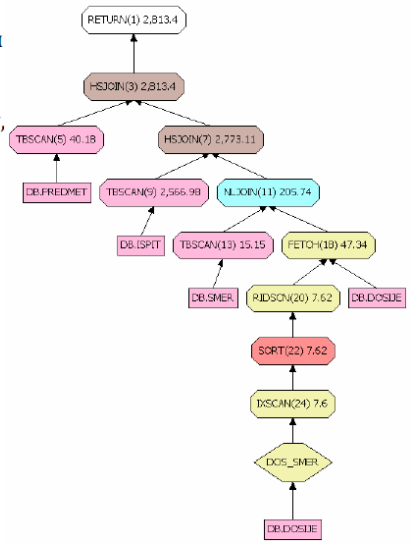
```
with ip as (
    select indeks, id_predmeta, ocena
    from db.ispit
    where ocena > 5
    and godina = 2012
),
smerovi as (
    select id
    from db.smer
    where naziv='Informatika'
)
select d.ime, d.prezime, p.naziv, ip.ocena
from ip
    join db.dosije d
        on d.indeks = ip.indeks
    join db.predmet p
        on ip.id_predmeta = p.id
where d.id_smera in (
  select * from smerovi
)
```



Primer 14: Plan izvršavanja upita.

Сви испити положени у школској 2012/13. години на смеру Информатика

```
select d.ime, d.prezime, p.naziv,
    ip.ocena
from db.dosije d,
    db.smer s,
    db.ispit ip,
    db.predmet p
where
    d.id_smera = s.id
    and s.naziv='Informatika'
    and ip.indeks = d.indeks
    and ip.ocena > 5
    and ip.godina = 2012
    and ip.id_predmeta = p.id
```



Manuelna optimizacija upita

Manuelna optimizacija upita je ranije bila mnogo potrebnija nego danas. Na različite načine se SUBP-u može sugerisati koji plan izvršavanja da primeni.

Danas je uloga manuelne optimizacije upita važnija u fazi pravljenja fizičkog modela nego u eksploataciji. Ne optimizuje se upit, nego se analiziraju planovi izvršavanja radi sagledavanja i prevazilaženja eventualnih slabosti predloženog modela.

Faktori odlučivanja

Pri pronalaženju najboljeg plana izvršavanja uzimaju se u obzir raspoloživi podaci o bazi podataka iz kataloga:

- struktura tabela i ključeva,
- struktura upita,
- postojeći indeksi,
- statistički podaci o sadržaju tabela i atributa,
- podaci o brzini fizičkih uređaja,
- veličina bafera stranica.

Statistike o bazi podataka

Da bi procena troškova izvršavanja bila što tačnija, moraju da postoje iscprne i ažurne statistike o bazi podataka. Statistike moraju da se prave na realnom sadržaju baze podataka. Detaljnost i preciznost statistika može da se bira. Ako su detaljnije i preciznije onda su:

- korisnije i omogućavaju efikasnije upite,
- obimnije i skuplje za čuvanje i korišćenje,
- složenije i skuplje za izračunavanje.

Statistike o tabelama mogu da se računaju

- za sve redove ili samo izabrane redove. Redovi se mogu birati na različite načine, uz određivanje stepena zastupljenosti.
- za svaku od kolona ili samo za izabrane kolone. Kolone primarnog ključa, stranih ključeva, manuelno izabrane,...
- samo opseg vrednosti ili detaljna raspodela (distribucija vrednosti (frekvencija, kvantili i sl .))
- automatski ili manuelno

Slične statistike se prave i za indekse.

8.5 Optimizacije na nivou strukture podataka

Ako imamo uzorak upita i njihove planove izvršavanja, onda možemo da sprovedemo optimizaciju strukture baze podataka u cilju povećavanja performansi uzorka upita. Ovde govorimo o optimizaciji fizičkog dizajna menjanjem fizičke strukture baze podataka.

Uzorak upita

Uzorak upita (engl . workload) je skup tipičnih upita (i promena baze podataka) za koje se zna da će činiti najčešći oblik pristupanja bazi podataka. Dobar uzorak se sastoji od:

- skupa upita i promena baze podataka,
- procenjene učestalosti izvršavanja i posebno vršno opterećenje za svaki upit i promenu,
- formulisanih ciljnih performansi.

Na osnovu uzorka...

Za upite:

- prepoznaje se koje se relacije (tabele, indeksi) koriste
- koji atributi se izdvajaju
- koji atributi učestvuju u uslovima spajanja i restrikcije

Za promene baze, pored toga se prepoznaje i:

- vrsta promene,
- atributi koji se menjaju (za ažuriranje).

Cilj strukturne optimizacije

Cilj strukturne optimizacije je prepoznavanje indeksa koje moramo da napravimo kao i prepoznavanje potrebnih promena u fizičkoj shemi:

- alternativna normalizacija,
- denormalizacija,
- uvođenje pogleda da bi se sakrile načinjene promene.

Alternativna normalizacija

Pri normalizovanju ne postoji samo jedno moguće rešenje:

- Normalizacija do 3. NF ili BKNF?
- Normalizacija do 3. NF različitim putevima.

Ako jedno rešenje ne daje zadovoljavajuće rezultate, možda drugo daje?

Primer različitih normalizacija

Neka relacija P opisuje projekte:

- $P(BrProjekta, Grad, Naziv)$

Imamo sledeće funkcionalne zavisnosti:

- fd1: $BrProjekta \rightarrow Grad$
- fd2: $BrProjekta \rightarrow Naziv$

Ako u jednom gradu postoji najviše jedan projekat i svi projekti imaju različite nazive, onda možemo da smatramo da postoji i:

- fd3: $Grad \rightarrow Naziv$
- fd4: $Naziv \rightarrow Grad$
- fd5: $Naziv \rightarrow BrProjekta$

Odatle sledi da imamo i alternativne normalizacije:

- Normalizacija 1:
  - $P(BrProjekta, Grad)$
  - $PN(Grad, Naziv)$
- Normalizacija 2:
  - $P(BrProjekta, Naziv)$
  - $PG(Naziv, Grad)$
- Pa čak i Normalizacija 3:
  - $P(BrProjekta, Grad)$
  - $PN(BrProjekta, Naziv)$

Denormalizacija

Neke tehnike denormalizacije:

- podela tabele (dekompozicija)
- spajanje tabela (kompozicija)
- dupliranje podataka

Upotreba slabije NF

Jača NF može da doslednije poštuje funkcionalne zavisnosti, ali da uvede višu cenu nekih operacija Međutim, ako upotrebimo slabiju NF i dodatne uslove integriteta, to može da bude još skuplje:

- $P(BrProjekta, Grad, Naziv)$
- ***check(not exists (select*** *naziv* ***from*** P ***group by*** *naziv* ***having count(distinct*** *grad)>1)))*

Svako slabljenje normalizacije zahteva da se funkcionalne zavisnosti održavaju na neki drugi način (uslovi integriteta ...). Savet je da se obavezno proveri da li se dobit (obično pri računanju upita) usled denormalizacije gubi pri menjanju sadržaja baze podataka.

Podela tabele (dekompozicija)

Ako imamo tabelu sa mnogo redova ili kolona, koja se često menja, onda to može da bude neefikasno jer:

- indeksi postaju „duboki“ i često se čita sa diska,
- upiti koji ne koriste indekse su tim pre neefikasni.

Jedan način da se rad ubrza je da se tabela podeli na dve (ili više). Imamo dve vrste podele: horizontalnu i vertikalnu.

Horizontalna podela tabele

Redovi tabele se podele u dve tabele sa istom strukturom. Pri tom obično jedna sadrži takozvane nove ili aktivne redove, a druga takozvane stare ili arhivske podatke.
Dobre strane:

- ubrzavaju se neke operacije (pre svega dodavanje i menjanje novih podataka).

Loše strane:

- dodatno se komplikuju neke operacije, pre svega analitički upiti nad svim podacima.

Particionisanje tabela

Jedan vid horizontalne podele je i particionisanje tabela. Često može da počiva na nekim kriterijumima raspona vrednosti. Na primer, podaci za svaku školsku godinu mogu da idu u posebnu particiju.

Što je više delova to su pojedinačne operacije sa podacima efikasnije, ali je pretraživanje cele tabele manje efikasno.

Vertikalna podela tabele

Ako se većina kolona ne koristi u svim upitima, već samo relativno retko, onda se česte operacije mogu ubrzati vertikalnom podelom

Prave se dve ili više tabela:

- svaka sadrži kolone primarnog ključa
- sve ostale kolone se grupišu prema upotrebi

Dobre strane:

- ubrzavaju se neke česte operacije

Loše strane:

- dodatno se komplikuju neke operacije, pre svega analitički upiti nad svim podacima

Spajanje tabela

Tabele koje se često (ili čak svaki put) spajaju u upitima, može da bude korisno spojiti u jednu tabelu.
Dobre strane:

- Spajanje je veoma skupa operacija i ovakva promena može da ima značajne efekte

Loše strane:

- Dobijena tabela obično narušava normalne forme i sadrži neke redundantnosti.
- To dodatno otežava održavanje podataka i staranje o integritetu.
- Može da se značajno poveća broj redova ili ukupno zauzeće prostora.

Dupliranje podataka

Jedna od tehnika optimizacije može da bude da se neki podaci svesno ponove u više tabela.
Dobre strane:

- Izbegavaju se suvišna spajanja ili unije

Loše strane:

- Redundantnost, odnosno otežano održavanje i povećano zauzeće prostora.

Druge vrste denormalizacije

Već pomenute promene spajanja tabela i ponavljanja podataka predstavljaju vidove denormalizacije.

Drugi oblici denormalizacije su različiti vidovi premeštanja ili kopiranja kolona iz jedne tabele u drugu, a koji za cilj imaju podizanje performansi (primarno kroz smanjivanje broja operacija spajanja). Svi vidovi denormalizacije imaju za posledicu uvođenje redundantnosti i sve što iz toga sledi.

Primeri:

- dva entiteta u odnosu 1-1 mogu da se modeliraju jednom tabelom
- dva entiteta u odnosu 1-N mogu da se modeliraju jednom tabelom

Dogma o denormalizaciji

Uvrežen je stav da je normalizovana baza podataka neefikasna. Međutim, to važi samo za specifične okolnosti i nije opšte pravilo. Na primer, baza podataka (ili njen deo) može da bude neefikasna ako je:

- normalizovana
- veoma velika
- mnogo su češći upiti nego transakcije
- upiti zahtevaju velika spajanja nad velikim skupom podataka

Denormalizacija vrlo često doprinosi efikasnosti ili sasvim malo ili nimalo. To je samo jedan od mogućih koraka.

Razrešavanje hijerarhija

Hijerarhije entiteta smo pominjali u modelu entiteta i odnosa. Hijerarhije postoje i u objektnom modelu i dijagramu klasa podataka. Pri pravljenju logičkog modela hijerarhija može da se prevede u relacije (tabele) na bar tri osnovna načina:

- jedna integralna tabelama,
- za svaki tip zasebna tabela, koja ima samo kolone ključa i kolone novih atributa,
- za svaki neapstraktan tip zasebna tabela, koja ima sve potrebne kolone,
- različite kombinacije.

Čest je problem sa hijerarhijama

- pri pravljenju logičkog modela se razmatraju kriterijumi „logičkih veza“ među podacima.
- pri pravljenju fizičkog modela zbog efikasnosti može da bude bolje da se primeni drugo rešenje.

Stabilno rešenje za hijerarhije

- Da se sve operacije čitanja odvijaju kroz poglede koji zaokružuju sve potrebne podatke o elementima hijerarhije
- Da se „zabrane“ direktne operacije menjanja podataka u tabelama koje pripadaju hijerarhiji
- Da se sva pisanja izvode kroz okidače na pogledima ili predefinisane procedure

Uvođenje pogleda da bi se sakrile načinjene promene

Ako se promeni fizička struktura baze podataka, onda se na konceptualnom nivou uvode novi pogledi koji skrivaju načinjene izmene. Ovo nije deo fizičkog modela, ali je posledica promena u fizičkom modelu.

8.6 Primer projektovanja baze podataka

Da rezimiramo, poslovi u okviru projektovanja baza podataka se mogu podeliti u 6 koraka:

1. Analiza zahteva: Šta korisnik želi od baze podataka
   - koje podatke treba čuvati u bazi podataka,
   - koje aplikacije treba izgraditi nad njima i
   - koje su najčešće operacije i koji su zahtevi za njihove performanase.
   - Ovo je obično neformalni proces i uključuje diskusiju s korisničkim grupama, anlizu radnog okruženja, očekivanih promena, dokumentacije o postojećim aplikacijama koje bi trebalo podržati bazom podataka i drugo.
2. Konceptualno projektovanje baze podataka
   - Prikupljena informacija se koristi za opis, visokog nivoa apstrakcije, podataka koji će se čuvati i ograničenja za te podatke.
   - Obično se koristi ER model podataka
3. Logičko projektovanje baze podataka
   - Izbor SUBP za implementaciju projekta baze podataka
   - Konverzija konceptualne sheme u shemu baze podataka odgovarajućeg modela (izabranog SUBP)
   - Kada je izabrani SUBP – RSUBP – konverzija ER sheme u shemu relacione baze podataka
   - Rezultat: konceptualna shema (zove se i logička shema) u RM
4. Profinjenje sheme
   - Analiza skupa relacija u relacionoj shemi – otkrivanje potencijalnih problema

- Za razliku od prethodnih koraka, za profinjenje sheme postoji moćna teorija
- Teorija normalizacije

5. Fizičko projektovanje baze podataka
   - Analiza tipičnog radnog opterećenja (workload)
   - Dalje profinjenje projekta baze podataka koje obezbeđuje kriterijume performansi
   - Izgradnja nekih indeksa, nekih sortova (klastera) ALI I
   - Reprojektovanje nekih delova sheme bp dobijene u prethodnim koracima
6. Projektovanje bezbednosti
   - Identifikuju se različite korisničke grupe i uloge koje korisnici imaju
   - Na primer, menadžer proizvodnje, agent podrške potrošača i sl.
   - Identifikovati delove baze podataka kojima svaki korisnik/uloga može odnosno ne sme da pristupi
   - Preduzeti mere za obezbeđivanje predviđenog pristupa
   - SUBP ima nekoliko mehanizama podrške ovog koraka

Navedenih 6 korka predstavlja samo klasifikaciju poslova. Potrebno je još i:

- (obično) neko "štelovanje" (tuning): svih 6 koraka se ponavlja, preklapa, do zadovoljavajućeg dizajna
- Implementacija dizajna
- Dizajn i implementacija aplikacionih nivoa nad SUBP.

8.6.1 Studija slučaja: Internet prodavnica

- DBDudes Inc., a well-known database consulting firm, has been called in to help Barns and Nobble (B&N) with their database design and implementation.
- B&N is a large bookstore specializing in books on horse racing, and they've decided to go online.
- DBDudes first verify that B&N is willing and able to pay their steep fees and then schedule a lunch meeting - billed to B&N, naturally - to do requirements analysis [13].

Analiza zahteva

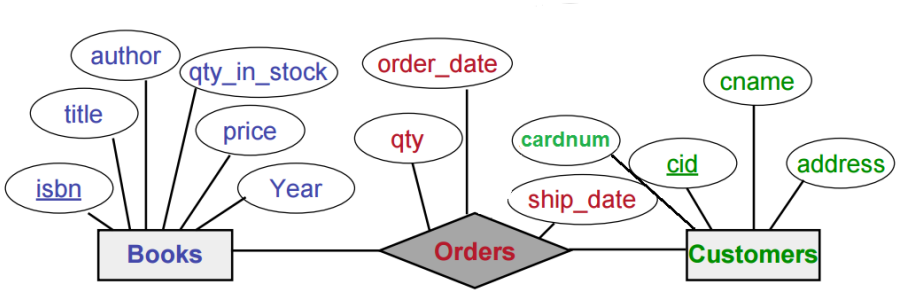
- Vlasnik B&N iznosi zahtev:
  ,,I would like my customers to be able to browse my catalog of books and to place orders over the Internet. Currently, I take orders over the phone. I have mostly corporate customers who call me and give me the ISBN number of a book and a quantity. I then prepare a shipment that contains

the books they have ordered. If I don't have enough copies in stock, I order additional copies and delay the shipment until the new copies arrive; I want to ship a customer's entire order together. My catalog includes all the books that I sell. For each book, the catalog contains its ISBN number, title, author, purchase price, sales price, and the year the book was published. Most of my customers are regulars, and I have records with their name, address, and credit card number. New customers have to call me first and establish an account before they can use my Web site.

On my new Web site, customers should first identify themselves by their unique customer identification number. Then they should be able to browse my catalog and to place orders online.“

Konceptualno projektovanje

- Inicijalni opis DBDudes visokog nivoa – ER dijagram:



Interpretacija: knjige (Books) i korisnici (Customers) modeliraju se kao entiteti i povezuju se narudžbenicama (Orders); to je odnos koji ima atribute, pri čemu atribut shp_date ima vrednost NULL pre slanja narudžbenice

- Prva interna recenzija (recenzent „Dude 2“):
  - Dude 2: What if a customer places two orders for the same book on the same day?
  - Dude 1: The first order is handled by creating a new Orders relationship and the second order is handled by updating the value of the quantity attribute in this relationship.
  - Dude 2: What if a customer places two orders for different books on the same day?
  - Dude 1: No problem. Each instance of the Orders relationship set relates the customer to a different book.
  - Dude 2: Ah, but what if a customer places two orders for the same book on different days?
  - Dude 1: We can use the attribute order date of the orders relationship to distinguish the two orders.
  - Dude 2: Oh no you can't. The attributes of Customers and Books must jointly contain a key for Orders. So this design does not allow a customer to place orders for the same book on different days.

– Dude 1: Yikes, you're right. Oh well, B&N probably won't care; we'll see.

Logičko projektovanje

- DBDudes preslikavaju ER dijagram u RM:

```
CREATE TABLE Books ( isbn CHAR(10),
                     title CHAR(80),
                     author CHAR(80),
                     qty_in_stock INTEGER,
                     price REAL,
                     year published INTEGER,
                     PRIMARY KEY (isbn))
CREATE TABLE Orders ( isbn CHAR(10),
                      cid INTEGER,
                      qty INTEGER,
                      order date DATE,
                      ship date DATE,
                      PRIMARY KEY (isbn,cid),
                      FOREIGN KEY (isbn) REFERENCES Books,
                      FOREIGN KEY (cid) REFERENCES Customers )
CREATE TABLE Customers ( cid INTEGER,
                      cname CHAR(80),
                      address CHAR(200),
                      cardnum CHAR(16),
                      PRIMARY KEY (cid)
                      UNIQUE (cardnum))
```

- U pokušaju da prevaziđe neplanirano ograničenje, Dude 1 predlaže proširenje ključa odnosa Order koje nije u skladu sa ER dijagramom:

```
CREATE TABLE Orders ( isbn CHAR(10),
                      ...
                      PRIMARY KEY (isbn,cid,order_date),
                      ... )
```

- Prezentacija B&N:
  - Novi zahtevi: „Customers should be able to purchase several different books in a single order. In addition, they could place more than one order per day, and they want to be able to identify the orders they placed."
  - Diskusija o načinu slanja knjiga:
    - B&N: „As soon as we have enough copies of an ordered book we ship it, even if an order contains several books."
  - DBDudes: novi atribut ordernum u relaciju Orders

```
CREATE TABLE Orders ( ordernum INTEGER,
                      isbn CHAR(10),
                      cid INTEGER,
                      qty INTEGER,
                      order_date DATE,
                      ship_date DATE,
                      PRIMARY KEY (ordernum, isbn),
                      FOREIGN KEY (isbn) REFERENCES Books
                      FOREIGN KEY (cid) REFERENCES Customers )
```

Profinjenje sheme

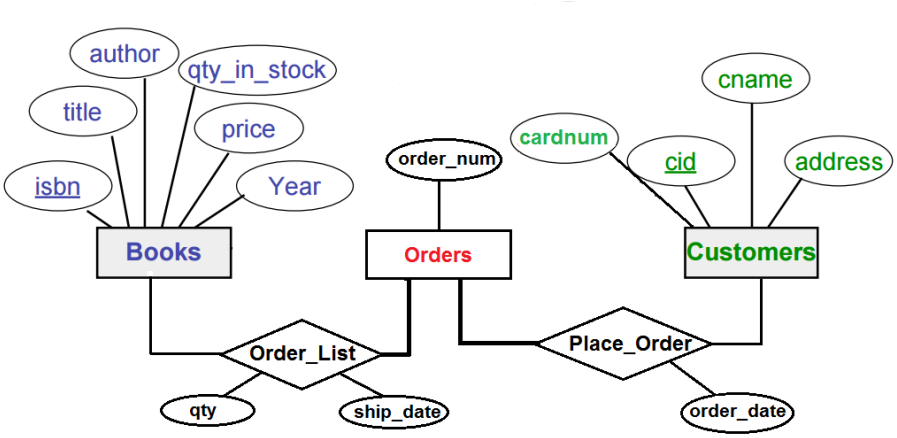
- DBDudes analiziraju moguće redundanse
  - Relacija Books u BCNF: samo jedan ključ (*isbn*) i nema drugih FZ Relacija Customers ima ključ *cid*, i FZ *cardnum* $\rightarrow$ *cid*, pa je i *cardnum* ključ. Nema drugih FZ pa je i relacija *Customers* u BCNF.
  - Relacija *Orders* ima ključ (*ordernum*, *isbn*) i važe FZ *ordernum* $\rightarrow$ *cid*, i *ordernum* $\rightarrow$ *order_date*, pa ona nije ni u 3NF (zašto? Da li je u 2NF?)
  - Relaciju *Orders* dekomponuju u dve relacije, obe u BCNF:
    Orders(ordernum, cid, order_date) i
    Orderlists(ordernum, isbn, qty, ship_date)
  - Dekompozicija je lossless join i čuva polazne FZ

```
CREATE TABLE Orders ( ordernum INTEGER,
                      cid INTEGER,
                      order_date DATE,
                      PRIMARY KEY (ordernum),
                      FOREIGN KEY (cid) REFERENCES Customers )
CREATE TABLE Orderlists ( ordernum INTEGER,
                      isbn CHAR(10),
                      qty INTEGER,
                      ship_date DATE,
                      PRIMARY KEY (ordernum, isbn),
                      FOREIGN KEY (isbn) REFERENCES Books)
```

  - Figure shows an updated ER diagram that reflects the new design.
  - Note that DBDudes could have arrived immediately at this diagram if they had made Orders an entity set instead of a relationship set right at the beginning. But at that time they did not understand the requirements completely, and it seemed natural to model Orders as a relationship set.
  - This iterative refinement process is typical of real-life database design processes. As DBDudes has learned over time, it is rare to achieve an initial design that is not changed as a project progresses.

Novi ER dijagram

- Orders je postao entitet! Napomena: Primetimo da je relacija Place_Order kardinalnosti više-jedan tako da neće biti preslikana u novu relaciju.



Fizičko projektovanje

- DBDudes razmatraju očekivano radno opterećenje
- Vlasnik očekuje najintenzivnije pretraživanje po isbn pre naručivanja
- Naručivanje uključuje unošenje jednog sloga u tabelu Orders i jedan ili više slogova u tabelu Orderlists
- Ako ima dovoljno primeraka, priprema se pošiljka i postavlja vrednost za *ship_date*. Smanjuje se raspoloživa količina
- Nove knjige stižu od dobavljača i povećavaju raspoložive količine
- **Odluke:**
  - **Books:**
  - Pretraživanje po isbn je pretraživanje na jednakost, heširanje
  - Ažuriranje količine (*qty_in_stock*) – prvo pretraga po *isbn*; efikasna
    - Ažuriranje *qty_in_stock* često – vertikalno particionisanje:
      BooksQty(isbn, qty), i
      BookRest(isbn, title, author, price, year_published)
    - Usporava pretragu svih informacija o knjizi (spajanje) – pa se odustaje
  - DBDudes očekuju česte pretrage po naslovu i autoru, pa dodaju indekse po atributima title i author – jeftini jer se naslovi i autori retko menjaju
  - **Customers:**
  - Najčešći upiti su pretraga na jednakost sa *cid* – heš indeks po *cid*
  - **Orders:**
  - Dva upita: unošenje novih i pretraga postojećih

  - *ordernum* atribut uključen: grupišući B+ indeks (dodeljuju se redom i odgovaraju datumu, pa je sortiranje po *ordernum* istovremeno i sortiranje po *order_date*)
  - **Orderlist:**
  - Uglavnom uključuje unošenja slogova i povremeno izmenu datuma pošiljke, ili
  - Listanje komponenti narudžbe
  - Grupišući B+ indeks po ordernum
  - Efikasna pretraga svih stavki jedne narudžbe
  - Za ažuriranje datuma, pretraga po paru (ordernum, isbn) – pomaže indeks po ordernum
  - Indeks po paru (ordernum, isbn) bi usporio unošenje (insert) slogova u odnosu na to kada imamo samo indeks po ordernum
  - Ostaje samo indeks po ordernum

„Štelovanje“ (tuning)

- Posle nekoliko meseci od uvođenja baze podataka, stiže žalba da se upiti nad narudžbama vrlo sporo obrađuju
- B&N su postali uspešni, tabele Orders i Orderlists su postale ogromne
- DBDudes uočavaju dva tipa narudžbi:
  - Kompletirane – sve knjige poslate (većina)
  - Delimično kompletirane (mali deo)
  - Većina kupaca nadgleda delimično kompletirane narudžbe
  - Horizontalno particionisanje tabela Orders i Orderlists po ordernum.
  - Četiri nove relacije: NewOrders, OldOrders, NewOrderlists, i OldOrderlists
  - Usporeni su samo (retki) upiti koji zahtevaju i stare i nove narudžbe, npr. sve narudžbe jednog kupca

Bezbednost

- Tri grupe korisnika: kupci, zaposleni i vlasnik (naravno, i db administrator sa svim pravima)
- Vlasnik ima sve privilegije nad svim tabelama
- Kupci mogu da pretražuju tabelu Books i da izdaju narudžbe ali nemaju pristup slogovima drugih kupaca i njihovih narudžbi
- DBDudes: Ograničenje pristupa na dva načina
  - Veb stranica sa nekoliko formi – mali broj različitih zahteva (bez mogućnosti SQL interfejsa)
    * Pretraga Books po isbn, author, title
    * Dva dugmeta: lista delimično kompletiranih i lista kompletiranih narudžbi

        * Poslovna logika: održavanje cid (pri logovanju na veb sajt)
    - Svojstva bezbednosti SUBP-a

- Ograničavanje pristupa prem korisničkoj grupi
- Nalog customer sa privilegijama:
  SELECT ON Books, NewOrders, OldOrders, NewOrderlists, OldOrderlists
  INSERT ON NewOrders,, NewOrderlists,
- Zaposleni mogu da dodaju knjigu katalogu, ažuriraju količinu knjiga u magacinu, izmene narudžbu kupca, ažuriraju podatke o kupcu OSIM informacije o kreditnoj kartici (ne treba ni da je vide)
- Pogled CustomerInfo
  CREATE VIEW CustomerInfo (cid,cname,address)
  AS SELECT C.cid, C.cname, C.address
  FROM Customers C
- Nalog employee ima privilegije:
  SELECT ON CustomerInfo, Books, NewOrders, OldOrders,
  NewOrderlists, OldOrderlists
  INSERT ON CustomerInfo, Books, NewOrders, OldOrders,
  NewOrderlists, OldOrderlists
  UPDATE ON CustomerInfo, Books, NewOrders, OldOrders,
  NewOrderlists, OldOrderlists
  DELETE ON Books, NewOrders, OldOrders, NewOrderlists, OldOrderlists
- Logovanje na veb sajt (cid) zahteva bezbedni protokol, npr. SSH
- Neke kompanije nude elektronsko plaćanje (druga tema)

Aplikativni nivoi

- Potreba za upravljanjem sesijom (čuvanje cid-a)
- Statičke ili dinamičke veb stranice za knjige; statička zahteva izradu posebne stranice za svaku knjigu; dinamička se generiše iz standardnog obrasca (templejta)
- Povezivanje aplikacije sa SUBP
    - CGI skript
        * Kodiranje logike upravljanja sesijom; nije lako
    - Infrastruktura aplikacionog servera
        * Korišćenje funkcionlnosti aplikacionog servera
    - Preporučuju da B&N implementira obradu na strani servera korišćenjem aplikacionog servera
    - B&N odbija da plati za aplikacioni server i želi CGI skriptove
    - DBDudes razvija:
        * Vršne HTML stranice za navigaciju i razne forme za pretragu kataloga i prezentaciju rezultata

    - Logika za praćenje sesije kupca – relevantna informacija ili na serveru ili u razgledaću (npr. kolačići)
    - Skriptovi za obradu zahteva
- U slučaju izbora aplikacionog servera, drugi skup zadataka

9

Distribuirane baze podataka

Pojmovi

- CBP – Centralizovana baza podataka je baza podataka koja u celosti počiva na jednom računaru.
- CSUBP – Centralizovani sistem za upravljanje bazama podataka je softverski sistem koji omogućava upravljanje CBP.
- DBP – Distribuirana baza podataka je skup više logički međuzavisnih baza podataka koje su distribuirane na računarskoj mreži.
- DSUBP – Distribuirani sistem za upravljanje bazama podataka (DDBMS – distributed DBMS) je softverski sistem koji omogućava upravljanje DBP tako da je distribuiranost transparentna za korisnika.

Zašto distribuiramo?

Distribuiran pristup prirodnije odgovara organizacionoj strukturi današnjih distribuiranih preduzeća i doprinosi većoj pouzdanosti sistema. Još važnije, mnoge današnje aplikacije su prirodno distribuirane, kao na primer, razne vrste veb aplikacija, elektronska trgovina preko Interneta, razne vrste multimedijalnih aplikacija i drugo. Osnovni razlog distribuiranja je ipak mogućnost boljeg nošenja sa problemima upravljanja podacima sa kojima se suočavamo danas, korišćenjem varijanti dobro poznatog ,,zavadi pa vladaj“ pravila.

Distribuiranost i baze

U kontekstu razmatranja distribuiranih baza podataka koristi se isključivo stroga definicija distribuiranosti. Suštinski je važno da su distribuirane komponente na različitim računarima s tim da udaljenost nije značajna. Dakle, mogu da budu jedan do drugog ili na različitim kontinentima. Dodatno, pretpostavlja se da je mreža jedini deljeni resurs.

DSUBP nije obična ,,kolakcija datoteka“ koje su pojedinačno smeštene na različitim računarima. Da bi govorili o DSUBP datoteke ne treba da budu samo logički povezane, već treba da budu struktuirane na određen način i pristup podacima treba da bude obezbeđen preko zajedničkog interfejsa.

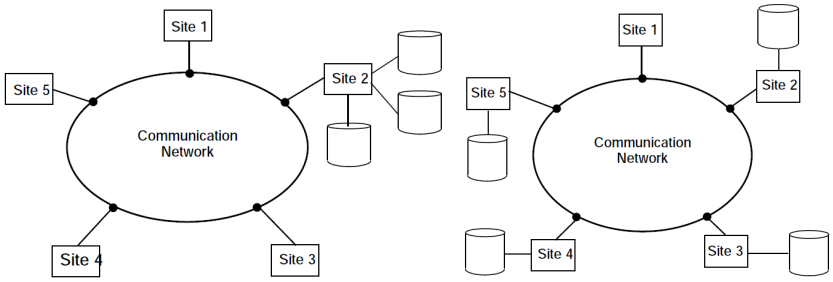
Višeprocesorski sistemi

Ako višeprocesorski sistemi dele memoriju nazivaju se *sistemi sa deljenom memorijom* ako se deli primarna memorija, odnosno *sistemi sa deljenim diskom* ako se deli sekundarna memorija. Ako dele i primarnu i sekundarnu memoriju i U/I uređaje, nazivaju se *sistemi koji dele sve*. U ovakve sisteme se svrstavaju i oni koji dele memoriju. Ako ne dele ni primarnu ni sekundarnu memorija ni U/I uređaje, nazivaju se *sistemi koji ne dele ništa*.

Višeprocesorski sistemi ne predstavljaju osnovu za distribuirane baze podataka. Ako dele memoriju (ili čak sve) onda je to jedan računar u kome se komunikacija ostvaruje kroz memoriju, a ne putem poruka kroz mrežu, pa nema distribuiranosti. Ako sistemi ne dele ništa, svaki prcesor ima svoju primarnu i sekundarnu memoriju, pa čak i svoje periferne uređaje, što predstavlja slično okruženje kakvo postoji u distribuiranim sistemima. Ipak, postoji razlika. Kod višeprocesorskih sistema se obično radi o simetričnim sistemima sa istim (ili skoro istim) procesorskim i memorijskim podsistemima, koji su kontrolisani jednim istim operativnim sistemom i čine jedan računar. U DSUBP heterogenost operativnog sistema kao i hardvera je podrazumevana. Baze podataka koje se izvršavaju na višeprocesorskim sistemima nazivaju se paralelnim sistemima baza podataka.

Baze podataka na mreži

Činjenica da se SUBP nalazi u okviru mreže nije dovoljna da bi se radilo o DSUBP. Ako je na računarskoj mreži sa većim brojem računara baza podataka locirana na jednom čvoru, tada to nije DSUBP (leva strana slike 9.1). SUBP je distribuiran akko je lociran na više različitih čvorova u mreži (desna strana slike 9.1).



Slika 9.1: Primeri CSUBP i DSUBP [9].

9.1 Doprinosi DSUBP

Osnovni doprinosi DSUBP su:

- Transparentno upravljanje distribuiranim i repliciranim podacima;
- Pouzdanost distribuiranih transakcija;
- Unapređenje performansi;

- Lakše proširivanje sistema.

Transparentno upravljanje distribuiranim i repliciranim podacima

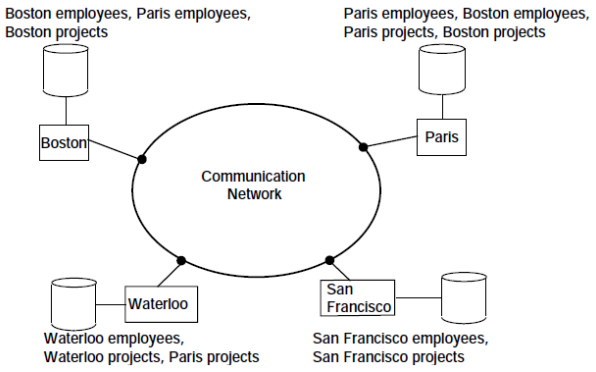
Transparentnost podrazumeva razdvajanje semantike visokog nivoa od problema koji nastaju pri implementaciji na niskom nivou.

Primer

Inženjerska firma ima kancelarije u Bostonu, Vaterlo (Waterloo), Parizu i San Francisku i u svakom od gradova zaposleni rade na nekim projektima. U bazi podataka potrebno je čuvati podatke o zaposlenima, o projektima kao i ostale prateće podatke. Ako pretpostavimo da je baza relaciona, ove informacije se mogu čuvati u dve tabele: *zaposleni* i *projekti*. Možemo imati i treću tabelu *zarade*, i četvrtu *rade_na_projektu* u kojoj se smeštaju podaci o tome koji zaposleni rade na kom projektu, sa kojom odgovornošću. Ako se ovi podaci čuvaju u CBP, i ukoliko želimo da pronađemo imena i zarade svih zaposlenih koji rade na projektu duže od 12 meseci, možemo napisati sledeći upit:

```
SELECT ime, plata
FROM zaposleni, rade_na_projektu, zarada
WHERE rade_na_projektu.trajanje > 12
   AND zaposleni.broj_projekta=rade_na_projektu.broj_projekta
   AND zarada.odgovornost=zaposleni.odgovornost
```

Imajući u vidu distribuiranu prirodu ove kompanije, poželjno je u datim okolnostima lokalizovati podatke tako da podaci o zaposlenima u Bostonu budu smešteni u Bostonu, podaci o zaposlenima u Parizu, budu smešteni u Parizu, i tako dalje. Isto važi za projekte i zarade. Dakle, potrebno je da relacije podelimo na particije koje će biti smeštene na različitim računarima. Ovo je poznato kao *fragmentacija*. Više od toga, možda će biti poželjno da dupliramo neke podatke sa jednog na drugi računar, kako bi povećali performanse i pouzdanost. Rezultat toga je distribuirana baza podataka koja je fragmentisana i replicirana (slika 9.2).



Slika 9.2: Fragmentisana i replicirana DBP [9].

Transparentno upravljanje distribuiranim i repliciranim podacima podrazumeva da nezavisno od toga gde se fizički čuvaju podaci, da li su distribuirani ili ne i slično, podacima se mora pristupati na isti univerzalan način. Dakle, korisnik može i dalje da izvrši prethodno definisan upit, ne obraćajući pažnju na to gde su podaci smešteni, da li su fragmentisani ili ne, da li su replicirani i slično. DSUBP će brinuti o tome.

Transparentnost ima sledeće aspekte:

- Nezavisnost podataka;
- Mrežna transparentnost;
- Transparentnost replikacije;
- Transparentnost fragmentacije.

Nezavisnost podataka

Nezavisnost podataka je najvažniji i jedini aspekt transparentnosti koji se zahteva i u CBP i u DBP. Možemo govoriti o dva tipa nezavisnosti podataka:

- *Logička nezavisnost podataka*, koja podrazumeva otpornost aplikacije na promene logičke strukture podataka. To znači da aplikacije dobro podnose dodavanje novih elemenata strukturi baze podataka.
- *Fizička nezavisnost podataka*, koja podrazumeva potpuno skrivanje fizičke strukture podataka od aplikacije. Aplikacija, dakle, ne sme da trpi nikakve posledice u slučaju promene fizičke strukture podataka, makar ona uključivala i promene u načinu distribuiranja podataka.

Mrežna transparentnost

U centralizovanim bazama podataka jedini resurs o kome se stara transparentno jesu podaci. Korisniku nije potrebno da zna kako se upravlja podacima.

U distribuiranim sistemima, mrežna struktura je potrebno da bude sklonjena od očiju korisnika i transparentno upravljana. Korisniku nije potrebno da zna gde su podaci. Ovaj aspekt transparentnosti se naziva još i distributivna transparentnost.

Transparentnost replikacije

Replikacija je čuvanje istih podataka na više lokacija. Preduzima se radi performansi, raspoloživosti i pouzdanosti. Korisnici ne bi trebalo da budu svesni činjenice da se podaci repliciraju. Bilo da su replicirani ili ne, oni se moraju koristiti na isti način.

Transparentnost fragmentacije

Fragmentacija je čuvanje različitih delova iste kolekcije podataka na više lokacija. Preduzima se radi performansi, raspoloživosti i pouzdanosti. Korisnici ne bi trebalo da budu svesni činjenice da su podaci fragmentisani.Bilo da su fragmentisani ili ne, oni se moraju koristiti na isti način. Obično se upiti (tzv. globalni

upiti) prevode u izvršavanje kolekcije manjih (tzv. lokalnih upita) i rezultati se uniraju.

Pouzdanost distribuiranih transakcija

Nosilac pouzdanosti u radu SUBP-a je transakcija. Izvođenje transakcija u distribuiranom okruženju donosi nove probleme, kao na primer, prevazilaženje otkazivanja udaljenog uređaja u fazi potvrđivanja transakcije.

Unapređenje performansi

Doprinos DBP performansama obično se ogleda u tome što:

- Ravnomerno je raspodeljeno opterećenje na više servera na različitim lokacijama.
- Fragmentisanjem podaci se čuvaju bliže mestu upotrebe.
- Manje se vremena troši na prenos podataka. Iako može delovati da brze mreže umanjuju značaj toga, činjenica je da se podaci distribuirano proizvode i koriste, pa će distribuirano čuvanje uvek biti prirodna organizacija. Posebno, brza mreža može da smanji trajanje prenosa, ali ne jdnako dobro i početno kašnjenje odgovora.
- Globalni upiti se paralelizuju. Podaci su distribuirani, pa se globalni upiti dele na manje, koji se odnose na lokalne podatke.

Lakše proširivanje sistema

Distribuiranom SUBP je mnogo lakše povećati kapacitet od centralizovanog. Skoro je uvek jevtinije istu snagu obezbediti pomoću više manjih računara nego pomoću jednog velikog. Naravno da postoji prostor za velike računare, ali je u velikom broju slučajeva iz ekonomskog ugla opravdanije ulaganje u distribuirani sistem jer je manje inicijalno ulaganje i dolazi do postepenog proširivanja u skladu sa realnim potrebama.

9.2 Otežavajući faktori

Najvažniji otežavajući faktori su:

- *Složenost.* Distribuirani sistemi su sve samo ne jednostavni. Distribuirani SUBP mora da reši sve probleme koje rešava i centralizovan SUBP i još mnoge druge.
- *Cena.* Distribuirani sistemi zahtevaju dodatni hardver (na primer, komunikacioni). Softverski aspekti sistema su mnogo složeniji i skuplji za razvijanje. Na svakoj lokaciji gde postoje serveri potrebno je i održavanje.
- *Distribuirana kontrola.* Otežani su sinhronizacija i koordinacija komponenti.
- *Bezbednost.* Kod centralizovanih BP staranje o bezbednosti je centralizovano. Ovde je distribuirano i uključuje dodatne aspekte bezbednosti računarskih mreža.

9.3 Problemi i teme DSUBP

Najvažniji problemi koji se rešavaju u DSUBP su:

- Projektovanje distribuiranih baza podataka
- Distribuirano izvršavanje upita
- Distribuirano upravljanje metapodacima
- Distribuirana kontrola konkurentnosti
- Distribuirano upravljanje mrtvim petljama
- Pouzdanost distribuiranih SUBP
- Podrška u operativnim sistemima
- Heterogene baze podataka

Projektovanje distribuiranih BP

Ovde se javljaju novi problemi u kontekstu distribuiranja. Naime podaci mogu da budu:

- particionisani – (fragmentisani) podeljeni na više lokacija
- replicirani – ponovljeni na više lokacija
- a moguće su i kombinacije.

U slučaju replikacije može se primeniti

- puna replikacija
- delimična replikacija

U slučaju particionisanja određuje se način fragmentisanja podataka:

- horizontalna fragmentacija ili
- vertikalna fragmentacija.

Odlučivanje često počiva na rešavanju NP složenih problema, pa se pristupa heuristički.

Distribuirano izvršavanje upita

Izvršavanje upita obuhvata analizu upita i prevođenje upita u niz operacija. Optimizacija upita je složeno pitanje i u centralizovanim BP, a u slučaju DBP to postaje još izraženije. Dodatni faktori u odlučivanju su:

- cena mrežnog transporta,
- različitost cena izvršavanja na različitim lokacijama,
- različitost raspoloživih resursa na različitim lokacijama,
- mogućnosti paralelizacije.

Distribuirano upravljanje metapodacima

Metapodaci obuhvataju sve informacije koje se o strukturi, distribuiranosti i sadržaju baze podataka moraju čuvati da bi njen rad bio moguć. Za razliku od centralizovanih BP, sada:

- Postoje dodatne informacije o lokacijama čvorova.
- Postoje dodatne informacije o raspoređenosti podataka po čvorovima:
  - fragmentacija
  - replikacija
- Svaki čvor mora da zna mnogo toga o drugim čvorovima, ali ne uvek sve.
- Metapodaci mogu biti:
  - centralizovani
  - replicirani
  - particionisani

Distribuirana kontrola konkurentnosti

Kontrola konkurentnosti podrazumeva sinhronizaciju pristupanja podacima od strane konkurentno izvršavanih procesa.

Obuhvata:

- staranje o integritetu baze podataka (kao i za CBP)
- staranje o konzistentnosti replika
- složenije staranje o izolovanosti

Nekoliko pristupa:

- pesimistički pristup, koji podrazumeva sinhronizaciju pre ostvarenog pristupa (zaključavanje, vremenski pečati)
- optimistički pristup, koji podrazumeva sinhronizaciju posle ostvarenog pristupa

Kontrola konkurentnosti se zasniva na sledećim konceptima:

- zaključavanje (engl. locking-based) – počivaju na međusobnom isključivanju. Osnovna ideja je obezbediti da podatku kojeg dele konfliktne operacije, može da pristupi samo jedna operacija u jednom trenutku. Razlikuju se ključevi za čitanje (engl. read lock) i ključevi za pisanje (engl. write lock). Mogu da rezultuju mrtvim petljama (engl. deadlocks).
- vremenski pečati (timestamp) – počivaju na održavanju redosleda izvršavanja.
- hibridni mehanizmi.

Distribuirano upravljanje mrtvim petljama

Potencijalno viša složenost problema i konkurentnih odnosa nego kod CBP. Rešava se na sličan način kao i na nivou operativnih sistema:

- prevencijom
- izbegavanjem
- prepoznavanjem i oporavkom

Pouzdanost distribuiranih SUBP

Pouzdanost ne dolazi sama po sebi, odnosno za svaki od narednih doprinosa su potrebni neki preduslovi.

- Ako jedan čvor otkaže, ostali i dalje rade.
- Ako jedan čvor otkaže, drugi mogu da preuzmu njegovu ulogu.
- Kada se čvor oporavi, automatski se osvežava njegov sadržaj na osnovu drugih čvorova.

Podrška u operativnim sistemima

Operativni sistemi ne pružaju uvek odgovarajuće usluge složenim sistemima, kakav DSUBP izvesno jeste. Problemi i zagušenja obično nastaju po pitanju:

- upravljanja radom procesora i raspodelom vremena
- upravljanja procesima
- upravljanja memorijom
- rada sa datotekama
- oporavak

Često se neki elementi implementiraju ispod nivoa operativnog sistema

- na primer, neposredan pristup hardveru

Heterogene baze podataka

Do sada je bilo reči o homogenim DBP, bazama koje su logički integrisane i koje, iako su distribuirane na više računara, pružaju jedinstvenu sliku o bazi kao celini.

U određenim slučajevima DBP nastaje kao fuzija više postojećih rešenja. Tada je obično narušena:

- homogenost softvera – jer različite baze rade pod različitim SUBP,
- homogenost strukture – jer struktura različitih baza podataka nije usaglašena.

I tada je potrebno rešavati neke od opisanih problema

- distribuirano izvršavanje upita
- upravljanje metapodacima
- upravljanje konkurentnošću

Ovakvi sistemi se nazivaju *sistemi sa više baza podataka* (engl. multi-database management systems, multi-DBMS) i ponekad se ne svrstavaju u DBP.

9.4 Distribuirani sistemi i pouzdanost

U distribuiranim sistemima je realnost da neke od distribuiranih komponenti nisu funkcionalne. Pitanje nije ,,da li" nego ,,koliko će i kojih" komponenti biti neispravno u nekom trenutku.

Neispravnosti obuhvataju:

- bagove,
- ljudske greške pri upravljanju sistemima,
- preopterećenost (operativna),
- zagušenje (komunikaciono),
- oštećenja medija za čuvanje podataka,
- oštećenja elektronskih uređaja,
- zlonamerne aktivnosti.

Odnos prema neispravnostima

- Izolovanje neispravnosti
- Tolerancija neispravnosti

Izolovanje neispravnosti

Princip:

- Obezbeđivanje da neispravnost jedne komponente ne utiče na funkcionalnost ostalih komponenti.

Posledica:

- Funkcija koju je obezbeđivala neispravna komponenta nije raspoloživa dok se problemi ne otklone i komponenta ne postane operativna.

Tolerancija neispravnosti

Princip:

- U slučaju neispravnosti neke od komponenti, druge komponente preuzimaju njene funkcije.

Posledice:

- Povećana fleksibilnost i pouzdanost sistema
- Povećana složenost i cena sistema

Ostvarivanje tolerancije

Osnovni metodi:

- Rekonfigurisanje sistema
- Prikrivanje neispravnih komponenti

Zajedničke osobine:

- Neophodna je redundandnost
- Korisnici sistema ne opažaju neispravnosti osim eventualno kroz umanjene performanse sistema

Aspekti pouzdanosti

Pouzdanost sistema (dependability) čine:

- Raspoloživost sistema (availability)
  - ,,Sposobnost sistema da primi zahtev".
  - Sistem je visoko raspoloživ ako je relativno nisko trajanje perioda kada sistem ,,ne radi" – ili retko dolazi do neispravnosti ili se one brzo otklanjaju.
- Odgovornost sistema (reliability)
  - ,,Sposobnost sistema da odgovori na zahtev".
  - Sistem će biti visoko odgovoran ako uopšte neće imati neispravnosti tokom rada ili će one biti prevazilažene u hodu tako da ne bude prekida operativnosti – ili retko dolazi do neispravnosti ili je sistem u potpunosti sposoban da se dovoljno brzo oporavi da one ne ometaju rad.

Raspoloživost i odgovornost

Sistem može biti visoko odgovoran a da nije visoko raspoloživ

- Primer: sistem koji ne prima zahteve tokom pravljenja rezervnih kopija, a u međuvremenu radi sa retkim neuspesima

Sistem može biti visoko raspoloživ a da nije visoko odgovoran

- Primer: sistem koji je skoro uvek u stanju da primi zahtev (na primer, automatski se prevazilaze eventualni kvarovi) ali relativno često ne uspeva da odgovori na zahtev (na primer, zbog neispravnosti softvera)

Mera odgovornosti

Sistem je potpuno odgovoran ako nikada ne trpi oštećenja tokom obrade zahteva. Mera odgovornosti je verovatnoća da će sistem odgovoriti na zahteve koje je primio.

- $R(t) = 1 - F(t)$

- $R(t)$ je odgovornost u intervalu vremena dužine t
- $F(t)$ je verovatnoća da će doći do greške u intervalu dužine t

$R(t) = 1 - \int_0^t f(x)dx$

- $f(t)$ je gustina verovatnoće neuspeha

Mera raspoloživosti

Sistem je potpuno raspoloživ ako je uvek u stanju da primi zahtev. Mera raspoloživosti $A(t)$ je verovatnoća da će sistem primiti zahtev u nekom trenutku t, odnosno da će sistem

- ili raditi ispravno u intervalu $[0, t]$
- ili će poslednji problem biti otklonjen u nekom trenutku x, gde je $0 < x < t$

Ova mera se naziva i ,,trenutna raspoloživost".

Mera pouzdanosti

Neka je s servis čije izvršavanje traje τ

- Raspoloživost $A_s(t)$ je verovatnoća događaja $I_s(t)$ da će servis biti uspešno iniciran u trenutku t
- Odgovornost $R_s(t, \tau)$ je uslovna verovatnoća događaja $T_s(\tau)$ da će sistem uspešno da završi obradu zahteva, pod uslovom da je nastupio događaj $I_s(t)$
- Pouzdanost sistema je verovatnoća $D_s(t, \tau)$, koji predstavlja konjunkciju događaja $I_s(t)$ i $T_s(\tau)$.

$$D_s(t, \tau) = A_s(t)R_s(t, \tau)$$

9.5 Replikacija

Šta je replikacija?

Replikacija je svako udvajanje neke komponente računarskog sistema koje donosi neki nivo redundantnosti

Zašto repliciramo?

Dva osnovna motiva:

- Performanse
- Pouzdanost

U slučaju distribuiranih baza podataka replikacija je često neophodno sredstvo za dostizanje potrebnog nivoa i performansi i pouzdanosti.

Podizanje performansi

- Kroz raspodelu opterećenja na replike
- Posebno je efikasno u slučaju čitanja
- U slučaju pisanja čak može biti negativan uticaj

Podizanje pouzdanosti

Implementacija tolerancije neispravnosti

- kroz maskiranje neispravnosti
- i rekonfigurisanje sistema

Prepoznavanje neispravnosti

Šta repliciramo

Predmet replikacije mogu biti:

- podaci
- procesi
- objekti
- poruke

Nas prvenstveno zanima replikacija podataka.

Predmet replikacije podataka

- Baza podataka
- Tabela
- Fragment tabele
- Globalni katalog baze podataka
- Sistem datoteka
- Pojedinačna datoteka

Ograničenja replikacije

Osnovnu cenu replikacije predstavlja

- dodatna cena prostora zbog redundantnog čuvanja podataka
- povećana cena pisanja zbog menjanja podataka na više lokacija
- cena dodatne obrade usled implementacije replikacije

Načini replikacije podataka

Osnovni princip

- Podaci se održavaju na više (distribuiranih) lokacija

Načini replikacije

- Čitaj jedan piši sve (ROWA)
- Konsenzus kvoruma (KK) (glasanje)
- Konsenzus kvoruma u uređenim mrežama

Protokoli ROWA

„Čitaj jedan piši sve“ („Read One Write All“). Koncept ovog protokola je:

- Samo primarna kopija se čita
- Sve kopije se menjaju

Protokoli ROWA

- tolerišu otkaze lokacija kada je u pitanju čitanje
- ne tolerišu otkaze lokacija kada je u pitanju pisanje
- ne tolerišu otkaze komunikacija

Varijante

- Osnovni ROWA protokol
- ROWA-A, sa menjanjem raspoloživih kopija
- ROWA sa primarnom kopijom
- ROWA sa tokenima pravih kopija

Osnovni ROWA protokol

Protokol ROWA

- Prevodi svaku operaciju čitanja u jednu operaciju čitanja, na jednoj od kopija.
- Prevodi svaku operaciju pisanja u N operacija pisanja, po jednu na svakoj kopiji. Kontroler konkurentnosti na svakoj lokaciji sinhronizuje pristup kopijama – svaka promena mora da uspe na svim lokacijama ili ni na jednoj.

U slučaju otkazivanja neke lokacije

- čitanje je i dalje moguće
- pisanje nije moguće do oporavka neispravne lokacije

Protokol ROWA–A („Read One Write All Available“)

Razlika u odnosu na osnovni protokol:

- pisanje se izvodi ne na „svim“ nego na „svim raspoloživim“ kopijama

U slučaju otkazivanja neke lokacije

- čitanje je i dalje moguće
- pisanje je i dalje moguće
- neispravne lokacije mogu da postanu raspoložive tek nakon oporavka uz puno prepisivanje svih izmena nastalih tokom neoperativnog perioda

ROWA sa primarnom kopijom

Razlika u odnosu na osnovni protokol:

- jedna kopija se proglašava za „primarnu“
- ostale kopije su „rezervne“
- čitanje se izvodi samo sa primarne kopije
- pisanje se izvodi na primarnoj i svim raspoloživim rezervnim kopijama

Ovaj protokol

- ne podiže performanse čitanja
- unapređuje samo pouzdanost sistema

Ako primarna kopija postane neoperativna, izabrana rezervna kopija (po unapred određenoj liniji preuzimanja) postaje nova primarna

- da bi ovo bilo moguće potrebno je da bude moguće nedvosmisleno razlikovati neoperativnost primarne kopije od problema u komunikaciji
- u suprotnom se može dogoditi da postoji više primarnih kopija, što je fatalno po konzistentnost

Ako rezervna kopija postane neoperativna

- ona ne može postati primarna sve dok ne bude u potpunosti oporavljena
- može preuzeti neke uloge rezervne i nakon delimičnog oporavka

Iako je ovaj algoritam relativno jednostavan i ima nekih slabosti, ipak je jedan od najčešće upotrebljavanih

- naravno, samo u slučajevima gde je jedini cilj podizanje nivoa pozdanosti

ROWA sa tokenima „pravih“ kopija

Razlika u odnosu na osnovni protokol

- svaki podatak, u bilo kom trenutku vremena, ima ili jedan ekskluzivan token ili skup deljivih tokena
- operacija pisanja mora da dobije eksluzivan token
- operacija čitanja mora da dobije bar deljivi token

Konceptualno slično distribuiranom zaključavanju
Pri pisanju:

- kada kopija mora da izvrši operaciju pisanja, ona locira i uzima ekskluzivan token
- ako on ne postoji, ona locira i proglašava za neispravne sve deljive tokene i pravi novi ekskluzivan umesto njih

Pri čitanju

- kada kopija mora da izvrši operaciju čitanja, kopija locira deljivi token, kopira njegovu vrednost i pravi i čuva novi deljivi token
- ako ne postoji deljivi token, ona locira ekskluzivan token i konvertuje ga u deljivi

Konsenzus kvoruma (glasanje)

Za razliku od metoda ROWA

- dopušta pisanje na podskupu (kvorum pisanja) operativnih lokacija
- čitanja se izvode sa poskupa (kvorum čitanja) lokacija koji mora da ima neprazan presek sa kvorumom pisanja

Pravilo preseka kvoruma obezbeđuje da će svako čitanje moći da se odvija sa najsvežije pisanim vrednostima Za lokaciju koja učestvuje u operaciji se kaže da je glasala za operaciju.

Konsenzus kvoruma u uređenim mrežama

Savremeni algoritmi replikacije pored tolerisanja neispravnosti stavljaju u prvi plan i optimizaciju troškova. Pokušava se postizanje smanjivanja broja lokacija koje odlučuju u glasanju uvođenjem logičke strukture u skup lokacija.

10

Projektovanje baza podataka za podršku odlučivanju

Podrška odlučivanju obično podrazumeva pružanje informacija od strateškog značaja, odnosno, podrazumeva proces prikupljanja visoko-kvalitetnih informacija o poslovnim temama, analizu podataka u cilju praćenja trendova u poslovanju kao i dobijanje odgovora na ključna poslovna pitanja. Koriste se različiti nazivi da označe ovaj pojam:

- podrška odlučivanju (engl . ,,decision support“)
- podrška planiranju (engl . ,,planning support“)
- poslovna inteligencija (engl . ,,business intelligence“)
- poslovno obaveštavanje

10.1 Osnove skladištenja podataka

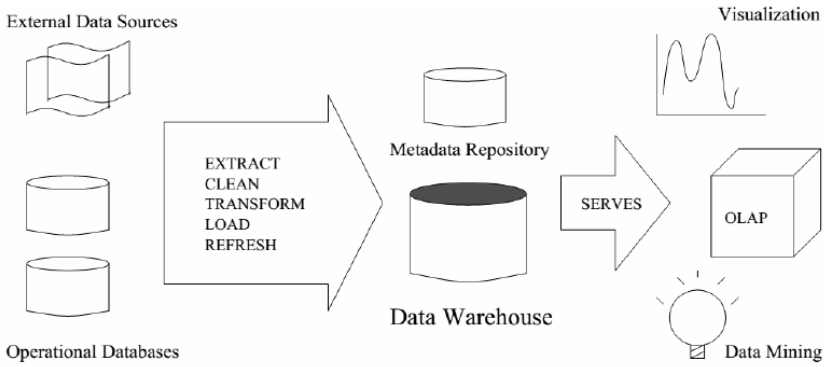
Skladište podataka (engl . ,,data warehouse“) je veliki prostor za prikupljanje (odlaganje) istorijskih podataka, koji se integrisano upotrebljavaju radi pružanja podrške odlučivanju. Osnovna namena im je analitička obrada podataka (takozvana ,,oflajn analitička obrada“). Podaci se obično prikupljaju iz različitih izvora a projektuju se u skladu sa specifičnostima.

Osnovni zahtevi koji stoje pred sistemima za skladištenje podataka su:

- poslovne informacije moraju biti lako dostupne,
- poslovne informacije moraju biti predstavljene konzistentno,
- sistem mora biti adaptibilan,
- sistem mora da ima obezbeđen kontrolisan pristup svim poverljivim informacijama,

- skladište podataka mora da služi kao osnova za podršku odlučivanju primenom tehnika poslovne inteligencije kao na primer, OLAP, istraživanje podataka i drugi.

Koncept podrške odlučivanju predstavlja skup softverskih alata i metodologija koji omogućavaju korišćenje podataka iz skladišta podataka i njihovo pretvaranje u informacije potrebne za donošenje adekvatnih poslovnih odluka [11].



Slika 10.1: Podrška odlučivanju

Poređenje transakcionih sistema i analitičkog izveštavanja

Transakcione sisteme, poznate kao OLTP (engl. On Line Transactional Processing) sistemi, odlikuju sledeće karakteristike:

- Složene strukture podataka.
- Normalizovani podaci, predstavljeni, najčešće u trećoj normalnoj formi.
- Visoke performanse, pre svega brzina čitanja, upisa i izmene slogova.
- Kratko vreme odziva.
- Raspoloživost krajnjim korisnicima.
- Pouzdanost.

Iako imaju brojne prednosti, transakcioni sistemi su se pokazali nepogodni i malo upotrebljivi kada je reč o analizi kvaliteta poslovnih informacija, praćenju trendova ili donošenju poslovnih odluka. Glavni nedostaci transakcionih sistema su [11]:

- *Kredibilitet podataka.* Nije retka pojava da iz dva različita organizaciona dela jedne kompanije izveštaji o nekom segmentu poslovanja budu dostavljeni rukovodstvu sa potpuno različitim rezultatima, na osnovu kojih se ne može doneti nikakav zaključak. Razlozi za to mogu da budu što su podaci izvedeni iz transakcionih sistema u različitim vremenskim trenucima, ili su se koristili različiti algoritmi za izvlačenje podataka, ili su koristili različite izvore podataka i načine njihove obrade i slično.

- *Produktivnost.* U slučaju potrebe za analizom veće količine podataka iz različitih organizacionih delova kompanije, potrebno je napraviti veliki napor u cilju objedinjavanja i usaglašavanja podataka, čime se efikasnost i produktivnost ozbiljno narušavaju.

- *Nemogućnost transformacije podataka u informaciju.* Ovo je ključni nedostatak klasičnih transakcionih sistema. Na primer, iz ovih sistema je gotovo nemoguće dobiti odgovor na uobičajeno bankarsko pitanje ,,Koliko se aktivnost određenog računa razlikuje ove godine u odnosu na prethodnih pet godina". Razlog za to je što su ovi sistemi obično neintegrisani i ne sadrže istorijske podatke, već rade na principu ,,živih", tekućih i trenutno važećih podataka.

Za razliku od transakcionih sistema, sistemi za skladište podataka se mogu okarakterisati kao [11]:

- *Poslovno orijentisani.* Za razliku od transakcionih sistema koji su aplikativno orijentisani, skladišta podataka se logički organizuju oko poslovnog subjekta (na primer, kupca, proizvoda, računa itd). Podaci o subjektu su uvek vremenski određeni (na primer, podaci na dnevnom, mesečnom ili godišnjem nivou).

- *Integrisani.* Podaci se prikupljaju iz različitih izvora, konvertuju se, reformatiraju, sumiraju i sortiraju a sve sa ciljem neutralizovanja nekonzistentnosti koje su postojale u izvoru podataka (u transakcionoj aplikaciji ili nekom drugom spoljnom izvoru). Ovaj proces integracije i transformacije zahteva vreme i ima svoju cenu.

- *Nepromenljivi.* Podaci se iz skladišta ne brišu i ne menjaju, već se periodično u serijama dodaju nove količine podataka.

- *Vremenski određeni.* Osnovna komponenta ključa u skladištima podataka je vremenska komponenta. Svaki slog u skladištu podataka je vremenski označen, odnosno, označava se vremenski trenutak u kom je taj slog bio važeći. Za razliku od transakcionih sistema, akcenat u sistemima za skladište podataka je naa istorijskim podacima, a manje na operativnim, dnevnim podacima.

Tabela 10.1 daje uporedan prikaz poređenja transakcionih sistema i sistema za skladište podataka.

Projektovanje skladišta podataka

Sistemi za skladištenje podataka su pre svega namenjeni analitičarima poslovanja. Zbog toga je važno, pri izradi ovih sistema, ući u njihov način razmišljanja, razumeti način na koji oni doživljavaju skladište podataka i utvrditi šta oni od skladišta podataka očekuju.

Jedna od metoda projektovanja koja se koristi u transakcionim sistemima je poznata pod nazivom ,,vodopad". Aktivnosti se uzastopno odvijaju jedna za drugom tako da kraj jedne inicira početak sledeće. Za razliku od toga, sistem za skladištenje podataka se najčešće razvija iterativno-inkrementalno.

Tabela 10.1: Poređenje transakcionih sistema i sistema za skladište podataka.

Osobine	OLTP	Skladišta podataka
Vreme odziva sistema	Milisekunde i sekunde	Sekunde, minuti, sati
Operacije nad bazom podataka	Upis, izmene, brisanje, čitanje	Uglavnom čitanje
Ažuriranje	Neprekidno pojedinačno ažuriranje	Periodično paketno ažuriranje
Priroda podataka	Tekući, trenutni podaci	Istorijski podaci
Organizacija podataka	Alikativno orijentisani	Subjekt orijentisani
Podaci	Normalizovani (više tabela sa po malo kolona)	Denormalizovani (malo tabela sa po mnogo kolona)
Obim podataka	Mali do veliki (do nekoliko GB)	Veliki i veoma veliki (od nekoliko GB do na stotine TB)
Aktivnosti u sistemu	Transakciona obrada	Alalitičke aktivnosti u cilju donošenja poslovnih odluka
Korisnici	Hiljade korisnika	Nekolicina korisnika

10.2 Životni ciklus skladišta podataka

- Prikupljanje i analiza zahteva
- Logičko projektovanje
- Fizičko projektovanje
- Distribuiranje podataka
- Implementacija, praćenje i menjanje

10.2.1 Prikupljanje i analiza zahteva SP

Ovaj korak, pre svega, obuhvata identifikaciju i analizu poslovnih procesa i subjekata u poslovanju, kao i odabir najbitnijih procesa i subjekata iz aspekta celokupnog poslovanja organizacije. Pri izboru poslovnih procesa koji će biti implementirani neophodno je voditi računa o tome koji procesi posebno utiču na efikasnost poslovanja, koji obezbeđuju najveći profit preduzeću, koji procesi imaju najpovoljniji odnos cene impelemntacije i koristi koju donose (engl. cost-benefit), koji procesi imaju najveći strateški značaj za firmu, koje vrsta odluka se mogu doneti na osnovu tih poslovnih procesa, koji se procesi mogu najlakše implementirati i koliko će dugo trajati implementacija svakog od procesa. Preporuka je da se u ovoj fazi sagleda celokupan poslovni sistem, bez obzira na inkrementalni pristup implementaciji sistema, kako bi se na početku postavile ,,granice“ modela podataka, pre same izrade modela.

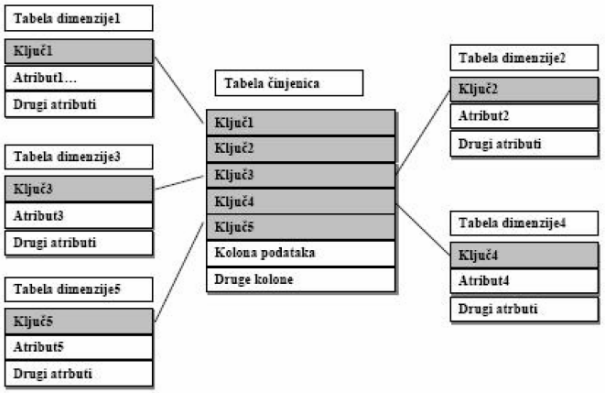
10.2.2 Logičko projektovanje SP

Logičko projektovanje skladišta podataka podrazumeva projektovanje tabela i pogleda skladišta podataka. Često se kao osnova koristi ,,dimenziono modeliranje“. Standardna tehnika koja se koristi u ,,dimenzionom modeliranju“ sistema za SP je modeliranje korišćenjem *zvezdastih (engl. star) shema* , ili nekih

njihovih modifikacija. Jedna od modifikacija je takozvana *shema pahuljice (engl. snowflake)*.

Zvezdasta shema

Zvezdastu strukturu čini jedna velika tabela činjenica (podataka, stavki, ...) koja je povezana sa više manjih dimenzionih tabela. *Tabela činjenica* (podataka, stavki, ...) sadrži denormalizovane osnovne posmatrane podatke. Obrada se uglavnom odvija nad tabelama činjenica. *Dimenzione tabele* su svojim primarnim ključem povezane sa delom složenog ključa tabele činjenica. Sadrže uglavnom opisne atribute i povezuju se sa tabelom činjenica radi pravljenja „čitljivih“ izveštaja. Kao ni tabele činjenica, dimenzione tabele nisu normalizovane. Denormalizacija i redundansa podataka u ovim modelima se uvodi u cilju poboljšanja performansi izvršavanja upita nad podacima iz skladišta.



Slika 10.2: Primer zvezdaste sheme opšteg tipa [11]

Osnovne karakteristike tabela činjenica su:

- Sadrže numeričke, kvantitativne podatke o poslovanju neophodne krajnjim korisnicima.
- Sadrže veći broj slogova u odnosu na dimenzione tabele
- Brzo se uvećavaju uzimajući u obzir količinu podataka koja se upisuje pri svakom novom ciklusu osvežavanja skladišta
- Podaci u njima mogu biti atomski, nedeljivi na sitnije delove, ali i izvedeni i sumarni podaci.
- Podaci su uglavnom aditivni, odnosno, nad njima mogu da se primenjuju agregatne funkcije
- Tabele činjenica su stranim ključem povezane sa primarnim ključem svake od povezanih dimenzionih tabela
- Mogu da sadrže oznake, takozvane indikatore događaja, koji se koriste za filtriranje podataka u tabelama činjenica

Osnovne karakteristike dimenzionih tabela su:

- Sadrže podatke koji se ređe menjaju u odnosu na podatke u tabelama činjenica
- Uglavnom sadrže opisne podatak koji se u upitima koriste za postavljanje uslova pri pretraživanju tabela činjenica
- Po obimu podataka ove tabele su manje od tabela činjenica
- Primarni ključ u dimenzionim tabelama je obično surogat ključ, veštački ključ sistemski generisan i nezavisan od prirodnog ključa. Primarni ključevi se ne menjaju čak ni kada dođe do promene prirodnog ključa u izvornom transakcionom sistemu. Primer prirodnog ključa je bar kod proizvoda u sistemu prodaje.
- Stranim ključem ove tabele su povezane sa tabelama činjenica.
- Jedna od dimenzionih tabela uvek je i vremenska dimenzija.
- Ove tabele su denormalizovane, odnosno vrednosti atributa su upisane redudantno.

locid	*city*	*state*	*country*
1	Ames	Iowa	USA
2	Chennai	TN	India
5	Tempe	Arizona	USA

Locations

pid	*pname*	*category*	*price*
11	Lee Jeans	Apparel	25
12	Zord	Toys	18
13	Biro Pen	Stationery	2

Products

pid	*timeid*	*locid*	*sales*
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26
12	2	2	45
12	3	2	20
13	1	2	20
13	2	2	40
13	3	2	5

Sales

Slika 10.3: Primer zvezdaste sheme

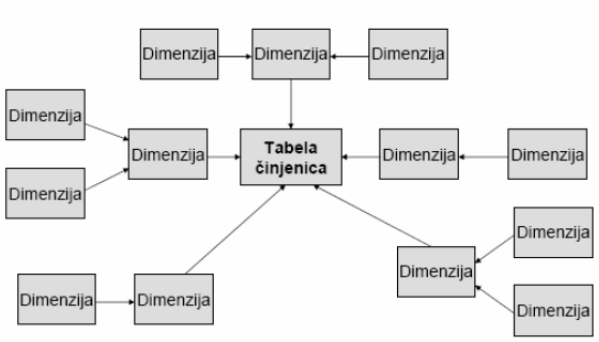
Zvezdaste sheme imaju sledeće karakteristike:

- Podržavaju multidimenzinalne analize.
- Poboljšavaju performanse i optimizovanost upita.

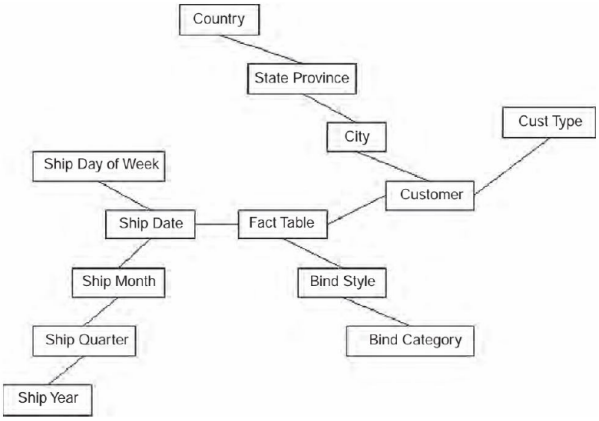
- Obezbeđuju intuitivan model sa stanovišta krajnjih korisnika.
- Obezbeđuju laku proširivost modela.

Shema pahuljice

Normalizacijom dimenzionih tabela zvezdaste sheme dobija se takozvani model pahuljice. Ovaj model je u manjoj meri prisutan u projektovanju sistema za skladištenje podataka.



Slika 10.4: Uopšteni prikaz sheme pahuljice [11].



Slika 10.5: Primer sheme pahuljice.

Hijerarhije. Tehnike ,,bušenja“ podataka

Hijerarhijski podaci se nalaze među dimenzionim podacima i svaka dimenzija može da sadrži jednu ili više hijerarhija. Primer hijerarhije kupaca u sistemu prodaje bi mogao da bude Kupac-Grad-Region-Država. Vremenska dimenzija najčešće sadrži višestruke hijerarhije. Na primer, jednu hijerarhiju čine: kalendarski dan, nedelja, mesec, kvartal, godina; drugu čine: fiskalna nedelja, fiskalni mesec, fiskalni kvartal, fiskalna godina za potrebe računovodstva.

Postojanje hijerarhija nad podacima u dimenzijama omogućava jednu od najvažnijih tehnika koje se primenjuju u analizama podaktaka u skladitištima, a to je bušenje (engl. drill) podataka. Bušenje omogućava dobijanje podataka na višem ili nižem nivou detaljnosti. U tom smislu razlikujemo:

- Bušenje naniže (engl. drill down). Ova tehnika podrazumeva prolaz do podataka veće granularnosti, odnosno višeg nivoa detaljnosti.
- Bušenje naviše (engl. drill up). Ova tehnika podrazumeva obrnut proces, odnosno prolaz do podataka manje granularnosti, odnoso nižeg nivoa detaljnosti.

Bušenje se može vršiti duž jedne hijerarhije ili prelaskom sa jedne na drugu hijerarhiju preko nehijerarhijskog atributa.

Agregati u skladištima podataka

Agregati su sumarne tabele činjenica koje sadrže agregirane slogove osnovnih tabela činjenica, odnosno onih tabela u kojima se nalaze najdetaljniji podaci. Jedina razlika koja postoji između osnovnih i sumarnih tabela činjenica je u nivou detaljnosti podataka.

Prednosti agregata su:

- Poboljšanje performansi upita, odnosno, obezbeđuju kraće veme odziva prilikom analitičkog izveštavanja
- Obezbeđuju optimizovano i poboljšano korišćenje računskih resursa, memorije, diska i drugo.
- Olakšavaju i ubrzavaju analitičke tehnike bušenja podataka.

Eksterni podaci u skladištima podataka

Eksterni podaci u skladištu podataka su, sa stanovišta jednog preduzeća, svi oni podaci koji ne postoje u sistemima tog preduzeća već u njega dolaze iz spoljnog okruženja i to, najčešće, u nestruktuiranom i nepredvidivom formatu te zahtevaju ozbiljne rekonstrukcije pre upotrebe. Ovakvi podaci najčešće dolaze iz časopisa, sa veb stranica, putem elektronske pošte i slično.

Neki od problema u radu sa eksternim podacima su:

- Eksterni podaci su nepredvidivi kako po pitanju izvora, tako i po pitanju vremena. To znači da mogu doći iz bilo kog izvora u bilo kom trenutku.
- Eksterni podaci su neprilagođeni i nekompatibilni sa internim podacima koji već postoje u sistemu za skladištenje. Potrebno ih je prestruktuirati, preformatirati i usaglasiti sa internim podacima po pitanju ključeva.
- Strukture ovih podataka su zahtevne, kako po pitanju prostora, tako i po pitanju njihove obrade. Ovde se, pre svega, misli na podatke kao što su slike i zvučni zapisi.

Zahtevni eksterni podaci mogu da se čuvaju u samom skladištu podataka (ukoliko je to pogodno i nije previše skupo) ali mogu da se čuvaju i izvan baze skladišta, a da se u metapodacima čuva informacija o tome gde se oni nalaze.

10.2.3 Fizičko projektovanje skladišta podataka

Fizička implementacija formiranog dimenzionog modela je poslednji korak pre puštanja sistema za skladištenje podataka u produkcioni rad. Na ovom nivou dimenzioni model se dopunjava specifičnostima konkretne baze podataka. Dok je dimenzioni model nezavisan od platforme i tehnologije, fizički model zavisi od osobina konkretne baze podataka i servera na kom će biti implementiran.

Kako bi se poboljšale performanse sistema za skladištenje podataka, koriste se projektantske tehnike koje imaju za cilj optimizaciju sledećih aktivnosti:

- punjenje podataka,
- pristup podacima,
- čuvanje podataka,
- arhiviranje podataka,
- brisanje podataka iz sistema,
- praćenje podataka.

Neke od pomenutih projektantskih tehnika su: pažljivo određivanje nivoa granularnosti podataka, spajanje u jednu tabelu činjenica koje se koriste u istom upitu, uključivanje redundantnih podataka u dimenzioni model, implementiranje sumarnih tabela, definisanje ključeva, formata podataka, null vrednosti i veza između podataka u sistemu.

Tehnike koje se koriste u cilju dodatnog poboljšanja performansi sistema za SP su particionisanje, indeksiranje i paralelizam.

Particionisanje

Pod particionisanjem podataka podrazumevamo razdvajanje podataka u zasebne, manje fizičke jedinice sa nezavisnim pristupom. Osnovne metode particionisanja su:

- *Horizontalno particionisanje.* Vrši se po grupama redova, obično po lokacijama ili vremenskim periodima. Na primer, u sistemima osiguravajućih agencija, particije mogu biti određene godinom i linijom osiguravajućih polisa, na primer: 2001/zdravstvene polise, 2002/zdravstvene polise, 1999/životno osiguranje, 2000/životno osiguranje i drugo.
- *Particionisanje intervala.* Slično je horizontalnom particionisanju i zasniva se na grupisanju podataka prema intervalima vrednosti određenih atributa.
- *Vertikalno particionisanje.* Vrši se po grupama kolona, obično po različitim karakteristikama kolona. Na primer, ukoliko se sadržaji nekih kolona tabela retko menjaju onda se takve kolone grupišu u posebne particije i na taj način se razdvajaju od kolona koje se često menjaju.
- *Heš particionisanje.* Sistem za upravljanje bazom podataka automatizovano, korišćenjem heš funkcije raspoređuje podatke po particijama.

Indeksiranje

Indeksiranje je u sistemima SP od posebne važnosti zbog velike količine podataka koja se u njima nalazi. U ovim sistemima je broj indeksa znatno veći nego u transakcionim sistemima. Indeksima se povećava brzina izvršavanja upita.

Strategija indeksiranja je zasnovana na sledećim pravilima:

- Indeksi se postavljaju nad kolonama koje se često pojavljuju u WHERE klauzuli upita (ne moraju biti deo primarnog ili stranog ključa).
- Sistemi za SP su mnogo fleksibilniji po pitanju ograničavanja broja indeksa od transakcionih sistema.
- Preporuka je da se indeksi ne koriste nad tabelama sa malim brojem slogova, jer su u takvim slučajevima prednosti indeksa zanemarljive u odnosu na cenu održavanja.

U sistemima za SP koriste se indeksi spajanja (za efikasno povezivanje tabele činjenica i dimenzionih tabela), bit-mapirani indeksi (za efikasno pretraživanje tabele činjenica), B-drvoidni indeksi i drugo.

Paralelizam

Većina sistema za upravljanje bazama podataka danas ima mogućnost paralelnog izvršavanja upita. Složeni zahtevi se, u cilju povećanja brzine, mogu izvršavati istovremeno na jednom ili više procesora. Paralelni režim rada omogućava paralelno punjenje podataka u skladište, paralelno formiranje tabela i indeksa, sortiranje, uzimanje sigurnosnih kopija (engl. backup), oporavak sistema (engl. recovery) i drugo.

10.2.4 Distribuiranje podataka SP

Obično se teži da SP budu centralizovana, ali ako je neophodno onda se distribuiraju. Projektuju se particionisanje, replikacija i raspoređivanje podataka.

10.2.5 Implementacija, praćenje i menjanje SP

Ovaj korak obuhvata pravljenje tabela, pogleda, skriptova i drugih metapodataka. Procedure i skriptovi za ažuriranje podataka:

- izdvajanje podataka
- prečišćavanje podataka
- transformisanje podataka
- punjenje podataka
- osvežavanje indeksa
- r azvoj korisničkih aplikacija

Praćenje i unapređivanje rada

- performanse
- sadržaj i kvalitet izveštaja

11

Teorema CAP

11.1 Svojstva (uslovi) CAP

Pri pravljenju distribuiranih sistema (ne samo baza podataka) kao najvažniji ciljevi se ističu tri važna svojstva – **CAP**:

- konzistentnost (consistency - **C**)
- raspoloživost (availability - **A**)
- prihvatanje razdvojenosti (partition tolerance - **P**)

Konzistentnost

- ,,Konzistentan sistem ili funkcioniše kao celina ili ne funkcioniše uopšte".
- ,,Sva čitanja, na svim čvorovima, moraju da daju isti rezultat".
- ,,Rezultat operacije (čitanja) nikada ne zavisi od čvora na kome se izvršava".

Raspoloživost

- ,,Sistem je uvek raspoloživ".

Raspoloživost se praktično definiše kao odzivnost sistema u nekim garantovanim granicama (u ovom kontekstu se razmatra samo vreme odziva). Sistem obično nije raspoloživ upravo kada je potreban. U vreme kada je raspoloživost najpotrebnija, onda je i najteže ostvariva, zato što je tada sistem najviše opterećen. Ako je sistem raspoloživ kada nije potreban, to nema značaja.

Tolerancija razdvojenosti

- ,,Nijedan skup problema, osim potpunog otkazivanja, ne sme da proizvede neispravan odziv sistema"

Drugim rečima, sistem mora da prihvata delimične otkaze komunikacije i da nastavlja ispravan rad (u ovom kontekstu se razmatra ispravnost odziva) Povremeni prekidi komunikacije među čvorovima su neizbežni. Razdvojenost (partition) je stanje komunikacione mreže u kome su delovi sistema (obično

usled kvara) podeljeni na particije (dva ili više razdvojenih skupova čvorova) između kojih ne postoji komunikacija

Hipoteza CAP

Eric Brewer, ,,Towards Robust Distributed Systems“, Berkley (2000).

Hipoteza 2/3: ,,Nije moguće definisati sistem koji zadovoljava sve CAP uslove (konzistentnost, raspoloživost i toleranciju razdvojenosti).“ Moguće je definisati sistem koji zadovoljava izabrana dva od ovih uslova.

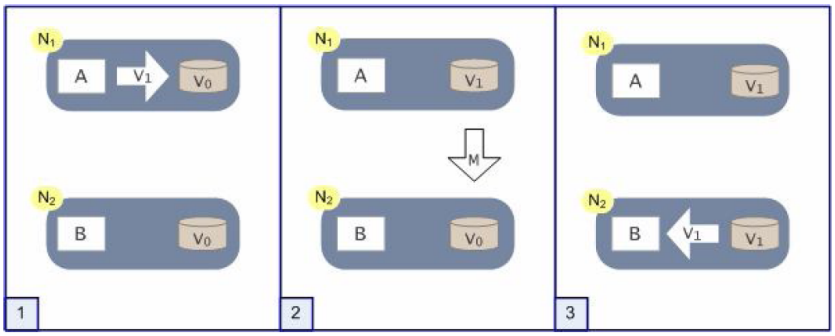
Teorema CAP

Teoremu CAP su dokazali Gilbert, Lynch, MIT (2002). Ipak, dokaz nailazi na osporavanja, uz tvrdnje da je ,,problematičan“ zbog ,,spornog“ načina formalnog definisanja CAP uslova. Predstavićemo samo ideju dokaza.

Pretpostavke: Neka imamo dva čvora i na njima repliciran podatak V:

1. neka na čvoru 1 transakcija A ažurira V
2. promena se porukom M propagira na čvor 2
3. neka na čvoru 2 nešto kasnije transakcija B čita V

Na sledećoj slici je predstavljeno željeno ponašanje.



Pretpostavimo da poruka M nije stigla na odredište? U tom slučaju se radi o razdvojenosti delova sistema.

U osnovi imamo tri moguća ponašanja sistema:

- transakcija A se poništava, što znači da sistem ne prihvata razdvojenost svojih delova;
- transakcija A se uspešno nastavlja (i zatim završava), što znači da sistem nije konzistentan;
- transakcija A čeka na uspešno slanje poruke M, što znači da sistem nije raspoloživ (nepoznato vreme odziva).

U svakom od slučajeva nije ispunjen jedan od uslova.

11.2 Kompromisi

Pri projektovanju (ili konfigurisanju) distribuiranog sistema neophodno je napraviti neki kompromis:

- odbacivanje tolerancije razdvojenosti
- odbacivanje raspoloživosti
- odbacivanje konzistentnosti
- ublažavanje uslova (odnosno odbacivanje više strogo definisanih uslova)
- zasnivanje sistema na drugačijem skupu uslova
- projektovanje zaobilaznih puteva

Odbacivanje tolerancije razdvojenosti

Ovo zapravo znači da ,,pristajemo da sistem ne radi u slučaju razdvojenosti". Jedan način prevazilaženja je da sve bude na jednoj mašini. Alternativa je da se višestrukim umrežavanjem razdvojenost (a time i otkaz sistema) svede na najmanju moguću meru. Mana ovog pristupa je da su obe alternative veoma skupe.

Imajući u vidu da se distriburana rešenja koriste samo onda kada su neophodna zbog obima podataka i poslova, ovaj vid kompromisa se retko pravi.

Odbacivanje raspoloživosti

Ovo zapravo znači da ,,u slučaju razdvojenosti ne garantuje se vreme odziva". Problem se redukuje pažljivim projektovanjem sistema, odnosno uspostavljanjem što niže sprege među čvorovima.

Sistem koji nije raspoloživ je praktično neupotrebljiv. Zbog toga se ovaj pristup koristi relativno retko. Dakle, ako je konzistentnost primarna i ako je tolerancija razdvojenosti nezaobilazna, onda se teži niskoj spregnutosti među čvorovima.

Odbacivanje konzistentnosti

To znači da ,,dopuštamo da isti upit daje različite rezultate na različitim čvorovima". Ovu opciju koristimu u slučaju ako su razlike prihvatljivije nego niska raspoloživost i ako je učestalost pojavljivanja svedena na prihvatljivu meru, odnosno ako je cena nekonzistentnosti prihvatljiva. Konzistentnost je jedan od osnovnih uslova za uspešan rad baza podataka. Ipak, postoje slučajevi kada nije primarna, na primer:

- pretraživači – ako se nešto ne pronalazi na svakom čvoru, to nije veliki problem;
- čak i prodavnice – ako se mali broj proizvoda proda po staroj ceni, neće propasti firma.

Izmenjen skup uslova - BASE

Pojmovi iz teoreme počivaju na uobičajenim konceptima rada sa bazama podataka, odnosno postoji zavisnost sa osobinama transakcija – ACID (Atomicity Consistency, Isolation, Durability). Može se definisati i drugačiji skup uslova, manje oštar, u kom slučaju se sve odgovarajuće osobine mogu uskladiti. Predlog je koristiti BASE umesto ACID.

BASE

- **B**asically **A**vailable – ne garantuje se raspoloživost odgovora, već samo sistema. To znači da ako korisnik ne može da se dobije odgovor na zahtev, bar će se dobiti obaveštenje o tome.

- **S**oft-state – stanje sistema može da se menja čak i kada nije u toku nijedna transakcija, na primer radi postizanja konzistentnosti.

- **E**ventually consistent – žrtvuju se garantovana stalna konzistentnost i izolovanost transakcija zarad raspoloživosti. Sistem će u nekom trenutku (eventually) postati konzistentan ali će raditi i davati odgovore (potencijalno različite) do tada.

Suština koncepta je u odloženom usklađivanju sadržaja na različitim čvorovima. Konzistentnost se ostvaruje, ali ne obavezno u okviru iste transakcije (eventual consistency). Sadržaj čvorova se menja i van odvijanja transakcija, a u cilju njihovog usklađivanja (soft state). Ako sistem nije u potpunosti raspoloživ, čvor može da pokuša da pruži manje pouzdan ili samo delimičan odgovor, uz odgovarajuću informaciju o tome (basically available).

Projektovanje zaobilaznih puteva

Projektom sistema se mogu u zavisnosti od uslova žrtvovati različite stvari. U zavisnosti od vrste transakcija bira se šta se žrtvuje.

Na primer, u slučaju prodavnice nije veliki problem ako cena proizvoda privremeno nije ujednačena, osim ako se radi o veoma skuupom proizvodu ili je učestalost odstupanja veoma visoka.

Nije problem ni ako evidentiran broj proizvoda u magacinu privremeno ne odgovara stvarnom stanju, ako je kupac unapred obavešten o takvoj mogućnosti i ako se to ne dešava često i ne traje dugo. U suštini, projektovanje zaobilaznih puteva se svodi na BASE.

Zaključak

Činjenica da nije moguće napraviti savršen distribuiran sistem (u odnosu na skup uslova ACID) ne znači da nije moguće napraviti sistem koji zadovoljava realno prihvatljive zahteve (skup uslova BASE). Mnogi veliki poslovni sistemi danas koriste distribuirane baze podataka, što potvrđuje da su relaksirani uslovi često prihvatljivi. Savet je pažljivije ući u projektovanje i preciznije definisati zahteve.

11.3 Projektovanje sa BASE uslovima

U osnovi se svodi na nekoliko novih aktivnosti:

- Funkcionalna dekompozicija podataka u fragmente — ako pretpostavljamo da se već projektuje distribuirana baza podataka, ovde samo malo preciznije određujemo uslove za definisanje fragmenata.
- Implementacija transakcija u BASE uslovima
- Određivanje uslova replikacije

Postoje i drugačiji pristupi ali je ovaj među jednostavnijim i verovatno među najzastupljenijim.

Dekompozicija

Skup podataka se deli na fragmente koji predstavljaju najmanje moguće funkcionalne celine, unutar kojih moraju da važe ACID uslovi i među kojima je dovoljno da važe BASE uslovi.

Podela transakcija na sinhroni i asinhroni deo

Svaka transakcija se locira u jednom matičnom fragmentu. Promene podataka u matičnom fragmentu se implementiraju sinhrono, odnosno na uobičajen način u okviru transakcije. Promene podataka u drugim fragmentima se u okviru transakcije samo evidentiraju pri čemu evidencija mora da zadovoljava ACID uslove. Evidentirane izmene u drugim fragmentima se sprovode asinhrono.

Određivanje uslova replikacije

Pretpostavlja se da među replikama važe BASE uslovi. Sinhronizacija replika se odvija asinhrono, van transakcija.

Projektovanje sa BASE uslovima, primer

Podela na fragmente:

- Fragment A: podaci o proizvodima
- Fragment B: podaci o prodaji

Transakcija ažuriranja cene proizvoda u režimu ACID:

```
BEGIN TRANSACTION
  UPDATE Product SET Price =: new_price
    WHERE ProductID =: product_id
  UPDATE CartItem SET Price =: new_price
    WHERE ProductID =: product_id
    AND CartID in (SELECT CartID FROM Cart WHERE Status = 'OPEN')
END TRANSACTION
```

Ista transakcija u režimu BASE, locirana u fragmentu A:

```
BEGIN TRANSACTION
  UPDATE Product SET Price =: new_price
    WHERE ProductID =: product_id
  QUEUE ADD
  UPDATE CartItem SET Price =: new_price
    WHERE ProductID =: product_id
    AND CartID in (SELECT CartID FROM Cart WHERE Status = 'OPEN')
END TRANSACTION
```

Posledice opisanog postupka

- Svi upiti u okviru jednog fragmenta su lokalno konzistentni, dakle, u okviru fragmenata su ispoštovana sva pravila integriteta.
- Rezultati upita na različitim replikama su potencijalno nekonzistentni, odnosno, više replika može da da različite rezultate.
- Rezultati logički povezanih upita na različitim fragmentima mogu da budu nekonzistentni.
- Fragment je u osnovi raspoloživ i u slučaju particionisanja, funkcionalan je i može da odgovara na neke upite.
- Svaka promena podataka se u nekom trenutku propagira i na replike i na druge povezane fragmente.
- Baza podataka ispunjava BASE uslove.

Konflikti

- Osnovni oblik: ako se dve replike istog podatka nezavisno menjaju, pitanje je koja verzija je „ispravna“ i kako uspešno izvesti usklađivanje.
- Složeniji oblik: kada su u pitanju redudantni podaci a ne sa replike.

Razrešavanje konflikata

Više načina rešavanja:

- Zabrana menjanja podataka u slučaju particionisanja. Posledica toga je umanjena raspoloživost.
- Definisanje hijerarhija među redundantnim kopijama konkretnih vrsta podataka. Definisanje primarnih i sekundarnih fragmenata za sve podatke. Samo na primarnim se izvode transakcije u odnosu na te podatke na sekundarnim se samo propagiraju (odložene) izmene. U primeru sa cenama, za cenu je primarni fragment A. Ovakav pristup ne umanjuje upotrebljivost, ali komplikuje implementaciju.
- Višestruke verzije podataka (MVCC). Ovakav pristup je alternativa zaključavanju. Za svaku kopiju podatka se vodi verzija. Ovo je povezano sa konceptima optimističkih transakcija i optimističke replikacije. Konflikti mogu automatski da se prepoznaju, ali ne i da se otklone.

12

Nerelacione baze podataka

Pojam nerelacionih baza podataka

Relacionim bazama podataka su prethodile baze podataka zasnovane na drugim modelima podataka: hijerarhijske, mrežne, ... Ipak, danas se termin nerelacione baze podataka (uglavnom) ne odnosi na njih.

U savremenom razvoju softvera termin nerelacione baze podataka se odnosi na sisteme za upravljanje kolekcijama podataka koje:

- nemaju strogu statičku strukturu podataka,
- nemaju iscrpnu proveru uslova integriteta,
- ne koriste upitni jezik SQL.

Pitanje koje se može postaviti je: Kakva je onda korist od njih?

Slabosti relacionih baza podataka

Relacione baze podataka imaju mnoge kvalitete koje nećemo sada nabrajati. Medjutim, ti kvaliteti imaju cenu:

- relativno visoka cena čitanja – zbog neredundantnosti i stroge strukture podataka, odnosno, zbog čestog spajanja podataka
- otežano distribuiranje – pre svega zbog ,,ultimativne“ konzistentnosti podataka
- skupa promena strukture – zbog povezanosti strukture sa upotrebom i optimizacijama

Primer: Globalne veb aplikacije

Današnje globalno dostupne aplikacije na vebu postavljaju unekoliko specifične izazove pred baze podataka:

- konkurentni korisnici (transakcije ili čitanja) – milioni njih,
- količina podataka – dnevna proizvodnja terabajta i petabajta podataka,

- obrada – svaka aktivnost korisnika zahteva neku obradu podataka, makar zbog praćenja navika korisnika,
- opterećenje – nepredvidiv porast opterećenja, neravnomerno opterećenje,
- velika dinamičnost – neprekidno dodavanje novih mogućnosti, promena postojećih komponenti.

Pri distribuiranju RBP performanse ne rastu dovoljno brzo, zapravo, ne rastu ni blizu linearno a opterećenje raste čak i eksponencijalno. Nije jednostavno dodavati i sklanjati čvorove. Činjenica da je shema normalizovana povlači to da zahteva spore operacije spajanja, zahteva vrlo pažljivo fragmentisanje i repliciranje, otežava menjanje...

Olakšica:

- Što je aplikacija veća, to je veći deo te aplikacije obično besplatan.
- Ako je aplikacija besplatna, može nešto da se žrtvuje.
- Obično se relativno bezbolno podnosi umereno žrtvovanje konzistentnosti.

RBP i teorema CAP

Osnovne karakteristike RBP su nastale radi obezbeđivanja čvrste i stabilne strukture baze podataka. Pitanja koja se postavljaju su: Šta ako se neka od osnovnih karakteristika relacionih baza podataka oslabi da bi se prevazišli problemi koje opisuje teorema CAP? Da li je isplativo plaćati cenu RBP, ako se pri tome ipak odustaje od nekih njihovih kvaliteta?

Kompromisi

Svet RBP je beskompromisno orijentisan prema pouzdanosti i konzistentnosti. U svetu DBP kompromisi su neophodni (teorema CAP). U domenu baza podataka danas je najaktivnija oblast upravo razvoj različitih vidova nerelacionih SUBP (NRSUBP, NRBP) – traži se prava mera kompromisa.

Korak unazad?

NRBP su postojale pre relacionih ... da li je, onda, njihovo oživljavanje korak unazad? Neki od savremenih NRSUBP imaju sličnosti sa starim nerelacionim modelima podataka ali većina nema.

Kada se razmišlja nerelaciono?

Neko može razmišljati nerelaciono zato što:

- ... ne poznaje dobro RBP, pa misli da RBP nisu rešenje čak i kada jesu,
- ... ima problem koji ne može da reši pomoću RBP.

Šta to znači u praksi?

Razmatraju se efikasna nerelaciona rešenja

- ako je potrebno da se ostvari
  - jednostavnije i efikasnije distribuiranje podataka
  - lako dodavanje čvorova
  - visoka raspoloživost
  - slobodna (ili bar fleksibilnija) struktura podataka
- ako je prihvatljivo da se žrtvuje
  - određen nivo konzistentnosti
  - automatska provera integriteta
- "ciljevi" i „žrtve" obično mogu da se parametrizuju

Definicija?

Nema jedinstvene definicije, ali može se reći da je NRBP baza podataka koja:

- ne počiva na relacionom modelu podataka – ili ga se bar ne drži čvrsto
- lako se distribuira
- horizontalno je skalabilna – odnosno, lako podnosi značajne promene sheme

Druge česte karakteristike:

- bez statičke sheme
- laka replikacija
- jednostavan API
- eventualna konzistentnost – počiva na skupu uslova BASE a ne ACID
- pretpostavlja izuzetno velike količine podataka

NoSQL

Često se NRBP označavaju kao NoSQL baze podataka,

- izvorno, odstupanje od SQL-a, kao simbola RBP – „no SQL"
- danas mnoge teže da imaju jezik nalik na SQL ... pa se često tumači kao „not only SQL"

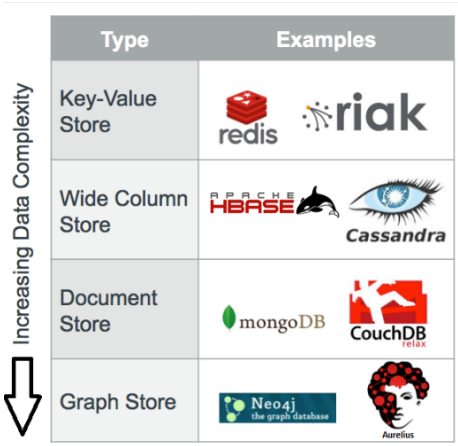
Tipični problemi i alternativna rešenja

- Složene strukture podataka u specifičnim domenima – objektne baze podataka
- Visok nivo distribuiranja – različite nerelacione baze podataka
- Slobodna ili fleksibilna struktura podataka – različite nerelacione baze podataka
- Neophodne izuzetno visoke performanse – memorijske baze podataka
- Ogromne količine podataka niske složenosti – različite vrste matričnih baza podataka

12.1 Vrste nerelacionih BP

Osnovne vrste nerelacionih baza podataka su:

- parovi ključeva i vrednosti
- skladišta širokih kolona
- skladište dokumenata
- grafovske baze podataka



Baze parova ključeva i vrednosti
(engl. key-value databases)

Struktura:

- svaka kolekcija podataka je jedna heš tabela
- podacima se pristupa isključivo na osnovu poznatog ključa
- vrednosti su u načelu jednostavne ili vrlo nisko strukturirane
- ako vrednosti imaju visoku složenost, obično se BP klasifikuje kao baza sa proširivim slogovima ili baza dokumenata

Doprinosi:

- podržavaju veoma velike skupove podataka
- veoma su brze
- obično podržavaju automatsko repliciranje i horizontalno particionisanje kolekcija

Slabosti:

- podrazumeva se visok nivo redundantnosti
- složene strukture se implementiraju velikim brojem kolekcija
- ako su podaci „gusto“ povezani, efikasnost drastično opada
- u osnovi nemaju mehanizme za očuvanje integriteta – često ne obezbeđuju čak ni atomičnost transakcija
- uslov traženja je isključivo fiksna vrednost ključa ili raspon vrednosti ključa

Primeri:

- Redis, Riak, Voldemort, Oracle NoSQ

Baze sa proširivim slogovima (engl. wide-column databases, extensible record stores)

Struktura:

- u osnovi slično kao baze parova ključeva i vrednosti
- vrednost predstavlja kolekciju velikog broja parova imena i vrednosti
- obično broj parova bar načelno nije ograničen
- sve skupa izgleda kao baza parova ključeva i vrednosti sa dve dimenzije

Doprinosi:

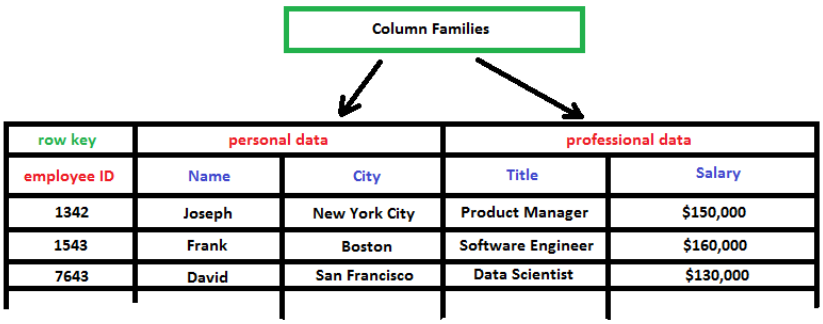
- podržavaju veoma velike skupove podataka – i velike podatke i veliki broj podataka
- veoma su brze, osim u nekim slučajevima vrednosti sa veoma složenom strukturom
- većina podržava automatsko repliciranje i horizontalno particionisanje kolekcija

Slabosti:

- sve kao za baze parova ključeva i vrednosti

Primeri:

- Cassandra, BigTable, Druid, Accumulo, HDFS (Hadoop Distributed File System), HBase



Baze dokumenata
(engl. document databases)

Struktura:

- u osnovi liči na bazu parova ključeva i vrednosti
- svaki dokument predstavlja vrednost kome se dodeljuje ključ
- dokumenti se zapisuju u strukturiranim (XML, YAML, JSON, BSON,...) ili nestrkturiranim oblicima (npr . PDF)
- svaki dokument ima metapodatke — obično nestrukturirane ili sa vrlo fleksibilnom strukturom
- telo dokumenta (ako je strukturiran) i metapodaci se automatski interno strukturiraju — korisnik ne mora ništa da zna o tome

Doprinosi:

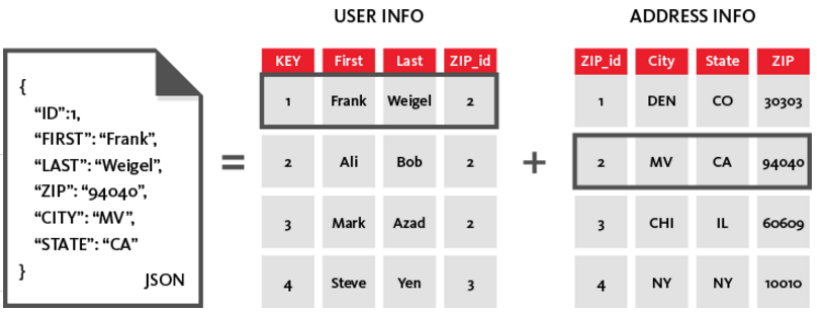
- obično veoma jednostavan i efikasan rad
- obično podržavaju bar poluautomatsko repliciranje i horizontalno particionisanje kolekcija – često samo replikacija tipa glavni-podređeni

Slabosti:

- relativno ograničen domen primene
- mnoge implementacije ne omogućavaju ad-hok upite i menjanja podataka
- neke implementacije ne trpe veliku učestalost menjanja podataka

Primeri:

- MongoDB, CouchDB, MarkLogic, RavenDB



Slika 12.1: Izvor: Couchbase website.

Grafovske baze podataka
(engl. graph databases)

Struktura:

- bazu čine čvorovi i veze među njima
- nema čvrste sheme, podaci imaju veoma slobodnu strukturu
- akcenat je na odnosima između podataka, a ne na strukturi podataka

Doprinosi:

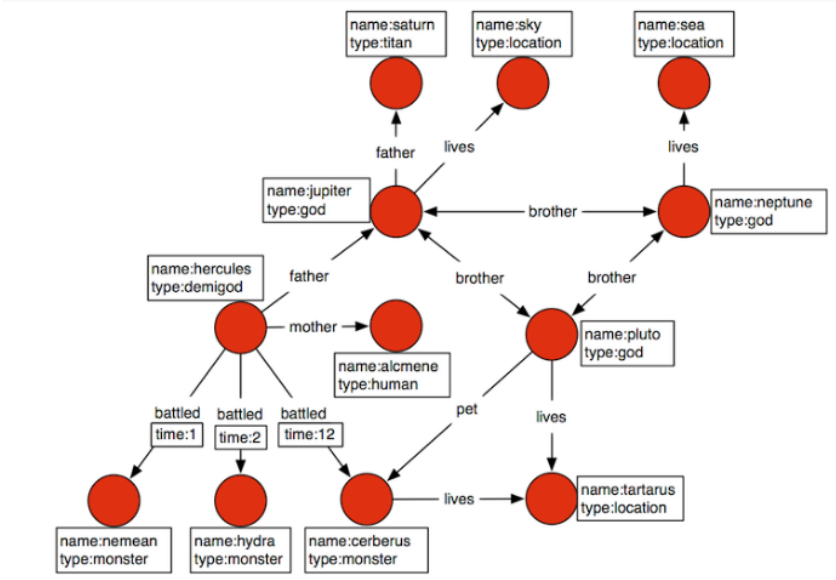
- nasuprot drugim nerelacionim BP, veoma su efikasne kada su u pitanju uobičajene operacije sa grafovima
- neke podržavaju transakcije i ACID režim uslova
- obično samo replikacija tipa glavni- podređeni

Slabosti:

- relativno ograničen domen primene
- nisu pogodne van svog domena

Primeri:

- Neo4J, OrientDB, ArangoDB, Allegro, Virtuoso



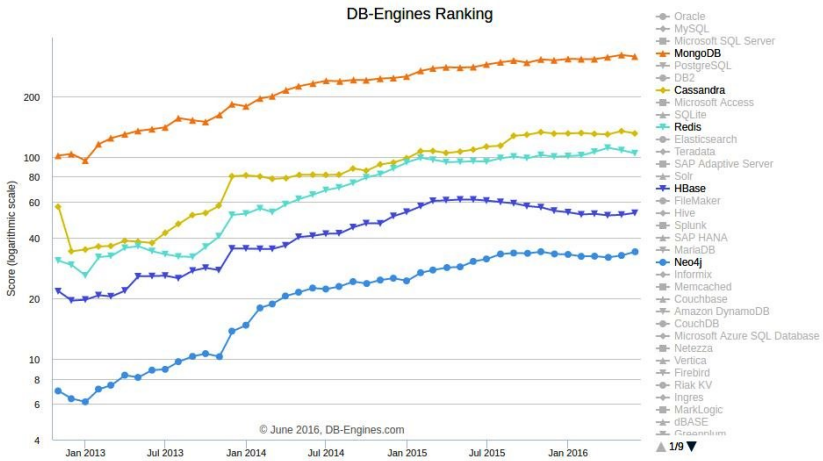
Slika 12.2: Izvor: Aurelis Github blog.

Još?

Prepoznaju se još neke kategorije baza podataka

- Višemodelne - predstavljaju spoj više modela
- Višedimenzione baze podataka
- Rešetkaste baze podataka
- XML baze podataka
- Objektne baze podataka
- Naučne baze podataka
- i druge ...

Prilično iscrpan pregled nerelacionih baza podataka postoji na stranici: http://nosql-database.org/



Slika 12.3: Izvor: DB-Engines Ranking.

12.2 Apache Cassandra

- Nerelacioni SUBP
- Primarno namenjen za DBP
- Dinamična shema
- Počiva na proširenom modelu kataloga – tj . baza parova ključeva i vrednosti
- Inicijalno razvijen od strane Facebook-a
- Projekat otvorenog koda
- Jedan od glavnih projekata fondacije Apache
- Među najrasprostranjenijim NRSUBP-ovima

Osnovni pojmovi

- Cassandra počiva na modelu kataloga.
- Osnovni pojmovi su:
  - kolona
  - familija kolona
  - superkolona
  - familija superkolona
  - ključ
  - prostor ključeva

Kolona

- jedna vrednost, praćena vremenom poslednje izmene
- pojam kolone je blizak pojmu atributa kod RBP
- trojka (ime, vrednost, vreme izmene)

```
{
name: ,,prezime‘‘,
value: ,,Petrović‘‘,
timestamp: 123456789
}
```

- pojednostavljen zapis:

```
prezime:  ,,Petrović‘‘
```

Superkolona

- složena vrednost, bez vremena poslednje izmene
- sadrži jednu ili više kolona
- ne postoji ekvivalent kod RBP, nešto kao složeni atribut

```
{
  name: "licniPodaci",
  value: {
    ime: { name: "ime", value: "Goran",
           timestamp: 123456789 },
    prezime: { name: "prezime", value: "Petrović",
               timestamp: 123456789 },
    grad: { name: "grad", value: "Smederevo",
            timestamp: 123456789 }
  }
}
```

Familija kolona

- struktura koja može da sadrži veći broj redova (konceptualno neograničeno)
- preslikava ključ u red
  - red = skup kolona
  - red je sličan superkoloni
- pojam familije kolona je blizak pojmu tabele kod RBP
- konceptualno predstavlja katalog redova

```
Autori : {
  ivoAndric: {
    // pojednostavljen zapis, bez naziva i TS
    ime: "Ivo",
    prezime: "Andrić",
    ...
  },
  goranPetrovic: {
    ime: "Goran",
    prezime: "Petrović",
    ...
  },
}
```

Prostor ključeva

- struktura koja sadrži više familija kolona ili superkolona
- konceptualno odgovara pojmu baze podataka ili sheme kod RBP

Prošireni model kataloga

Cassandra ima dva nivoa ugneždenosti:

- prvi nivo je obavezan
  - čini ga par (ime kolone, kolona)
  - jedan red (slog, record) se sastoji od proizvoljnog broja parova
  - parovi su uređeni samo po imenu kolone
  - red ima oblik kataloga
  - red MORA da ima bar jednu kolonu
- drugi nivo je opcion
  - umesto da drugi element para bude kolona, to može biti kolekcija parova – superkolona

Imena kolona

Imena kolona predstavljaju istovremeno

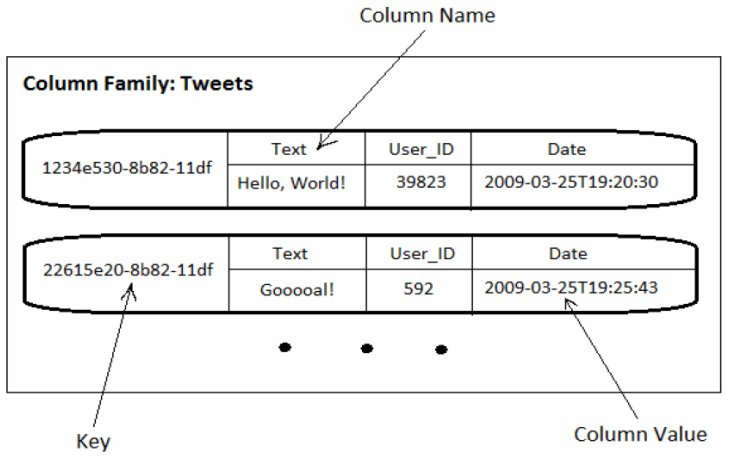
- ključeve i
- vrednosti

Svaki red (ili superkolona) može da sadrži proizvoljno mnogo kolona

- nazivi kolona mogu da imaju ulogu vrednosti
- vrednosti kolona mogu da budu prazne

Primer: tvitovi

- Svaka poruka je poseban red
  - ključ reda je kod vremena pisanja poruke
  - kolone odgovaraju atributima i predstavljaju različite elemente opisa poruke

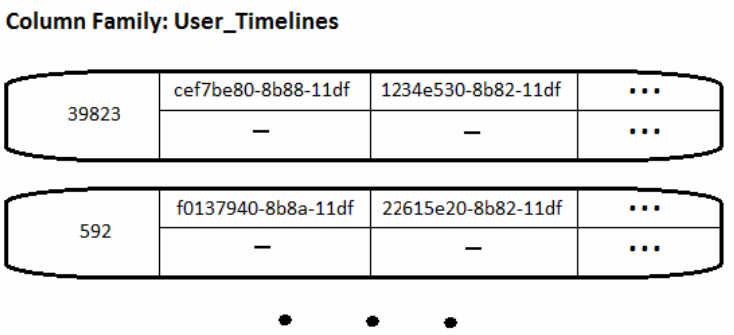


- Ovaj primer liči na jednu tabelu relacione baze:
  - jedan red familije kolona odgovara redu tabele
  - jedna kolona odgovara jednom atributu reda
  - redovi imaju ujednačen format, odnosno kolone sa istim nazivima (tako je u ovom primeru, ali NE MORAJU u opštem slučaju da imaju isti format)

Primer: tvitovi jednog korisnika

- Jedan red čine ključevi poruka jednog korisnika

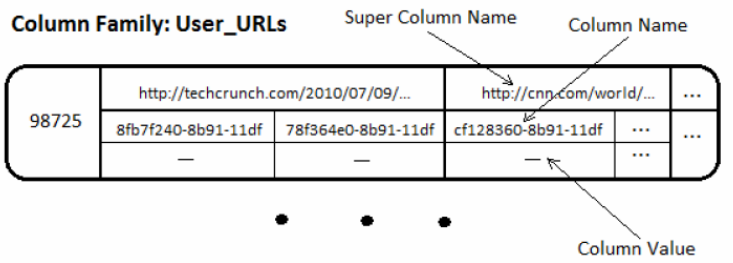
  - Svaka poruka predstavlja posebnu kolonu koja je označena kodom vremena pisanja. Kolone su uređene po imenu, pa time i po vremenu pisanja.
  - Vrednosti kolona su prazne – potreban je samo spisak ključeva poruka.



- Ovaj primer nimalo ne liči na tabelu relacione baze podataka:
  - jedan red familije kolona predstavlja kolekciju podataka
  - jedna kolona odgovara jednom podatku, a ne atributu
  - redovi nemaju ujednačen format – broj kolona i nazivi kolona nisu ujednačeni
- Ovaj primer više liči na indeks

Primer: svi tvitovi sa nekim URI-jem

- Jedan red čine podaci o porukama jednog korisnika
  - Jedna superkolona se odnosi na jednu veb lokaciju. Naziv označava veb lokaciju na koju se odnosi poruka. Vrednost je kolekcija kolona koje identifikuju poruke. Nazivi kolona su ključevi poruka a vrednosti su prazne.



- Ovaj primer još manje liči na tabelu relacione baze podataka:
  - jedan red familije kolona predstavlja kolekciju složenih podataka- superkolona

  - jedna superkolona predstavlja kolekciju kolona, pa čak ni struktura superkolona nije ujednačena
  - kao i u prethodnom primeru, redovi nemaju ujednačen format

Particionisanje

- Svaka familija kolona je horizontalno particionisana
- Particionisanje se vrši seckanjem (heširanje) po ključu, na odgovarajući broj particija

12.3 CQL

U prvim verzijama podaci su se koristili sa semantikom familija kolona. Nove verzije imaju unapređen upitni jezik CQL 3 (Cassandra Query Language). Ovaj jezik namerno i imenom i sintaksom liči na SQL. Terminologija je približena terminologiji relacionih baza podataka. Olakšano modeliranje i učenje. Sada je malo prikrivena semantika.

- prostor ključeva = baza podataka
- tabela = familija kolona
- indeks = familija kolona koja sadrži kolekciju ključeva neke tabele i služi za brzo pretraživanje

Pravljenje prostora ključeva

```
CREATE KEYSPACE Test1
WITH REPLICATION = {
  'class' : 'SimpleStrategy',
  'replication_factor' : 3
};
```

- klasa strategije određuje način repliciranja
- faktor replikacije određuje koliko replika će imati svaka particija

Strategije replikacije

- SimpleStrategy
  - podrazumevana strategija
  - uniformna u odnosu na čvorove
  - parametrom replikacije se određuje na koliko čvorova želimo da se ponovi svaka replika
  - dobra za osnovne primene
- NetworkTopologyStrategy
  - naprednija strategija

  - omogućava optimizaciju u odnosu na centre podataka
  - parametrom replikacije se određuje na koliko čvorova u svakom centru podataka želimo da se ponovi svaka replika
  - bolja za primenu u praksi

Brisanje prostora ključeva

```
DROP KEYSPACE Test1
```

Promena prostora ključeva

```
ALTER KEYSPACE Test1
WITH REPLICATION = {
  'class': 'SimpleStrategy',
  'replication_factor':  3
};
```

Upotreba prostora ključeva

Ekvivalent povezivanju sa bazom podataka

```
USE Test1
```

Pregledanje prostora ključeva

Spisak svih prostora ključeva

```
DECRIBE KEYSPACES
```

Opis prostora ključeva (spisak i struktura tabela)

```
DECRIBE KEYSPACE [<naziv>]
```

Ako se ne navede naziv, opisuje se prostor ključeva koji se trenutno koristi

Tipovi podataka

- Tekstualni
- Numerički
- Logički
- Kolekcije
- Univerzalni jedinstveni identifikatori
- Ostalo

Tekstualni

- ascii – 8 - bitne niske
- inet – IP adresa u formatu IPv4 ili IPv6
- text – niska u formatu UTF8
- timestamp – formatirana niska sa datumom i vremenom
- varchar – niska u formatu UTF8

Numerički

- bigint – 64-bitni označeni broj
- counter – 64-bitni broj posebne namene
- decimal – decimalni tip promenljive tačnosti
- double – 64-bitni pokretni zarez IEEE-754
- float – 32-bitni pokretni zarez IEEE-754
- int – 32-bitni označeni broj
- varint – ceo broj proizvoljne preciznosti/dužine (Java)

Logički

- boolean – logička vrednost, true ili false

Kolekcije podataka

- list – uređena lista sa jednim ili više elemenata

  ```
  list<int>, list<text>
  ```

- map – JSON katalog u obliku: {literal: literal, literal: literal ...}

  ```
  map<uuid,int>, map<int,text>
  ```

  odgovara listi kolona

- set – neuređen skup sa jednim ili više elemenata

  ```
  set <int>, set <text>
  ```

- tuple – torka od 2 ili 3 podatka

  ```
  tuple<int,text,float>
  ```

- nije dopušteno da element kolekcije bude kolekcija, poput:

  ```
  list<list< ... >>
  ```

Ostalo

- blob – proizvoljan niz bajtova

Univerzalni jedinstveni identifikatori

- uuid – jedinstveni identifikator u standardnom formatu UUID

Pravljenje tabele

Namerno je slično kao u SQL- u

```
CREATE TABLE student (
  indeks text PRIMARY KEY,
  ime text,
  prezime text,
  smer text);

CREATE TABLE student (
  indeks text,
  ime text,
  prezime text,
  smer text,
  PRIMARY KEY(indeks)
);
```

Dodavanje podataka

```
NSERT INTO student (indeks, ime, prezime, smer)
VALUES ('10442014', 'Petar', 'Petrovic', 'R1');

INSERT INTO student (indeks, ime, prezime, smer)
VALUES ('10432014', 'Ivan', 'Markovic ', 'M1') ;

INSERT INTO student (indeks, ime, prezime, smer)
VALUES ('10452014', 'Tijana', 'Zivkovic', 'R1');
```

Menjanje podataka

```
UPDATE student
SET smer = 'V1'
WHERE indeks = '10432014'
```

Brisanje podataka

Briše izabrane redove:

```
DELETE FROM student
WHERE indeks = '10432014' ;
```

Briše izabrane kolone izabranih redova:

```
DELETE ime
FROM student
WHERE indeks = '10432014';
```

Briše sve redove:

```
TRUNCATE student;
```

Čitanje podataka

```
SELECT * FROM student
WHERE indeks='10452014';

SELECT * FROM student
WHERE ime='Tijana';
```

Zavisno od verzije i konfiguracije, pretraživanje po kolonama za koje ne postoji indeks nije dozvoljeno.

Pravljenje i brisanje indeksa

```
REATE INDEX student_ime ON student (ime);
CREATE INDEX student_prezime ON student (prezime);
CREATE INDEX student_smer ON student (smer);
DROP INDEX ...
```

Upotreba kolekcija

```
ALTER TABLE student ADD napomene list<text>;
```

Postavljanje cele liste:

```
UPDATE student SET napomene = ['aaa', 'bbb']
WHERE indeks = '10432014';
```

Dodavanje elemenata na početak i kraj je brza operacija, ne čita listu:

```
UPDATE student SET napomene = ['xxx'] + napomene
WHERE indeks = '10432014';

UPDATE student SET napomene = napomene + ['yyy']
WHERE indeks = '10432014';
```

Menjanje elemenata je sporije, zato što čita listu:

```
UPDATE student SET napomene [2] = '222'
WHERE indeks = '10432014';
```

Brisanje elemenata takođe čita listu:

```
DELETE napomene[2]
FROM student WHERE indeks = '10432014';
UPDATE student SET napomene = napomene - ['aaa']
WHERE indeks = '10432014';
```

Za vežbu ..

- Instalirati i isprobati SUBP Cassandra (zvanična veb adresa projekta Cassandra: http://cassandra.apache.org/)
- Razmotriti mogućnosti CQL (zvanična dokumentacija za CQL: http://www.datastax.com/documentation/cql/3.1)
  - prikrivena struktura indeksa
  - sličnosti i razlike u odnosu na SQL

Literatura

[1] A.A. Helal, A.A. Heddaya, B.B. Bhargava, Replication Techniques in Distributed Systems, Kluwer Academics Publishers, 1996.

[2] Codd, A relational model of data for large shared data banks, Comm.ACM, 13(6), 1970.

[3] Codd, Extending the database relational model to capture more meaning, ACM ToDS, 4(4), 1979.

[4] Codd, The Relational Model for Database Management – Version 2, Addison Wesley Publ. Inc., 1990.

[5] Darwen, Date, The Third Manifesto, 1995.

[6] db4u, how to design, think about, and use databases, http://db4u.wikidot.com/start

[7] IBM, Database Administration Concepts and Configuration Reference, 2012.

[8] N. Mitić, Uvod u relacione baze podataka, Slajdovi sa predavanja iz kursa Uvod u relacione baze podataka, na Matematičkom fakultetu Univerziteta u Beogradu.

[9] Ozsu, M. Tamer, and Patrick Valduriez. Principles of distributed database systems. Springer Science & Business Media, 2011.

[10] G.Pavlović-Lažetić: Uvod u relacione baze podataka, drugo izdanje, Matematički fakultet, 1999: poglavlja 8-13. http://poincare.matf.bg.ac.rs/~gordana//FINALE.pdf

[11] Nataša V. Pucanović. Principi i metode inteligentnog poslovanja – Primena u Narodnoj banci Srbije. Magistarski rad. Matematički faklutet Univerziteta u Beogradu.

[12] Peter Pin - Shan Chen, The Entity - Relationship Model: Toward a Unified View of Data, ACM Transactions on DB, 1976

[13] Ramakrishnan, Raghu, and Johannes Gehrke. Database management systems. McGraw-Hill, 2000.

[14] Silberschatz, Korth, Sudarshan, Database System Concepts, 6th ed. 2010.

[15] Teorey, et.al, Database Design, know it all, 2008.

[16] Lynch, Gilbert, Brewer's conjecture and the feasibility of consistent, available, partition tolerant web services, ACM SIGACT News, Vol.33/2 (2002)

[17] Eric Brewer, Towards Robust Distributed Systems, PODC (2000)

[18] Dan Pritchett, BASE: An Acid Alternative, ACM Queue, 2008/06

[19] http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[20] http://danweinreb.org/blog/what-does-the-proof-of-the-cap-theorem-mean

[21] http://maniagnosis.crsr.net/2010/09/some-misconceptions-about-cap-theorem.html