1. Koje su to očekivane karakteristike savremenih baza podataka?

- Predefinisane operacije za upravljanje podacima i strukturama podataka (ne želimo da njima upravljamo na najnižem nivou – potrebno je da njihovo upravljanje bude pruženo kroz neke operacije višeg nivoa)
- Postojanje deklarativnog jezika za postavljanje upita nad podacima (opisujemo šta od podataka želimo, ne kako do njih da dođemo)
- Implicitne pomoćne strukture podataka (indeksi, katalozi, ...), implicitni višeprocesni rad, potencijalno i distribuirani rad
- Mogućnost udaljenog pristupa podacima (pored toga, ne bi trebalo da nas interesuje na kom konkretnom računaru se nalaze podaci)
- Integritet podataka (automatsko očuvanje integriteta)
- Pouzdanost
- Sigurnost
- Transakcije
- Programski interfejsi

2. Koji su najvažniji ciljevi upravljanja podacima?

- Raspoloživost podataka – razumna cena pristupu podataka, na jednostavan način, čak i kada je u pitanju udaljen pristup. Podaci trebaju biti pruženi u nekom smislenom formatu, a potrebno je da postoje i aplikativni interfejsi za pristup tim podacima. Važno je napomenuti da je raspoloživost podataka najčešće najpotrebnija upravo kada je opterećenost baze najveća.
- Integritet podataka – integritet predstavlja aspekte konzistentnosti koje je moguće automatski očuvati, atomičnost transakcija i višekorisnički rad (izolovanost transakcija – nju često žrtvujemo do nekog nivoa radi boljih performansi).
- Sigurnost i zaštita podataka – podacima treba da ima pristup samo onaj ko je za to autorizovan, i niko drugi.
- Održavanje sistema – pružanje alata za upravljanje i automatizaciju upravljanja, kao i obezbeđenje od kvarova (rezervne kopije, dnevnici transakcija, ...).
- Nezavisnost od aplikacija – fizička struktura i logička struktura baze treba da budu što skriveniji (oni koji pišu aplikacije ne bi trebalo da do detalja znaju kako je baza implementirana).
- Visoke performanse.
- Skalabilnost (performanse ne smeju da naglo opadnu povećanjem broja korisnika – distribuirane baze podataka i paralelizacija).
- Proširivost (moguće dodavanje novih osobina i kolekcija podataka bazi podataka bez većih problema).

Ove osobine se najviše odnose na relacione baze podataka. U nekim situacijama, potrebno je projektovati bazu za neke specijalizovane namene radi, na primer, povećanja performansi. U ovom slučaju, najčešće je potrebno žrtvovati neke druge osobine, što se često radi prelaskom sa relacionih na neke druge baze podataka.

3. Navesti najvažnije modele podataka kroz istoriju računarstva do danas.

- Mrežni model
- Hijerarhijski model
- Relacioni model
- Model entiteta i odnosa
- Prošireni relacioni model (ovo je ono što je danas u upotrebi kada mislimo na relacione baze podataka)
- Objektni model
- Objektno relacioni model

4. Objasniti osnovne koncepte mrežnog modela podataka.

Podaci su se čuvali u "strukturama podataka". Ove strukture su sadržale polja koja bi odgovarala atributima nekog entiteta. Jedna konkretna instanca date strukture bi odgovarala instanci konkretnog entiteta (osoba [ime, prezime, grad], račun [broj računa, stanje], ...).

Ovi podaci, tj. strukture, bi se dalje povezivali vezama – analogon pokazivačima u programskim jezicima. Dakle, mrežni model podataka dosta podseća na dijagramsko povezivanje podataka.

Radi pristupa podacima, korisnik bi imao ulogu "navigatora" kroz ove pokazivače, što dovodi do problema u slučaju prekida veza između struktura usled promena u bazi podataka.

5. Objasniti osnovne slabosti mrežnog modela podataka.

Osnovni problem mrežnog modela je taj što je porastom složenosti baze naglo rastao i broj različitih puteva između struktura, pa bi, pošto je korisnik baze sam imao zaduženje da "naviguje" tim vezama, dolazilo do problema izbora dobrog puta. Takođe, usled promena struktura podataka, dolazilo bi do usložnjavanja puteva (i time cene pristupa), pa čak i prekida putevima veza između podataka.

Radi smanjenja cene navigacije između podataka povezanih složenim putevima, često su se uvodile i veze "prečice", ali, ovo je ništa drugo do dodatne redundantnosti u vezama između podataka.

6. Objasniti osnovne koncepte hijerarhijskog modela podataka.

Osnovni koncept i ideja iza hijerarhijskog modela je da se podaci (tj. slogovi) organizuju hijerarhijski (u stablo), gde su čvorovi povezani jedan-više vezama (jedan roditelj može imati više dece). Koren ove hijerarhije predstavlja tzv. "prazan čvor" od kojeg kreće svaka pretraga. Važi da od korena do bilo kog drugog čvora u hijerarhiji postoji tačno jedan put – dakle, redundantnost veza (koja je stvarala problem u mrežnom modelu) je eliminisana, pa je, usled jedinstvenih puteva, omogućeno automatsko pronalaženje puta između podataka. Međutim, jedinstveni putevi dovode do novog problema, a to je redundantnost podataka.

Radi povećanja efikasnosti u pristupu, često se uvode pomoćne hijerarhije (slična uloga indeksima u relacionim bazama podataka), što je, opet, uvođenje redundantnosti podataka (slično kao što je kod mrežnog modela uvođenje veza prečica dodavanje na redundantnost veza).

Neke savremene nerelacione baze podataka počivaju na sličnim principima kao hijerarhijske baze podataka (npr. Apache Cassandra).

7. Objasniti osnovne slabosti hijerarhijskog modela podataka.

Osnovna slabost hijerarhijskog modela je redundantnost podataka. Naime, zbog ograničenja jedinstvenih puteva od korena do svakog čvora, da bi se izveli složeniji odnosi između podataka, potrebno je uvesti redundantnost, bilo u vidu čuvanja više hijerarhija, ili više kopija istog podatka unutar jedne hijerarhije.

Na primer, zamislimo situaciju da je potrebno pratiti studijske programe, studente na studijskim programima i predmete koje su ti studenti upisali. Ukoliko koreni čvor (zanemarujući "prazan čvor") predstavlja studijski program, pod pretpostavkom da jedan student može biti upisan na samo jedan program, nećemo imati problema pri povezivanju studenata sa studijskim programom. Međutim, do problema dolazi pri dodavanju predmeta – zbog jedinstvenosti veza, za svaki upisani predmet bi imali zaseban slog u hijerarhiji, čak iako je to jedan te isti predmet (ukoliko 100 studenata upiše projektovanje baza podataka, imali bi 100 slogova predmeta projektovanje baza podataka).

8. Objasniti osnovne nivoe apstrakcije kod savremenih baza podataka.

- Spoljašnji nivo (nivo korisnika) – šta (koje i kakve podatke) korisnik vidi – ovi podaci mogu biti transformisani, sakriveni i sl. u zavisnosti od konkretnog korisnika.
- Konceptualni (logički) nivo – sve što čini logički model podataka (podaci i odnosi među njima), potrebno je da podržava ceo spoljašnji nivo.
- Nivo fizičkih podataka – sama fizička organizacija podataka u softverskom sistemu.
- Nivo fizičkih uređaja – sama fizička organizacija podataka na fizičkim uređajima.

9. Iz kojih uglova se obično posmatra arhitektura baze podataka?

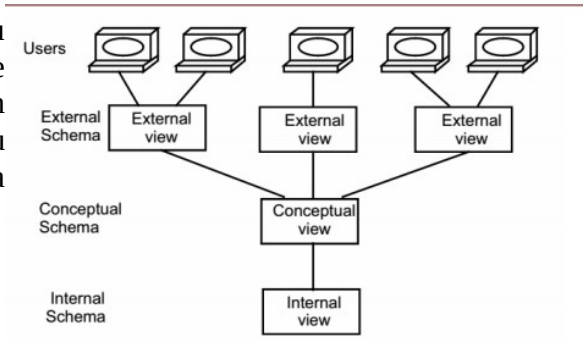
- Iz ugla komponenti – funkcionalne celine SUBP-a i njihove interakcije kojima se ostvaruje puna funkcionalnost sistema. Ovaj ugao posmatranja je važniji za implementaciju SUBP-a nego za neku konkretnu bazu podataka (BP).
- Iz ugla funkcija – prepoznaju se različite klase korisnika i funkcije koje sistem obavlja za njih (npr. sistem se može ponašati drugačije, tj. pružati drugačije podatke, u zavisnosti od toga da li ga koristi profesor ili student). Prednost ovakvog ugla posmatranja je jasnoća funkcija i ciljeva sistema, ali mana je mali uvid u to kako se te funkcije i ostvaruju.
- Iz ugla podataka – prepoznaju se različite vrste podataka i kako različite funkcionalne jedinice koriste te podatke na različite načine. Prednost je u isticanju podataka kao centralne i glavne jedinice u sistemu, ali je mana u nemogućnosti potpunog opisa arhitekture ukoliko nisu opisane funkcionalne celine.

U praksi, potrebno je koristiti sva tri pristupa da bi se dobio dovoljno dobar model arhitekture (konceptualne definicije strukture sistema – komponente, njihove funkcije i njihove interakcije i odnosi).

10. Objasniti "standardizovanu" arhitekturu ANSI/SPARC.

Iako ova arhitektura nije zaista formalno usvojena, u praksi je široko prihvaćena i primenjena. Načelno je kreirana iz ugla podataka, ali zaista čini celokupan pogled na arhitekturu baze podataka. Ideja je da se u ovoj arhitekturi prepoznaju tri nivoa, tj. pogleda (sheme) na podatke:

- Spoljašnja shema
- Konceptualna shema
- Interna shema



11. Objasniti osnovne nivoe podataka (poglede na podatke) u arhitekturi ANSI/SPARC.

- Spoljašnji nivo (pogled, shema) – predstavlja najviši nivo apstrakcije podataka i pogled na njih iz ugla korisnika. Ovakvih pogleda može biti više, jer različiti korisnici mogu imati različite potrebe.
- Konceptualni nivo (pogled, shema) – manji nivo apstrakcije od spoljašnjeg nivoa, ali opet veći od internog (ne uzima u obzir fizičku implementaciju). Ovaj nivo opisuje kako su zaista (na logičkom nivou) implementirani podaci i njihovi odnosi, pa je prilagođen konkretnom modelu podataka (npr. relacionom). Potrebno je da usaglašava i podržava sve poglede spoljašnjeg nivoa.
- Interni nivo (pogled, shema) – najniži nivo apstrakcije, koji opisuje elemente fizičke implementacije (pa time i kako su podaci implementirani u SUBP i na konkretnoj softverskoj i hardverskoj platformi).

12. Objasniti spoljašnju shemu arhitekture ANSI/SPARC.

Predstavlja najviši nivo apstrakcije podataka i pogled na njih iz ugla korisnika. Ovakvih pogleda može biti više, jer različiti korisnici mogu imati različite potrebe. Ovaj nivo zapravo opisuje kako korisnici, kao i programeri koji rade na nekoj konkretnoj aplikaciji koja koristi bazu, vide podatke. Podaci koje oni vide mogu biti sakriveni ili transformisani u odnosu na one koji se mogu videti iz konceptualne sheme.

13. Objasniti konceptualnu shemu arhitekture ANSI/SPARC.

Predstavlja manji nivo apstrakcije od spoljašnjeg nivoa, ali opet veći od internog (ne uzima u obzir fizičku implementaciju). Ovaj nivo opisuje kako su zaista (na logičkom nivou) implementirani podaci i njihovi odnosi, pa je prilagođen konkretnom modelu podataka (npr. relacionom). Potrebno je da usaglašava i podržava sve poglede spoljašnjeg nivoa.

14. Objasniti internu shemu arhitekture ANSI/SPARC.

Predstavlja najniži nivo apstrakcije koji opisuje elemente fizičke implementacije (pa time i kako su podaci implementirani u SUBP i na konkretnoj softverskoj i hardverskoj platformi) – da li se podaci čuvaju distribuirano, gde se konkretno nalaze na disku, da li su možda u baferu u radnoj memoriji, ...

15. Kakav je odnos relacionih baza podataka i standardizovane arhitekture ANSI/SPARC?

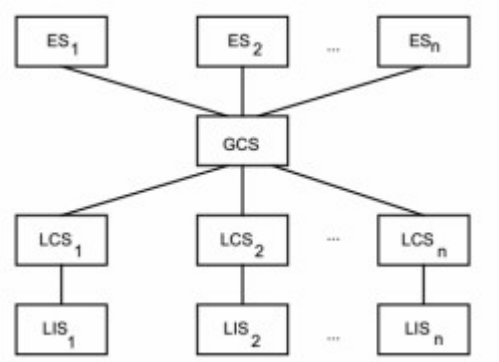
Nivoi ANSI/SPARC arhitekture se okvirno mogu predstaviti i u relacionom modelu – spoljašnji nivo (tj. spoljašnje sheme) bi predstavljali svi kreirani pogledi u okviru baze podataka, konceptualni nivo bi bila sama shema baze podataka (opisi tabela, atributa, primarnih i stranih ključeva, ostalih ograničenja, ...), a interni nivo bi činili fizički aspekti baze – indeksi, prostori tabela (db2 SYSCATSPACE, USERSPACE1, TEMPSPACE1, ...), optimizacije i slično.

16. Objasniti primer arhitekture klijent-server.

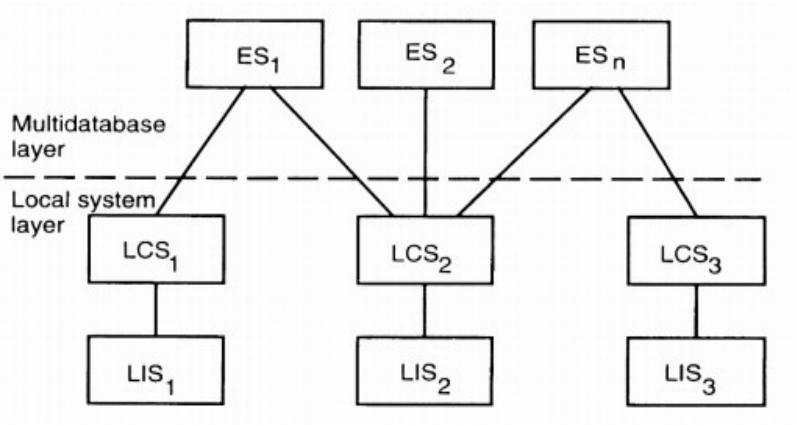
Ideja ove arhitekture je da se razdvoje funkcionalnosti servera (serverskog dela SUBP-a), koji pruža usluge klijentima, i klijenta (klijentskog dela SUBP-a). Server zapravo vrši svo upravljanje podacima – obradu upita, optimizaciju, upravljanje transakcijama i slično, dok je klijent samo zadužen za uspostavljanje komunikacije između konkretne aplikacije i servera (poput nekog međusloja) i upravljanje podacima koji su keširani na strani klijenta – upravljanje katancima i provera konzistentnosti transakcija.

17. Objasniti koncept distribuiranih arhitektura na primeru arhitekture ravnopravnih čvorova.

U slučaju distribuiranih arhitektura, imamo više servera koji imaju različite ili deljene uloge. Jedan primer ovakve arhitekture je arhitektura ravnopravnih čvorova – svaki server, tj. čvor, sadrži neki deo podataka. On ima svoju lokalnu internu shemu, kao i lokalnu konceptualnu shemu (koja je potrebna zbog fragmentacije i replikacije podataka). Ove lokalne konceptualne sheme se zatim kombinuju u globalnu konceptualnu shemu (ili eksplicitno, u kom slučaju se takva shema čuva na nekom posebnom, centralnom čvoru, ili implicitno – gde se ona implicitno gradi iz svih lokalnih konceptualnih shema). Korisnici onda imaju svoje spoljašnje sheme koje su povezane sa globalnom konceptualnom shemom.



Još jedna distribuirana arhitektura je tzv. federativna arhitektura u kojoj se ne koristi globalna konceptualna shema, već svaki čvor pruža svoju tzv. izvoznu shemu – unija ovih izvoznih shema praktično predstavlja globalnu bazu podataka. Važno je napomenuti da u ovom slučaju, zasebni čvorovi najšešće predstavljaju kompletne baze podataka koje imaju svoju lokalnu ulogu i koje se udružuju. Primer neke federativne arhitekture bio bi udruživanje više baza različitih fakulteta (tj. njihovih delova definisanih izvoznih shemama) u jednu federativnu bazu univerziteta.



18. Objasniti odnos vrste modela (i modeliranja) podataka i nivoa podataka prema arhitekturi ANSI/SPARC.

Baze podataka se projektuju u nivoima, ali ovi nivoi ne prate u potpunosti i nivoe arhitekture (u ANSI/SPARC arhitekturi to su spoljašnji, konceptualni i interni nivo). Naime, granice nivoa projektovanja i nivoa arhitekture se neće tačno poklapati. Slede nivoi projektovanja:

- Konceptualno projektovanje (modeliranje) – prvi korak projektovanja, u ANSI/SPARC arhitekturi odgovara planiranju spoljašnje sheme i dela konceptualne sheme (spoljašnje sheme se kombinuju radi dobijanja apstraktnog *konceptualnog modela* podataka, ali on još uvek nije prilagođen nekom konkretnom implementacionom modelu podataka, poput relacionom).
- Logičko projektovanje (modeliranje) – drugi korak, većim delom odgovara pravljenju konceptualne sheme – konceptualni model podataka dobijen iz prethodne faze se prilagođava konkretnom modelu podataka (npr. relacionom) i time se dobija *logički model*.
- Fizičko projektovanje (modeliranje) – odnosi se na internu shemu u ANSI/SPARC arhitekturi i na neke aspekte konceptualne sheme. Od logičkog modela dobijenog iz prethodne faze pravi se *fizički (implementacioni) model*.
- Projektovanje bezbednosti – ova faza je uglavnom ortogonalna na prethodne, pa se u nekoj meri može preplitati sa njima (mada se u najvećoj meri odvija u toku i nakon faze fizičkog projektovanja).

19. Navesti i ukratko objasniti osnovne faze (celine) pri projektovanju baza podataka.

Konceptualno projektovanje – prva faza projektovanja baze podataka. U suštini, ova faza podrazumeva pravljenje projekta koji što bolje opisuje sam domen, tj. elemente iz stvarnog sveta i njihove odnose koje je potrebno modelirati u bazi podataka. Rezultat ove faze je konceptualni model.

Logičko projektovanje – u ovoj fazi se konceptualni model (dobijen iz prethodne faze) prilagođava konkretnom modelu podataka koji će se koristiti (relacioni, neki nerelacioni – npr. objektni, ...). Rezultat ove faze je logički model.

Fizičko projektovanje – u obzir se uzimaju sve specifičnosti koje mogu uticati na efikasnost implementacije, pa se u ovoj fazi vrši optimizacija strukture baze podataka i slično. Rezultat ove faze je fizički, tj. implementacioni model.

Projektovanje bezbednosti – ova faza se delom vrši paralelno sa ostale tri, ali je najveći deo najčešće nakon faze fizičkog projektovanja.

20. Navesti i objasniti osnovne korake po fazama projektovanja baze podataka.

- Konceptualno projektovanje
  - Analiza zahteva – prikupljanje i analiza informacija o domenu.
  - Projektovanje pogleda (konceptualno projektovanje pojedinačnih spoljašnjih shema) – modeliranje apstraktnog modela podataka svake spoljašnje sheme posebno.
  - Objedinjavanje pogleda (pravljenje objedinjenog konceptualnog modela) – objedinjavanje apstraktnih modela podataka dobijenih prethodnim korakom u objedinjen model.
  - Grupisanje entiteta – na neki način inverzan korak od prethodnog, vrši se grupisanje radi veće razumljivosti dobijenog modela.
- Logičko projektovanje
  - Prevođenje konceptualnog modela u logički model – konceptualni model dobijen prethodnom fazom se, u skladu sa izabranim modelom podataka (npr. relacionim), prevodi u logički model.
  - Prečišćavanje sheme – dodatno, potpuno prilagođavanje logičkog modela jeziku i pravilima konkretnog modela podataka i rešavanje potencijalnih problema (npr. eliminacija redundantnosti kroz proces normalizacije u slučaju relacionog modela).
- Fizičko projektovanje
  - Fizičko projektovanje implementacije – optimizacija logičkog modela iz prethodne faze prema SUBP-u koji će se koristiti i prema očekivanom načinu upotrebe (indeksi, baferi, distribuiranje baze, ...).
  - Fizičko projektovanje spoljašnjih shema – projektovanje implementacije spoljašnjih shema (koje su konceptualno projektovane već u prvim koracima).
- Projektovanje bezbednosti
  - Projektovanje bezbednosti – ovaj korak se delom odvija paralelno sa ostalim koracima, mada se većim delom vrši na kraju (podrazumeva, na primer, određivanje grupa korisnika, dodeljivanje pristupa tim grupama, određivanje dozvoljenih operacija nad određenim delovima baze i slično).

21. Navesti i objasniti korake pri pravljenju konceptualnog modela BP.

- Analiza zahteva – prikupljanje i analiza informacija o domenu.
- Projektovanje pogleda (konceptualno projektovanje pojedinačnih spoljašnjih shema) – modeliranje apstraktnog modela podataka svake spoljašnje sheme posebno.
- Objedinjavanje pogleda (pravljenje objedinjenog konceptualnog modela) – objedinjavanje apstraktnih modela podataka dobijenih prethodnim korakom u objedinjen model.
- Grupisanje entiteta – na neki način inverzan korak od prethodnog, vrši se grupisanje radi veće razumljivosti dobijenog modela.

22. Objasniti korak *Analiza zahteva* pri projektovanju BP. Cilj? Sadržaj?

Glavni zadatak ovog koraka je prikupljanje i analiza informacija o domenu. Naime, cilj je, kroz prikupljene informacije, identifikacija glavnih celina domena u kontekstu trajnih podataka koji će se čuvati u bazi podataka, razumevanje tih podataka, njihove strukture i njihovih međusobnih odnosa. Dodatno, potrebno je, na nekom višem nivou, definisati i dodatne zahteve u vidu integriteta podataka, potrebnih performansi, bezbednosti i slično (implementacija ovih dodatnih zahteva se, naravno, vrši u kasnijim fazama).

Pored ovoga, potrebno je sakupiti i informacije o uslovima implementacije – konačna implementacija će, vrlo verovatno, morati da koristi neke određene tehnologije, da radi na nekom već određenom hardveru i da zadovoljava neke potrebe korisnika i aplikacija u vidu određenih aplikativnih i korisničkih interfejsa. Dodatno, potrebno je izraditi iscrpnu dokumentaciju za sve ovo.

Možda i najvažniji cilj ovog koraka je prepoznavanje i dekompozicija sistema na skup slabije povezanih celina (koji će se kasnije, tj. u fazi objedinjavanja, ponovo spojiti u celinu). Ovo se najčešće svodi na uočavanje spoljašnjih shema (u suštini vrsta korisnika) i njihovo dalje dekomponovanje po potrebi. Na primer, za bazu podataka fakulteta je, između ostalog, potrebna shema za studente, za nastavnike, za studentsku službu, ... Ukoliko je potrebno, ove sheme se razbijaju na manje, funkcionalne celine uočavanjem različitih poslova unutar jedne spoljašnje sheme, na primer prijavljivanje ispita ili upis godine (ili čak i potpuno izdvojenih poslova, npr. organizacija upisnih rokova).

23. Navesti i objasniti ciljeve koraka *Analiza zahteva* pri projektovanju BP. Cilj? Sadržaj?

Isto kao 22.

24. Objasniti ukratko korak *Konceptualno modeliranje pojedinačne sheme* pri projektovanju BP. Cilj? Sadržaj?

Ovaj korak se zasebno izvršava za svaku pojedinačnu shemu prepoznatu prilikom analize zahteva. Cilj je pravljenje apstraktnog modela podataka koji, iako apstraktan (izbegavamo neke implementacione detalje koji nisu potrebni za razumevanje semantike), potpuno opisuje semantiku podataka i odnosa između njih unutar date sheme. Ovakav apstraktan model podrazumeva klasifikaciju skupova podataka unutar sheme, tj. određivanje entiteta (tj. klasa u kontekstu UML-a), određivanje strukture podataka u vidu atributa tih entiteta i prepoznavanje odnosa između entiteta.

25. Objasniti ukratko korak *Objedinjavanje pogleda* pri projektovanju BP. Cilj? Sadržaj?

Korak koji prethodi ovom, tj. korak konceptualnog modeliranja pojedinačnih shema (projektovanje pojedinačnih pogleda), kao rezultat ima pojedinačne poglede koje je u ovom koraku potrebno objediniti u jedan, pa možemo reći da je cilj ovog koraka pravljenje objedinjenog apstraktnog modela podataka. Ovaj proces objedinjavanja je iterativan – određeni konflikti su mogući (i verovatni), i oni se razrešavaju iterativnim postupkom.

26. Objasniti ukratko korak *Grupisanje entiteta* pri projektovanju BP. Cilj? Sadržaj?

Pošto korak objedinjavanja pogleda može rezultovati shemom sa čak i više stotina entiteta, potrebno je nekako uprostiti dobijeni dijagram. Korak grupisanja entiteta kao zadatak ima pravljenje većeg broja manjih i preglednijih dijagrama modela, postupkom grupisanja, od polaznog, kompleksnog dijagrama. Iako ovaj korak nije suštinski neophodan (ne pravi se novi model, već se njegov prikaz uprošćava), često je od velikog značaja za razumevanje složenijih modela dobijenih iz prethodnih koraka.

27. Objasniti ukratko korak *Prevođenje konceptualnog u logički model* pri projektovanju BP. Cilj? Sadržaj?

Elementi dobijenog konceptualnog modela se prevode u elemente logičkog modela, u skladu sa izabranim modelom podataka (npr. relacionim).

28. Objasniti ukratko korak *Prečišćavanje sheme* pri projektovanju BP. Cilj? Sadržaj?

Korak prečišćavanja sheme (logičkog modela) dodatno (do doslednosti) prilagođava model korišćenom modelu podataka (npr. relacionom). Uz to, prepoznaju se i potencijalni problemi (koji zavise od modela podataka) i rešavaju se (u slučaju relacionog modela, ovo se najčešće svodi na eliminaciju redundantnosti, tj. dovođenje modela u određenu normalnu formu).

29. Objasniti ukratko korak *Fizičko projektovanje implementacije* pri projektovanju BP. Cilj? Sadržaj?

Ciljevi ovog koraka su optimizacija logičkog modela dobijenog iz prethodnog koraka prema specifičnostima SUBP-a koji će se koristiti, kao i prema očekivanom načinu upotrebe (možda su, na primer, neke tabele nepromenljive, pa je jednostavno uvesti dodatnu redundantnost radi bržeg pristupa). Dodatno, određuje se tačna interna shema baze podataka, što uključuje parametre implementacije tabela baze, indekse, bafere, projektovanje distrubiranja ukoliko je potrebno da baza bude distribuirana i slično. Takođe, u ovoj fazi se određuju i mehanizmi preslikavanja interne (fizičke) u konceptualnu shemu baze podataka (kako se strukturiraju podaci koji su sačuvani na disku).

30. Objasniti ukratko korak *Fizičko projektovanje spoljašnjih shema* pri projektovanju BP. Cilj? Sadržaj?

Glavni ciljevi ovog koraka su projektovanje implementacije spoljašnjih shema (koje su inicijalno projektovane na konceptualnom nivou), kao i aplikativnih interfejsa za pristup podacima. Interna shema se dodatno prilagođava specifičnostima spoljašnjih alata koji će se koristiti.

31. Objasniti ukratko korak *Projektovanje bezbednosti* pri projektovanju BP. Cilj? Sadržaj?

Za razliku od ostalih koraka projektovanja baze podataka, ovaj korak se delom odvija paralelno sa ostalima (mada se većim delom ipak odvija tek po završetku ostalih koraka). Ovde se prepoznaju vrste i uloge korisnika baze podataka, kao i vrste aplikacija koje koriste bazu podataka, određuju se korisničke grupe i skupovi privilegija nad bazom unutar te grupe, kao i za aplikacije. Dodatno se prepoznaju posebno osetljivi delovi baze kojima je potrebno dodatno redukovati pristup.

32. Objasniti odnos faza i koraka projektovanja i arhitekture baze podataka.

Faza konceptualnog projektovanja najvećim delom odgovara apstrakciji spoljašnje sheme i delu konceptualne sheme, faza logičkog projektovanja uglavnom odgovara konceptualnoj shemi, dok faza fizičkog projektovanja odgovara internoj shemi, ali i implementaciji spoljašnje sheme, te se zbog ove faze kroz nivoe arhitekture zapravo prolazi dva puta.

Концептуално пројектовање

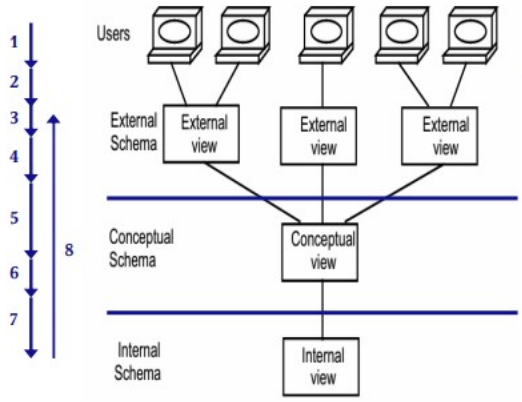
1. Анализа захтева
2. Концептуално пројектовање погледа
3. Обједињавање концептуалног модела
4. Груписање ентитета

Логичко пројектовање

5. Превођење концептуалног модела у логички
6. Пречишћавање схеме

Физичко пројектовање

7. Оптимизовање логичког модела и пројектовање имплементације према моделу употребе
8. Имплементација спољашње схеме



33. Objasniti detaljno ciljeve koraka *Analiza zahteva* pri projektovanju BP.

Isto kao 22.

34. Objasniti poslove pri analiziranju zahteva.

Isto kao 22. (ono detaljno opisuje korak analize zahteva – eventualno se vratiti na ova pitanja i uprostiti odgovore)

35. Koje poslove obuhvata konceptualno modeliranje podataka?

Glavni poslovi u okviru konceptualnog modeliranja jedne sheme su klasifikacija skupova podataka (tj. prepoznavanje entiteta (ili klasa u slučaju UML-a) i njihovih atributa) i prepoznavanje odnosa između tako dobijenih entiteta (klasa).

36. Šta obuhvata klasifikacija skupova podataka?

Klasifikacija skupova podataka, kao jedan od poslova u okviru koraka konceptualnog modeliranja podataka, podrazumeva prepoznavanje etniteta (ili klasa u slučaju UML-a), kao i njihovih atributa, u okviru jedne sheme. U suštini, svaka imenica iz opisa sheme predstavlja kandidata za entitet (ili klasu), ili atribut.

37. Kako se ustanovljava da li bi nešto trebalo da bude atribut ili entitet?

Problem izbora između entiteta i atributa nema uvek jasno određeno rešenje. Naime, jedan isti "objekat iz domena" se, zavisno od slučaja, može bolje modelirati kao entitet ili kao atribut. Ipak, postoje određene smernice koje pomažu pri ovom izboru.

Ukoliko nešto ima pridodate opisne informacije (na primer, ispit opisuju student koji ga je polagao, predmet na kojem je taj ispit, osvojen broj bodova, dobijena ocena i slično), onda bi to trebalo da bude entitet.

Ukoliko nešto ima višestruku ili složenu vrednost umesto skalarne, najčešće je bolje da to bude entitet (na primer, student može pripadati većem broju studentskih organizacija, pa je umesto čuvanja liste naziva organizacija za svakog studenta bolje napraviti entitet studentske organizacije). Sa druge strane, ukoliko neki podatak ima samo skalarnu vrednost (koja može biti opisnog karaktera), na primer "ocena", onda je najčešće najbolje da to bude atribut.

Dodatno, ukoliko se za neki podatak očekuje da bude promenljiv, često je najbolje da on bude entitet (na primer, ukoliko bi "grad" bio atribut koji predstavlja naziv grada i koji se čuva za svakog studenta, promenom naziva grada bi, vrlo verovatno, bilo potrebno izmeniti nazive tog grada kod svakog studenta, dok bi, ukoliko je grad entitet sa atributom "naziv" i nekim surogat ključem, bilo dovoljno izmeniti naziv kod samo jedne instance tog entiteta).

38. Kako se ustanovljava kom entitetu pripada koji atribut?

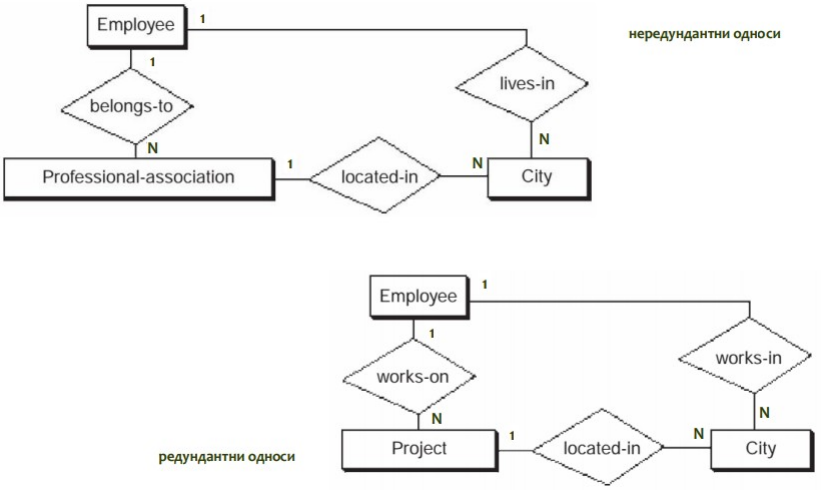
Glavno pravilo pri dodeljivanju atributa je da oni treba da pripadaju onim entitetima koje najneposrednije opisuju. Ovo se često može odrediti posmatranjem funkcionalnih zavisnosti, ali postoje slučajevi kada to nije baš jednostavno (npr. u slučaju tranzitivnih zavisnosti).

Kao primer, možemo posmatrati entitet "Student". Između ostalog, "Student" ima atribut "Ime" i on je vezan za neki konkretan studijski program. Ukoliko on upiše dva studijska programa, postojaće redundantnost u atributu "Ime" (za svaki upisani studijski program čuvaćemo po jedno ime). Ovde se može primetiti skrivena zavisnost Student -> Osoba -> Ime, pa ima smisla izdvojiti novi entitet "Osoba" kojem može pripadati više entiteta "Student" (svaki od njih odgovara jednom upisanom studijskom programu).

39. Koji elementi odnosa moraju da se ustanove pri modeliranju odnosa?

Prilikom modeliranja odnosa, potrebno je ustanoviti učesnike u tom odnosu, kardinalnost tog odnosa, potencijalne dodatne atribute koji opisuju taj odnos i jasan naziv i semantiku odnosa u domenu koji se modelira.

40. Objasniti problem redundantnih odnosa. Kako se uočavaju redundantni odnosi?



Redundantni odnosi predstavljaju odnose koji su već, semantički ekvivalentno u kontekstu domena, predstavljeni (na primer nekim tranzitivnim zavisnostima), pa su time i nepotrebni. Kao rezultat imaju teško otklonjivu redundantnost u fazi implementacije, a dodatno ometaju i konzistentnost i normalizaciju baze, pa je potrebno ukloniti ih. Ipak, redundantnost je ponekad dozvoljena na nivou konceptualnog modela ukoliko ona povećava razumljivost (što može biti slučaj u jako složenim modelima), ali ona mora biti jasno dokumentovana radi uklanjanja u kasnijim fazama, tj. pri izradi logičkog modela.

41. Kako se ustanovljavaju odnosi generalizacije/specijalizacije pri klasifikovanju skupova podataka (konceptualno projektovanje)?

Generalizacija (i specijalizacija) se primenjuje u slučajevima kada je potrebno modelirati posebne slučajeve nekog opštijeg slučaja (npr. opšti slučaj student, specijalni slučajevi student osnovnih, master, doktorskih studija).

Međutim, postavlja se pitanje, u kom smeru treba da ide generalizacija u slučaju projektovanja baze podataka? Naime, u OO paradigmi, smer generalizacije/specijalizacije se uspostavlja na osnovu ponašanja objekata (npr. svaki kvadrat (specijalizacija) je istovremeno i pravougaonik (generalizacija)). Međutim, u bazama podataka ovaj problem je složeniji – često smo u iskušenju da posmatramo strukturu umesto ponašanja, pa bi time, u prethodnom slučaju, kvadrat bio generalizacija (jer ga opisuje samo podatak za dužinu stranice a), dok bi pravougaonik bio specijalizacija (imamo podatak za dužinu stranice a, ali i za stranicu b). Ovakav pristup posmatranja samo strukture može izazvati probleme u dubljim hijerarhijama

Što se tiče atributa i odnosa u hijerarhijama generalizacije/specijalizacije, njih je potrebno popeti što više uz hijerarhiju (što apstraktnije, to bolje).

42. Šta su redundantni odnosi i kako se prepoznaju?

Isto kao 40.

43. Kako se postupa sa odnosima višeg reda (sa više od dva učesnika)?

Odnosi višeg reda su često semantički (a i tehnički, za kasnije korake modeliranja) zahtevniji, pa je veoma poželjno eliminisati ih. Ovo se radi pokušajem razbijanja jednog takvog odnosa na više binarnih odnosa. Međutim, ponekad se ispostavi da je to nemoguće, pa se u tom slučaju ostavlja odnos višeg reda.

Jedan primer razbijanja ovakvog odnosa je sledeći – Nastavnik A drži čas iz predmeta B u učionici C u terminu D. U pitanju je složeni odnos sa četiri učesnika. Međutim, ukoliko uvedemo koncept (entitet) grupe koja povezuje predmet i nastavnika koji taj predmet drži datoj grupi, onda možemo imati, i dalje složeni, ali ipak sa manje učesnika, odnos oblika – Grupa E ima čas u učionici C u terminu D.

44. Objasniti razliku između lokalnih i globalnih shema pri modeliranju BP. Zašto su obično neophodni lokalni pogledi?

Lokalni sheme (lokalni pogledi) modeliraju samo manje, odvojene delove čitavog domena, dok globalna shema predstavlja spojene lokalne sheme u jedan celovit konceptualni model. Pored toga što je modeliranje manjih delova domena mnogo jednostavnije i efikasnije od modeliranja celokupnog domena odjednom, lokalni pogledi su potrebni i zato što različiti korisnici baze podataka imaju različite potrebe, te svakoj vrsti korisnika može odgovarati zasebna lokalna shema. Na primer, možemo imati lokalnu shemu za studente nekog fakulteta i za profesore tog fakulteta.

45. Objasniti ukratko korak *Integrisanje pogleda* pri pravljenju konceptualnog modela BP. Navesti ciljeve i osnovne postupke.

Korak integrisanja pogleda ima svrhu spajanja svih lokalnih pogleda dobijenih iz prethodnog koraka u jednu celovitu, globalnu shemu, čime se dobija objedinjeni konceptualni model baze podataka.

Osnovni koraci u ovom postupku su:
- Poređenje lokalnih shema i prepoznavanje konflikata među njima
- Preuređivanje tih lokalnih shema radi razrešavanja otkrivenih konflikata
- Spajanje i restrukturiranje shema

Ovaj postupak je najčešće iterativan.

46. Koji su osnovni problemi pri integrisanju pogleda?

Osnovni problemi, tj. konflikti koji se pojavljuju pri integrisanju pogleda su:
- Konflikti imena
- Strukturni konflikti
- Konflikti ključeva
- Konflikti zavisnosti

Dodatno, tu je i problem spajanja i restukturiranja sheme na ispravan način nakon razrešavanja konflikata. Shema dobijena spajanjem mora da bude potpuna, minimalna i razumljiva.

47. Navesti i objasniti vrste konflikata koji mogu da nastanu pri integrisanju pogleda.

Konflikti imena – postoje nesuglasice u imenima entiteta i odnosa u lokalnim shemama koje se spajaju (različito ime za isti entitet ili odnos, tj. sinonimi, ili isto ime za različite entitete ili odnose, tj. homonimi).

Strukturni konflikti – različiti strukturni elementi se upotrebljavaju za modeliranje istog koncepta (npr. u nekom pogledu se neki koncept modelira kao entitet, u drugom kao odnos).

Konflikti ključeva – isti entitet u različitim pogledima ima dodeljene različite ključeve.

Konflikti zavisnosti – isti odnosi se u različitim pogledima modeliraju na različite načine (npr. negde se modelira kao asocijacija, negde kao agregacija, možda sa različitim kardinalnostima i sl.).

48. Objasniti detaljno konflikte imena pri integrisanju pogleda.

Konflikti imena predstavljaju nesuglasice u imenima entiteta i odnosa u različitim lokalnim shemama koje je potrebno spojiti. Dve glavne vrste ovakvih nesuglasica su sinonimi, u kom slučaju se različiti nazivi koriste za iste koncepte (entitete ili odnose), i homonimi, u kom slučaju se isti naziv koristi za različite koncepte. Sinonimi su prilično jednostavni za rešavanje (u kom slučaju, najjednostavnije, možemo izabrati jedan od naziva), dok su homonimi malo kompleksniji, pa iz tog razloga imaju prioritet prilikom rešavanja. Primer homonima bi bio da se "Ispit" naziva entitet u jednom pogledu gde on predstavlja položen ispit, a takođe i u drugom pogledu gde on predstavlja prijavu ispita.

49. Objasniti detaljno strukturne konflikte pri integrisanju pogleda.

Strukturni konflikti predstavljaju situaciju kada se različiti strukturni elementi koriste za opisivanje istog koncepta kroz više različitih lokalnih pogleda. Na primer, u jednom pogledu bi nešto moglo biti predstavljeno kao entitet, a u drugom kao odnos (potencijalno kao odnos sa atributima). Konkretan primer bi bila narudžbenica u sistemu gde korisnik naručuje neki proizvod. U jednom pogledu bi mogao postojati entitet "Narudžbenica", a u drugom odnos "Naručuje" sa atributom "ID".

50. Objasniti detaljno konflikte ključeva pri integrisanju pogleda.

Konflikti ključeva nastaju kada isti entiteti imaju različite ključeve u različitim lokalnim pogledima. U toj situaciji je potrebno ujednačiti ključeve, jer u suprotnom može doći do redundantnosti ključeva (zamislimo, na primer, da se koriste dva ili više različita surogat ključa za jedan isti entitet).

51. Objasniti detaljno konflikte zavisnosti pri integrisanju pogleda.

Konflikti zavisnosti nastaju kada se isti odnosi modeliraju na različite načine kroz različite lokalne poglede. Na primer, u različitim lokalnim pogledima jedan isti odnos može biti asocijacija, a u drugom agregracija, ili pak da imaju različite kardinalnosti. Ovi konflikti se razrešavaju ujednačavanjem karakteristika odnosa u konfliktu, a ponekad čak i menjanjem strukture (npr. dodavanjem novih odnosa).

52. Kako se razrešavaju konflikti pri integrisanju pogleda?

U zavisnosti od konkretnih konflikata koji su otkriveni, zavisi i proces razrešavanja. Međutim, moguće je da je, usled velikog broja teško razrešivih konflikata, potrebno izvršiti dodatnu ili ponovljenu analizu zahteva ili veliku modifikaciju lokalnih pogleda. Dodatno, radi lakšeg razrešavanja konflikata usled boljeg razumevanja lokalnih pogleda, poželjno je da u razrešavanju učestvuju upravo projektantni tih problematičnih lokalnih pogleda.

53. Kojim principima se rukovodi pri spajanju i restrukturiranju lokalnih shema? Objasniti ih.

Tri osnovna principa kojima je potrebno rukovoditi se prilikom spajanja i restrukturiranja lokalnih shema (nakon usaglašavanja konflikata) su:

- Potpunost – svi koncepti (entiteti i odnosi) iz lokalnih shema se moraju očuvati prilikom prenošenja u globalnu shemu.
- Minimalnost – nakon ostvarivanja potpunosti spajanjem lokalnih shema u globalnu, potrebno je eliminisati svu redundantnost iz globalne sheme.
- Razumljivost – najvažniji princip je da globalna shema mora biti razumljiva svim korisnicima, kako implementatorima, tako i naručiocima posla (radi potencijalnih ispravki propusta u zahtevima).

54. Objasniti motivaciju za pravljenje modela entiteta i odnosa.

Osnovna motivacija iza modela entiteta i odnosa jeste da se usvoji prirodniji pristup pri modeliranju stvarnog sveta bazom podataka (pod pretpostavkom da se stvarni svet sastoji od entiteta i odnosa – ova pretpostavka zapravo dovodi do određenih problema). Naime, ostali modeli (u trenutku pisanja inicijalnog rada na ovu temu, to su bili mrežni, hijerarhijski i relacioni model) ne čuvaju meta-informacije o entitetima i odnosima između njih – relacioni model uopšte ne razlikuje entitete od odnosa, pa se gube određene semantičke informacije o stvarnom svetu koji se modelira.

U praksi, ne postoji ni jedna implementacija (baza podataka) koja koristi model entiteta i odnosa. Međutim, on je veoma pogodan za konceptualno projektovanje baze, te se često koristi kao jedan od inicijalnih koraka pri projektovanju.

55. Objasniti osnovne koncepte i pretpostavke modela entiteta i odnosa.

U originalnom radu koji opisuje model entiteta i odnosa, pominju se četiri nivoa pogleda, tj. načina posmatranja, na podatke:

1. Nivo informacija koje se tiču entiteta i odnosa, a koje postoje samo u našem umu – odgovara konceptualnom modelu i apstraktnim semantičkim informacijama.
2. Nivo strukture informacija, tj. način organizovanja informacija, u kome se entiteti i odnosi predstavljaju podacima – odgovara logičkom modelu, tj. odnosi se na strukturu podataka koji se čuvaju u bazi podataka.
3. Nivo strukture podataka nezavisne od pristupnog puta, tj. koje ne zahtevaju sheme pretraživanja, indeksiranja i slično – odgovara višem delu fizičkom modela.
4. Nivo strukture podataka koje zavise od pristupnog puta – odgovara nižem delu fizičkog modela, tj. posmatraju se podaci bez skoro ikakve apstrakcije.

Prema ovome, mrežni nivo se prvenstveno bavi 4. (najnižim) nivoom (setimo se – pretraživanje podataka se vrši navigacijom kroz njih, tj. praćenjem pokazivača), relacioni prvenstveno 2. i 3. nivoom, a model entiteta i odnosa bi trebalo da se bavi 1. i 2. nivoom.

Što se tiče 1. nivoa pri posmatranju ER modela, važni su koncepti entiteta i odnosa:

- Entiteti predstavljaju stvari iz stvarnog domena koji modeliramo, koje se mogu jednoznačno identifikovati (student, studijski program, ispit, ...). Svaki pojedinačni entitet se klasifikuje u različite skupove entiteta (svakom od ovih skupova odgovara predikat koji proverava da li mu neki entitet pripada), s tim što ti skupovi ne moraju biti disjunktni. Dodatno, informacije o entitetima se izražavaju preko parova atribut-vrednost, pa atribute formalno možemo definisati kao funkcije nad skupom entiteta.
- Odnosi predstavljaju međusobno pridruživanje dva ili više entiteta, gde entitet, u okviru odnosa, ima ulogu funkcije koju obavlja u njemu. Formalno, odnosi se mogu modelirati kao relacije između entiteta učesnika. Slično kao i kod entiteta, informacije o odnosima se modeliraju parovima atribut-vrednost.

Ukoliko posmatramo 2. nivo ER modela, glavni koncepti su koncept primarnog ključa i relacije:

- Primarni ključ predstavlja sredstvo za razlikovanje i jednoznačno referisanje elemenata skupova entiteta ili odnosa.
- Skupovi entiteta i skupovi odnosa se modeliraju relacijama (intuitivno, tabelama).

56. Objasniti kako se ER model uklapa u nivoe 1 i 2 pogleda na podatke.

ER model se upravo bavi prepoznavanjem entiteta i odnosa između njih u realnom sistemu koji modeliramo. Ovo direktno odgovara nivou 1. Međutim, iako se po originalnom radu ER model odnosi i na 2. nivo, to je relativno upitno, jer i dan danas (a pitanje da li će ikada to toga i doći) ne postoji ni jedna implementacija SUBP sa modelom entiteta i odnosa u svojoj osnovi, pa i nema svrhe razmišljati o strukturiranju informacija u ovom modelu. Umesto ovoga, čak i sam autor (Peter Chen) je predložio da se za prelazak na 2. nivo izvrši prevođenje ER modela na relacioni model.

57. Objasniti razliku između entiteta i odnosa u ER modelu.

Entiteti – stvari koje se jednostavno mogu identifikovati (sve što se fizički ili misaono postoji kao prepoznatljiv element stvarnog domena) (npr. student, studijski program, ...).

Odnosi – međusobno pridruživanje dva ili više entiteta (npr. student-studijski program).

Međutim, stroga granica između ova dva koncepta u realnom domenu se ne može napraviti – nije uvek moguće razgraničiti šta je entitet, a šta odnos (a ponekad je i jedno i drugo).

Ukoliko malo formalnije definišemo entitete i odnose, dobijamo sledeće:

Entiteti se klasifikuju u različite skupove kojima je pridodat predikat koji za neki entitet proverava da li pripada tom skupu – ovaj predikat je potreban jer jedan entitet može pripadati više skupova entiteta istovremeno. Sa druge strane, skup odnosa je matematička relacija između N entiteta koji pripadaju nekim, tačno određenim, skupovima entiteta, gde su elementi ovog skupa odnosi. Uloga entiteta u odnosu je onda funkcija koju on u tom odnosu obavlja.

58. Šta su slabi i jaki entiteti?

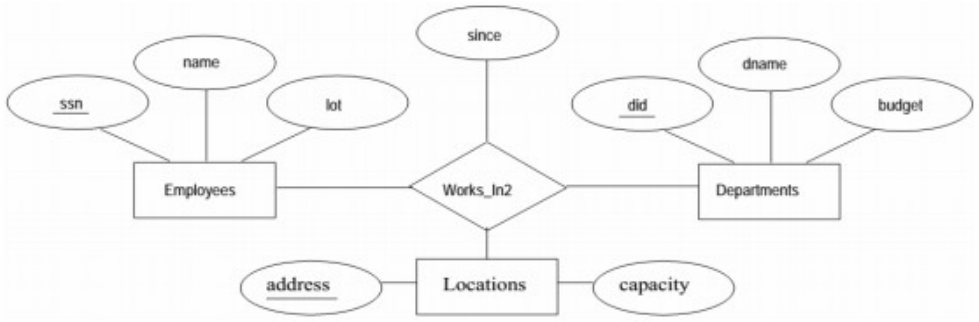
Slabi entiteti su entiteti koji se ne mogu identifikovati sami po sebi, tj. u njihovoj identifikaciji mora učestvovati odnos sa nekim drugim entitetom (slaba entitetska relacija). Sa druge strane, jaki entiteti se mogu identifikovati sami za sebe, bez potrebe za učestvovanjem u odnosu sa nekim drugim entitetom.

U praksi, ovo bi značilo da slabi entiteti kao deo svog primarnog ključa imaju referencu na neki jaki entitet.

Primer: entitet Student (id_fakulteta, broj_indeksa) je slab entitet u odnosu sa entitetom Fakultet – bez fakulteta, studenta je nemoguće identifikovati.

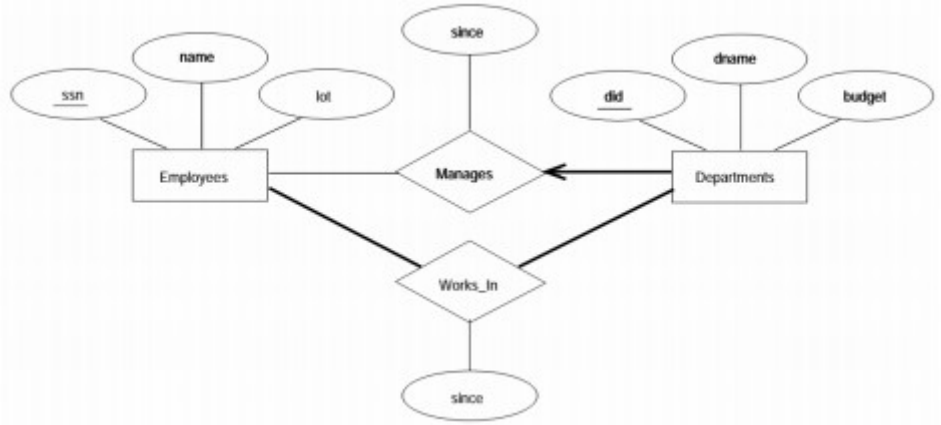
Važno je napomenuti da slabi entiteti uopšte nisu manje važni od jakih entiteta – zapravo, u većim bazama podataka, slabi entiteti najčešće čine čak i većinu entiteta.

59. Nacrtati primer ER dijagrama i objasniti njegove osnovne elemente.



- Skupovi entiteta se označavaju pravougaonicima (Employees, Locations, Departments).
- Odnosi se označavaju rombovima (Works_in) – više entiteta može učestvovati u odnosu.
- Atributi se označavaju elipsama, sa tim što, ukoliko neki atribut učestvuje u primarnom ključu (mada određivanje primarnih ključeva više pripada drugom, tj. strukturnom nivou), on se podvlači. Odnosi takođe mogu imati atribute.

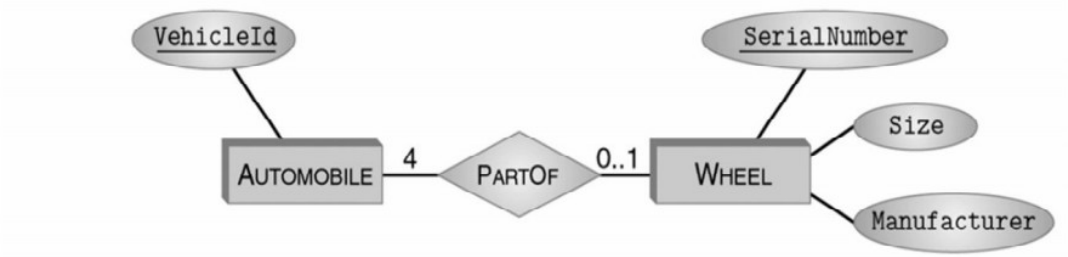
60. Šta su i kako se na ER dijagramima označavaju uslov ključa, uslov učešća i puno učešće?



Uslov ključa označava da neki entitet u odnosu jednoznačno određuje i sve ostale entitete u odnosu. Na ER dijagramu, on se označava strelicom (od tog entiteta ka odnosu). U primeru sa dijagrama, svako odeljenje ima najviše jednog rukovodioca (ako znamo odeljenje, tj. rukovodioca, znaćemo i sve zaposlene kojima on rukovodi) – ovo odgovara 0..1 kardinalnosti.

Uslov (punog) učešća označava da svaki entitet nekog skupa entiteta mora da učestvuje u bar jednom odnosu tog tipa. Ovo se označava podebljanom linijom između skupa entiteta i odnosa. U primeru dijagrama, ovo bi značilo da svaki radnik radi u barem jednom odeljenju, kao i to da svako odeljenje ima barem jednog radnika. Vidimo da ovo odgovara 1..M kardinalnosti. Takođe, vidimo da svako odeljenje ima barem jednog rukovodioca, pa uz uslov ključa (ima najviše jednog rukovodioca), dobijamo da svako odeljenje ima tačno jednog rukovodioca, ili 1..1 kardinalnost. Ukoliko učešće u odnosu nije puno, ono je parcijalno.

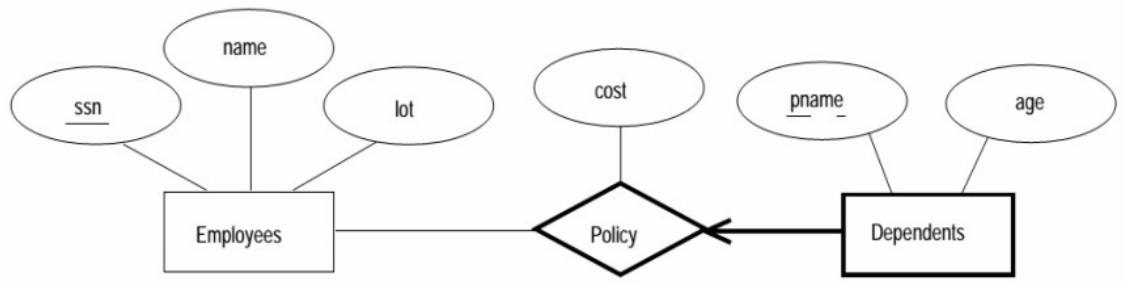
61. Kako se na ER dijagramima označava kardinalnost i šta tačno označava?



Kardinalnost označava donju i gornju granicu u smislu broja entiteta koji učestvuju u jednom odnosu. Ona se označava na strani entiteta koji učestvuje u odnosu (suprotno od UML-a). Na primer, točak može biti deo nula ili tačno jednog automobila (0..1) – ovu kardinalnost označavamo na strani točka. Sa druge strane, automobil može sadržati tačno 4 točka, što označavamo sa strane automobila. Primeri ispravnih oznaka kardinalnosti: 1, 4, 0..1, 2..5, 1..*, 0..* (negde se koristi i npr. M umesto * za označavanje "više" – nedefinisane gornje granice, npr. 0..M – nula ili više).

62. Šta su slabi i jaki entiteti i kako se označavaju na ER dijagramima?

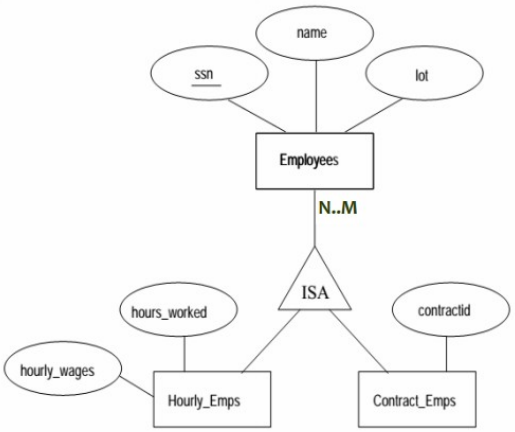
Slabi entiteti su entiteti koji se ne mogu identifikovati sami po sebi, tj. u njihovoj identifikaciji mora učestvovati odnos sa nekim drugim entitetom (slaba entitetska relacija). Sa druge strane, jaki entiteti se mogu identifikovati sami za sebe, bez potrebe za učestvovanjem u odnosu sa nekim drugim entitetom.



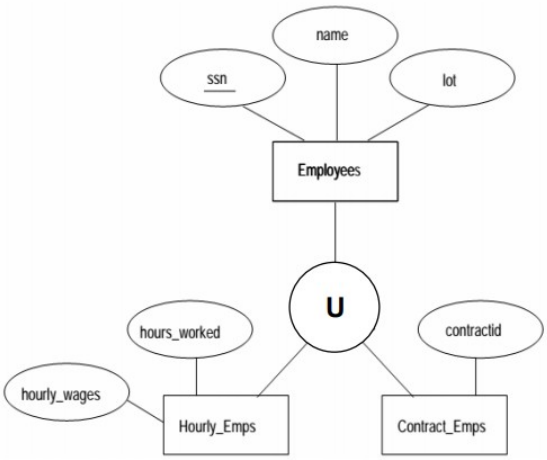
Kardinalnost na strani slabog entiteta u odnosu je uvek tačno 1 (1..1), pa se ova veza označava podebljanom strelicom (uslov ključa i učešća, zajedno). Dodatno, i odnos i slabi entitet se podebljavaju (ili označavaju duplom linijom). U dijagramu sa primera, osiguranika ima smisla posmatrati samo u kontekstu polise osiguranja koja je vezana za nekog zaposlenog.

63. Kako se na ER dijagramima označavaju hijerarhije entieteta?

Hijerarhije entiteta se odnose na skupove entiteta i opštiji su pojam od nasleđivanja u OO. Naime, uvodi se pojam uslova preklapanja (da li isti entitet može pripadati dva izvedena skupa iste hijerarhije?) i pojam uslova prekrivanja (da li svaki entitet mora pripadati nekom od izvedenih skupova hijerarhije?). Kardinalnost na strani izvedenih klasa je uvek tačno 1 (1..1) – zaposleni po ugovoru i zaposleni koji se plaćaju po satu su sigurno zaposleni. Sa druge strane, kardinalnost na strani osnovnog skupa entiteta zavisi upravo od uslova preklapanja i uslova prekrivanja.



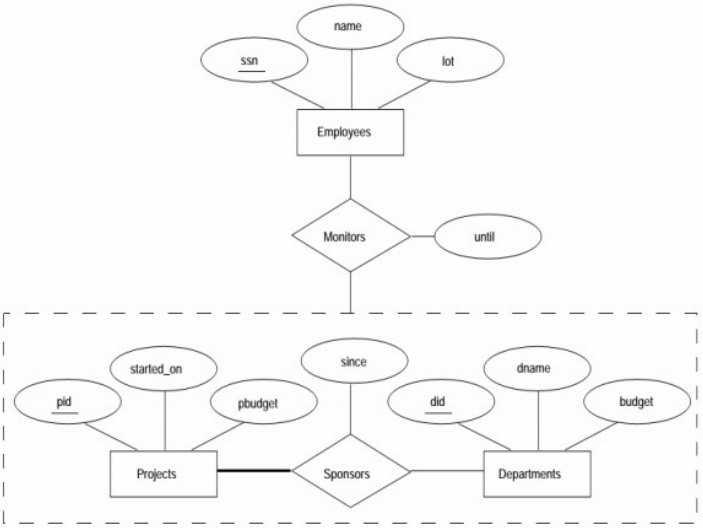
Specijalni slučaj hijerarhije gde važi uslov prekrivanja, a ne važi uslov preklapanja, tj. gde je kardinalnost na strani bazne klase 1..1 se nazivaju unije i umesto trougla sa ISA oznakom se koristi krug sa U oznakom.



64. Šta je i kako se na ER dijagramima označava "agregacija"?

Agregacija predstavlja odnose koji učestvuju u drugim odnosima – u ovom slučaju, ovakve odnose (one koje učestvuju u drugim) uokvirujemo isprekidanom linijom.

U datom primeru, neko odeljenje može da sponzoriše (ili sprovodi) nekakav projekat. Međutim, i neki zaposleni bi trebalo da može da nadgleda to sprovođenje projekta. Jedna alternativa bi bila pravljenje ternarnog odnosa (zaposleni na sponzoriše), ali to nije toliko jasno, a takođe i nije toliko dobro rešenje u slučaju da se neki projekat može izvoditi bez nadgledanja zaposlenog.



65. U čemu je osnovni doprinos modela entiteta i odnosa?

Osnovni doprinos ER modela je u tome što je on mnogo bolje prilagođen (u odnosu na relacioni model) najvišem nivou posmatranja stvarnog domena kojeg modeliramo zbog toga što bolje čuva semantiku entiteta i odnosa među njima (u relacionom modelu čak i nismo pravili razliku između entiteta i odnosa). Takođe, on pruža i dijagramsku tehniku zapisa kao jednostavno i izražajno sredstvo komunikacije.

Iz ovih razloga, model entiteta i odnosa je izuzetno dobar alat za modeliranje na konceptualnom nivou, tj. alat za projektovanje.

66. Objasniti osnovne slabosti ER modela.

Glavna slabost modela entiteta i odnosa je u tome što nije uvek moguće tačno odrediti šta je entitet, a šta odnos – ne postoji nikakav formalni način za utvrđivanje ovoga. Takođe, još jedan problem je u razlikovanju složenih atributa i entiteta – da li neki složeni atributi (koji se sastoje od više skalarnih atributa) treba da budu entiteti ili ne (npr. adresa – ulica, broj, grad)?

Ipak, ovo u praksi nije toliki problem jer se, u suštini, konceptualni ER model svakako prevodi u relacioni model gde se razlika između entiteta i odnosa gubi.

Dodatni, više tehnički problem je pretrpanost ER dijagrama u kompleksnijim bazama podataka koje mogu imati i po par stotina entiteta. Ipak, ovo se do neke mere može prevazići ukoliko se modeliranje vrši u koracima.

67. Gde se danas koristi ER model?

Model entiteta i odnosa se, zbog svog očuvanja semantike stvarnog domena, danas najviše koristi pri konceptualnom projektovanju za pravljenje konceptualnog modela, koji se kasnije koristi za pravljenje logičkog modela (npr. prevođenjem u relacioni model podataka).

68. Koje su osnovne dileme/nedoumice do kojih dolazi pri pravljenju ER modela?

- Da li je nešto bolje modelirati entitetom ili atributom?
- Da li je nešto bolje modelirati entitetom ili odnosom?
- Da li je bolje koristiti složeni (npr. ternarni) odnos ili više binarnih?
- Da li je bolje koristiti agregaciju ili ternarni odnos?

69. Objasniti dilemu "entitet ili atribut".

U slučaju modeliranja složenog atributa (atributa koji je sačinjen od više prostih atributa, npr. adresa – broj, ulica, grad), pri pravljenju ER modela, imamo više opcija. Naime, možemo ga modelirati kao jedan složeni atribut, više prostih atributa ili jedan entitet (u suštini, čak i prost atribut se možda može modelirati kao entitet), a pitanje je šta je najbolji izbor.

Što se tiče izbora između složenog atributa i više prostih atributa, često je bolje izabrati jedan, složeni atribut u slučaju da da on ima neku svoju internu strukturu gde su njegove komponente u nekom smislu zavisne. Na primer, ukoliko imamo atribut adresu i neko se preseli, verovatno će se menjati sve komponente tog atributa (ulica, broj, grad), pa je bolja ideja da ona bude složeni atribut. Ovakvi atributi svakako nisu podržani u relacionom modelu – u njemu će se koristiti više prostih.

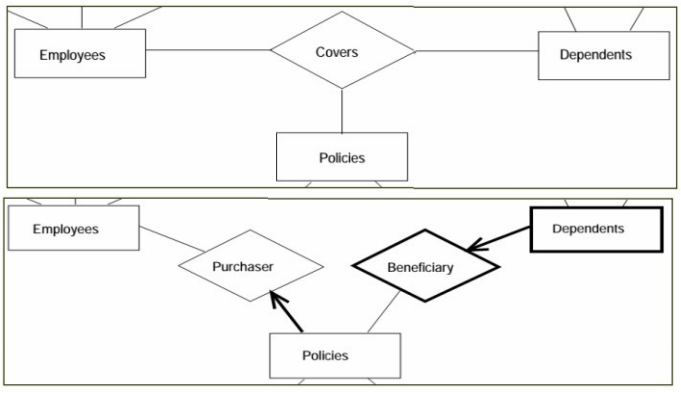
Ipak, važniji izbor je između atributa i entiteta. Jedno pravilo koje se može koristiti za izbor u korist entiteta je sledeće – ukoliko postoji međuzavisnost između vrednosti "atributa" kroz različite entitete (na primer, ukoliko u okviru entiteta Student čuvamo broj ESPB studijskog programa na kojem oni studiraju i odlučimo da promenimo obim ESPB, onda ćemo morati da menjamo taj obim za svakog studenta), onda je bolja odluka koristiti entitet, a ne atribut.

70. Objasniti dilemu "entitet ili odnos".

Prilikom pravljenja modela entiteta i odnosa, često je potrebno napraviti izbor između toga da li nešto iz stvarnog domena koji modeliramo treba modelirati kao entitet, ili kao odnos. Realnost je da ne postoji neko striktno pravilo koje će odrediti šta je bolje u svakom mogućem slučaju i da je u velikom broju slučajeva jednako ispravna bilo koja odluka, te je potrebno izabrati onu koja je "razumljivija" za onoga ko će proučavati konceptualni model.

Međutim, u jednoj situaciji je često najbolje da odnos "preraste" u entitet – ukoliko se atributi ovog odnosa istovremeno odnose na veći broj instanci ovog odnosa, tj. ukoliko se oni ponavljaju (postoji redundantnost), onda je najbolje napraviti entitet.

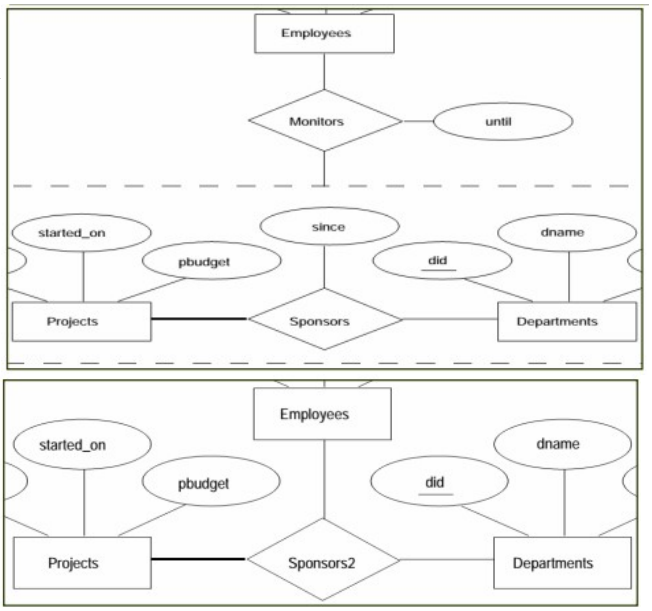
71. Objasniti dilemu "složeni odnos ili više binarnih odnosa".



Tokom pravljenja modela entiteta i odnosa, može doći do situacije gde je potrebno birati između složenijeg, na primer ternarnog, ili više binarnih odnosa. Naime, ternarni odnos, pri prevođenju u relacioni model, stvara redundantnosti, dok u slučaju binarnog to nije slučaj. Ipak, u konceptualnom modelu, ova redundantnost ne postoji.

Jedno pravilo za podelu složenog odnosa na više jednostavnijih glasi da se to može (ali ne mora) uraditi samo onda kada se tom podelom i dalje čuva semantička smisao dobijenih odnosa. Čak i kada je to moguće uraditi, ali složeniji odnos ipak ima očiglednu semantiku, onda ne treba vršiti podelu.

72. Objasniti dilemu "agregacija ili ternarni odnos".



Kod pravljenja modela entiteta i odnosa, jedna moguća dilema je u izboru između agregiranog odnosa (odnos koji učestvuje u odnosima – možemo posmatrati kao skup odnosa ili skup entiteta) i ternarnog odnosa. Opet, najbolje je izabrati ono što je jasnije u konkretnom slučaju, ali neka smernica za izbor agregiranog odnosa je da, ukoliko u odnosu imamo neki neobavezan deo (deo odnosa koji ne mora uvek postojati, npr. zaposleni koji nadgleda izvođenje projekta), onda je potencijalno bolje agregirati odnos.

73. Kako entiteti i odnosi ER modela mogu da se prevode u relacioni model?

Prevođenje entiteta iz ER modela u relacioni model je uglavnom pravolinijsko – svaki entitet se prevodi u jednu relaciju, a atributi tog entiteta u atribute te relacije. Za primarni ključ se uglavnom uzimaju atributi obeleženi kao deo ključa u ER modelu, uz potencijalno dodavanje surogat ključeva.

Što se tiče odnosa, priča je malo komplikovanija. Naime, većina odnosa se takođe može prevesti u relacije tako što se kao primarni ključ te relacije postavljaju (neki od, zavisno od kardinalnosti) primarni ključevi entiteta koji su učestvovali u odnosu (ovi primarni ključevi su ujedno i strani ključevi na date entitete). Međutim, nekada ovo nije potrebno, već se odnosi, u slučaju određenih kardinalnosti, mogu "ugraditi" u relacije koje odgovaraju entitetima koji učestvuju u tom odnosu. Naime, ukoliko je sa strane nekog odnosa kardinalnost 0..1 ili 1..1 (a sa druge strane uglavnom kardinalnost više, tj. *), tada se u relaciju entiteta tog odnosa koji je sa strane gde je gornja granica kardinalnosti 1, može direktno ugraditi taj odnos postavljanjem stranog ključa na entitet sa druge strane (na primer, (entitet) Radnik koji radi u tačno jednom (entitetu) Odeljenju – relacija za radnika bi imala postavljen strani ključ na relaciju koja odgovara odeljenju). Treba obratiti pažnju da ovo može stvoriti određene probleme, naručito u slučaju rekurzivnih odnosa (brak kao odnos i entitet osobe – može doći do veze da je osoba A u braku sa osobom B, a osoba B u braku sa osobom C).

Dodatno, u slučaju atributa koji predstavljaju skupove podataka (npr. atribut studenta koji predstavlja sve njegove položene ispite), potrebno je predstaviti takav atribut kao entitet (uz dodatne odnose između entiteta), pa ga potom prevesti u relaciju.

74. Kako se hijerarhije ER modela prevode u relacioni model?

Za prevođenje hijerarhija iz ER modela u relacioni model zapravo imamo tri opcije, svaku sa svojim prednostima i manama:

- Prevođenje čitave hijerarhije u jednu relaciju – koristimo nedefinisane vrednosti za atribute koji ne pripadaju konkretnom entitetu. Ovo opšte rešenje radi dobro i u slučaju uslova prekrivanja (ili manjka uslova prekrivanja) i u slučaju uslova preklapanja. Međutim, velika količina memorije se koristi na nedefinisane vrednosti.
- Listovi se prevode u relacije, nasleđujući atribute roditelja iz hijerarhije – ukoliko uslov prekrivanja ne važi (moguće je da entitet ne pripada ni jednom listu iz hijerarhije), bilo bi potrebno uvesti i relaciju za roditelja u hijerarhiji, ali onda dolazi do sledećeg problema – operacije nad "roditeljskom" relacijom bi trebalo da se izvrše i na relacijama listova (jer i oni su entitet koji odgovara roditelju). U ovom slučaju, ili se mogu čuvati dodatni redovi u roditelju za sve listove (što uvodi redundantnost), ili da se sve operacije rade uz vršenje unije nad ovim relacijama.
- Svaki entitet se prevodi u relaciju, sa tim da se nasleđuje samo primarni ključ roditelja (kao strani ključ) – za postavljanje upita nad ovim relacijama, potrebno je vršiti dosta spajanja. Međutim, ovo rešenje radi dobro i kada je prekrivanje dozvoljeno, i kada nije, a radi dobro i u slučaju preklapanja. Međutim, problem nastaje u situaciji kada preklapanje nije dozvoljeno – kako proveravati da se neki primarni ključ roditelja ne javlja u više od jednog lista? Ovo je relativno skupa provera.

Izbor najbolje strategije zavisi od konkretnog slučaja. Interesantno je to da, ukoliko imamo dublju hijerarhiju, moguće je na različitim nivoima koristiti različite strategije prevođenja, zavisno od potrebe.

75. Kako se slabi entiteti ER modela prevode u relacioni model?

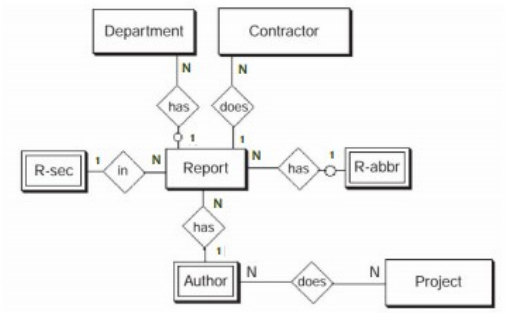
Slabi entiteti se obično prevode u zasebnu relaciju koja će, uz svoje atribute, kao deo svog primarnog ključa, imati i pridodate atribute primarnog ključa jakog entiteta sa kojim je u vezi.

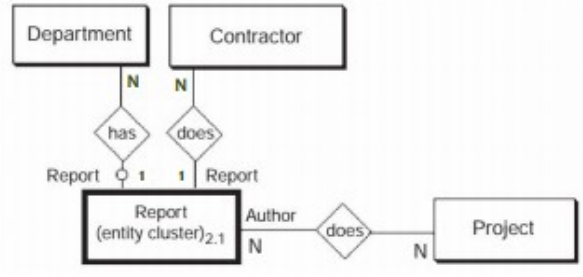
76. U kojoj fazi projektovanja baza podataka se najviše koristi ER model?

ER model se najviše koristi (zapravo, u ovoj fazi se i kreira) u konceptualnoj fazi projektovanja, ali se koristi i prilikom prevođenja na neki model podataka (npr. relacioni) u fazi logičkog projektovanja.

77. Šta predstavlja korak *Grupisanje entiteta* pri pravljenju konceptualnog modela BP? Navesti ciljeve i osnovne postupke. Zašto je to važno?

Pošto korak objedinjavanja pogleda može rezultovati shemom sa čak i više stotina entiteta, potrebno je nekako uprostiti dobijeni dijagram. Korak grupisanja entiteta kao zadatak ima pravljenje većeg broja manjih i preglednijih dijagrama modela, postupkom grupisanja, od polaznog, kompleksnog dijagrama. Iako ovaj korak nije suštinski neophodan (ne pravi se novi model, već se njegov prikaz uprošćava), često je od velikog značaja za razumevanje složenijih modela dobijenih iz prethodnih koraka.





78. Navesti i objasniti principe grupisanja entiteta pri pravljenju konceptualnog modela BP.

Četiri principa grupisanja entiteta su:

- Grupisanje prema dominantnosti – entiteti koji učestvuju u većem broju odnosa se smatraju dominantnijim, te su oni pogodni za grupisanje (gde bi najdominantniji entitet predstavljao grupu).
- Grupisanje prema apstraktnosti – vrši se u slučaju postojanja hijerarhija, grupu predstavlja bazni entitet te hijerarhije (u slučaju specifičnih odnosa za elemente hijerarhije, onda je moguće podeliti je na više grupa).
- Grupisanje prema uslovima – vrši se uklanjanje elemenata dijagrama čija je primarna uloga opisivanje uslova integriteta, ali nisu od primarnog značaja za celinu.
- Grupisanje prema odnosima – vrši se gurpisanje odnosa višeg reda (sa više od dva učesnika) i odgovarajućih entiteta u jednu celinu.

79. Objasniti grupisanje entiteta prema dominantnosti.

Entiteti koji učestvuju u većem broju odnosa se smatraju dominantnijim, te su oni pogodni za grupisanje (gde bi najdominantniji entitet predstavljao grupu, tj. slabiji entiteti bi se pridruživali njemu). Primer ovoga bi bilo pridruživanje delova celini u slučaju agregacije ili kompozicije.

80. Objasniti grupisanje entiteta prema apstraktnosti.

Grupisanje prema apstraktnosti se vrši u slučaju postojanja hijerarhija, gde bi grupu predstavljao bazni entitet te hijerarhije (u slučaju specifičnih odnosa za elemente hijerarhije, onda je moguće podeliti je na više grupa).

81. Objasniti grupisanje entiteta prema uslovima.

Grupisanje prema uslovima se vrši uklanjanjem elemenata dijagrama (atributa, odnosa, entiteta, komentara) čija je primarna uloga opisivanje uslova integriteta, a koji nisu od primarnog značaja za celinu. Primer bi bio uklanjanje entiteta "Grad" koji predstavlja skup dopuštenih gradova. On može biti u odnosu sa većim brojem, potencijalno dominantnih, entiteta, pa čak i više puta (npr. za entitet "Osoba" čuvamo atribute "Mesto rođenja" i "Prebivalište" koji su strani ključevi za entitet "Grad"). Razlika ovoga u odnosu na grupisanje prema dominantnosti je ta što entiteti koje eliminišemo mogu biti u vezi sa većim brojem drugih dominantnih entiteta.

82. Objasniti postupak grupisanja entiteta.

Grupisanje se vrši rekurzivno, u narednim koracima:

- Prvo je potrebno prepoznati sve elemente sheme koji formiraju funkcionalne oblasti, tj. neke grupe podataka koje se najčešće koriste zajedno (ili poslovne jedinice).
- Entiteti se zatim grupišu, s tim da svaka dobijena grupa mora pripadati tačno jednoj funkcionalnoj oblasti uočenoj u prethodnom koraku (ukoliko je ona deljena između više funkcionalnih oblasti, bolje je ne praviti takvu grupu).
- Rekurzivnom primenom prethodnih koraka se prave grupe višeg reda (grupe unutar grupa).
- Proverava se ispravnost dijagrama nakon grupisanja – da li su svi interfejsi između elemenata dijagrama konzistentni?

83. Kako se UML koristi pri modeliranju baza podataka?

UML se, zbog svoje dobre mogućnosti predstavljanja semantike odnosa (bolje i od dijagrama tabela i od ER dijagrama), najčešće koristi prilikom faze konceptualnog modeliranja baze podataka, mada se, delom, može koristiti i prilikom logičkog modeliranja.

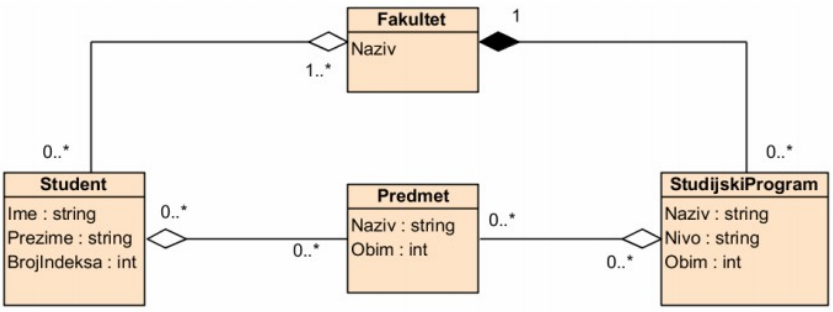
Motivacija za ovakav pristup modeliranju podataka je u tome što je UML već dobro rasprostranjen i poznat jezik za modeliranje softvera, pa često nije potrebno veliko "doučavanje" za njegovu primenu. Dodatno, ovo približava jezik modeliranja podataka jeziku koji se koristi za modeliranje softvera koji će te podatke i koristiti. Da bi se upotreba UML-a dodatno pojednostavila i prilagodila za ove svrhe, predlagan je poseban "profil" UML-a za modeliranje podataka, mada njegov razvoj nikada nije zaživeo jer se shvatilo da suštinski i nije neophodan.

Međutim, u zavisnosti od logičkog modela podataka, dolazi do potencijalnog problema. Naime, UML opisuje strukturu klasa, što ne bi predstavljalo problem kod objektnog modela podataka, međutim, u slučaju relacionog modela, koriste se relacije (tj. skupovi) za modeliranje podataka. Klase i skupovi nisu isto, (klase predstavljaju tip podataka), te se postavlja pitanje – kako modelirati skupove podataka UML-om koji opisuje klase podataka? Ovaj problem se, u praksi, prevazilazi na tri moguća načina:

1. Ukoliko modeliramo više skupova istog tipa, možemo uvesti više klasa, za svaki od tih skupova, sa apstraktnom baznom klasom koja odgovara datom tipu (na primer, student kao apstraktna bazna klasa, a student osnovnih, student master i student doktorskih studija kao izvedene klase).
2. Označavanje stereotipovima za razlikovanje klase, tj. tipa (<<type>>) i skupa (<<persistent>>). Međutim, ovo dosta opterećuje dijagram, pa se koristi samo kad je neophodno.
3. Ignorisanje problema – smatramo da je klasa skup, a ne tip, što je prihvatljivo rešenje samo za manje domene u kojima se ne očekuje više skupova istog tipa.

84. Objasniti specifičnosti dijagrama klasa domena (tj. dijagrama klasa podataka).

Prilikom modeliranja modela baze podataka UML dijagramima, koristi se tzv. strukturni (konceptualni) dijagram klasa (takođe nazivan i dijagram klasa domena u UML-u ili dijagram klasa podataka u radu sa bazama podataka). Naime, za razliku od funkcionalnih (koji opisuju samo ponašanje, tj. metode klasa) i implementacionih (koji opisuju i ponašanje i strukturu klasa) dijagrama klasa, ovi, tj. strukturni, dijagrami opisuju samo strukturu (atribute) i odnose klasa (ponašanje se skoro potpuno zanemaruje, kao i enkapsulacija). Dodatno, koriste se i razne dopune UML-a za specifične oznake.



85. Objasniti dopune UML dijagrama koje se koriste u specifičnim oblastima primene.

Dopune koje se koriste u UML dijagramima, koje se uvode u svrhu omogućavanja dodatnih označavanja i detaljnijih opisa elemenata dijagrama, su stereotipovi, označene vrednosti i proširenja.

Stereotipovi se mogu posmatrati kao vrsta šablona za klase ili atribute. Oni predstavljaju neki predefinisani skup osobina ili ponašanje koje data klasa ili atribut ima. Primeri nekih stereotipova za klase su <<abstract>>, <<persistent>> (uz <<persistent>> i <<entity>> i <<set>> - ova tri tipa označavaju da dati element predstavlja relaciju, a ne tip, te se trajno zapisuje negde (u nekoj bazi podataka), s tim što je persistent opštiji od entity po tome što struktura sačuvanog "entiteta" nije fiksirana, a set označava da nema pridodatog ponašanja), <<type>>, <<view>>, ... Primeri atributskih stereotipova su <<key>> (<<PK>>), <<FK>>, <<unique>>, <<null>>, <<not null>>, <<generated>> (generisan na osnovu drugih atributa), <<auto>> (automatski generisan, najčešće za surogat ključeve), ... Dodatno, umesto stereotipova se mogu koristiti simboli (grafička dopuna u vidu oznake, ili čak i potpuno izmenjen oblik elementa (drugačiji oblik umesto pravougaonika)).

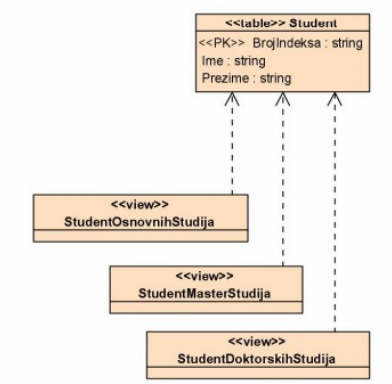
Označene vrednosti predstavljaju uopštenje stereotipova. Naime, oni se navode kao lista parova ime-vrednost, pa ih je zgodnije koristiti ukoliko to neko "svojstvo" može imati veći broj vrednosti (bolje nego da uvodimo poseban stereotip za svaku od njih) – npr. {isPersistent=true, minCount=5}, {nullable=true}, ...

Proširenja su dodatni elementi koji se povezuju sa osnovnim elementom (na koji se odnose) strelicom sa popunjenim trouglom na kraju (strelica ide od osnovnog elementa ka proširenju). Primeri proširenja su komentari. Da bi se ovi dodatni elementi međusobno razlikovali, mogu se obeležavati različitim oblicima (od pravougaonika), te je ovaj koncept dosta sličan dodatnim elementima u dijagramima tabela. Ukoliko je dodatni element obavezan, može se staviti {required} oznaka.

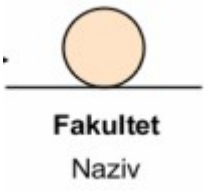
86. Šta su stereotipovi UML-a i kako se označavaju?

Stereotipovi se mogu posmatrati kao vrsta šablona za klase ili atribute. Oni predstavljaju neki predefinisani skup osobina ili ponašanje koje data klasa ili atribut ima. Primeri nekih stereotipova za klase su <<abstract>>, <<persistent>> (uz <<persistent>> i <<entity>> i <<set>> - ova tri tipa označavaju da dati element predstavlja relaciju, a ne tip, te se trajno zapisuje negde (u nekoj bazi podataka), s tim što je persistent opštiji od entity po tome što struktura sačuvanog "entiteta" nije fiksirana, a set označava da nema pridodatog ponašanja), <<type>>, <<view>>, ... Primeri atributskih stereotipova su <<key>> (<<PK>>), <<FK>>, <<unique>>, <<null>>, <<not null>>, <<generated>> (generisan na osnovu drugih atributa), <<auto>> (automatski generisan, najčešće za surogat ključeve), ...

Stereotipovi klasa se najčešće označavaju iznad naziva klase, dok se stereotipovi atributa označavaju levo od naziva atributa.



Dodatno, umesto stereotipova se mogu koristiti simboli (grafička dopuna u vidu oznake, ili čak i potpuno izmenjen oblik elementa (drugačiji oblik umesto pravougaonika, npr. kružić sa linijom ispod za perzistentne objekte)).



87. Objasniti osnovne odnose u dijagramima klasa podataka.

Osnovni odnosi koji se koriste u UML dijagramima klasa podataka su:

- Asocijacija – označava odnos između dve klase, te može predstavljati funkcionalni odnos između njih (klasa A radi nešto za klasu B) ili strukturalni odnos (klasa A predstavlja nešto za klasu B). U kontekstu modeliranja podataka, skoro isključivo se koristi strukturalna asocijacija i ona označava da jedna klasa "zna" za objekte druge klase, što najčešće označava postojanje stranog ključa. Asocijacija se na dijagramima označava običnom nepopunjenom strelicom.
- Agregacija – može se posmatrati kao posebna vrsta asocijacije koja ima semantiku sastojanja (objekat neke klase se sastoji od objekata neke druge klase sa kojom je u agregiranoj vezi). Ovakva veza označava da jedan isti deo (objekat) može biti sadržan u više različitih, složenijih objekata, kao i to da taj deo može da postoji nezavisno od složenijih objekata koji ga sadrže. Označava se "strelicom" koja na strani složenog objekta ima prazan romb.
- Kompozicija – slična semantika agregaciji, s tim što jedan deo može pripadati samo jednom složenom objektu (ovaj odnos je uvek binaran), i on ne može postojati nezavisno od tog složenog objekta. Odgovara slaboj entitetskoj relaciji u ER modelu. Označava se strelicom koja na strani složenog objekta ima popunjen romb (primer – odnos student – fakultet – student nije student bez fakulteta i on može biti student tačno jednog fakulteta).
- Nasleđivanje – predstavlja odnos između osnovne klase, tj. opšteg slučaja (generalizacije) i izvedene klase, tj. posebnog slučaja (specijalizacije). U relacionom modelu ne postoji odgovarajući koncept, ali postoji u ER modelu. Označava se strelicom sa praznim trouglićem prema baznoj klasi. Primer bi bila opšta klasa student i izvedene klase student osnovnih, master i doktorskih studija.

Često dolazi do zabune u izboru između obične asocijacije i agregacije. Naime, oni imaju jako sličnu semantiku i najčešće se mogu zameniti. Međutim, postoje i neke situacije gde asocijacija bolje prenosi semantiku od agregacije. Na primer, student konceptualno ne sadrži predmete, ali na nivou podataka sadrži UPISANE predmete. Ovde je, onda, možda bolje upotrebiti asocijaciju između studenta i predmeta, ali može i biti signal da je potrebno uvesti tabelu upisani predmeti sa kojom bi se pravila kompozicija, a dalje obična asocijacija sa tabelom predmeta. Ipak, najvažniji kriterijum za izbor je razumljivost dijagrama, a to zavisi od slučaja do slučaja (dodatno, ukoliko je neka klasa u velikom broju odnosa, često se za one važnije odnose koristi agregacija i kompozicija, a za one manje bitne asocijacija).

88. Kako se dijagrami klasa podataka koriste na različitim nivoima modeliranja?

Na konceptualnom nivou je glavni cilj prikazivanje same suštine modela, tj. semantike odnosa unutar domena problema, te na ovom nivou nije potrebno da dijagram klasa podataka sadrži detalje (surogat ključevi se ne navode, atributi stranih ključeva se ne navode, oznake ključeva se najčešće ne navode (osim, možda, prirodnog primarnog ključa), precizni tipovi atributa se najčešće ne navode (mada je poželjno navesti ih barem okvirno)).

Kako dijagram na logičkom nivou tačno izgleda zavisi od konkretnog modela podataka koji se koristi. Tako, na primer, ukoliko se koristi relacioni model podataka, potrebno je navesti i surogat ključeve i strane ključeve. Dodatno, tipovi atributa su precizniji.

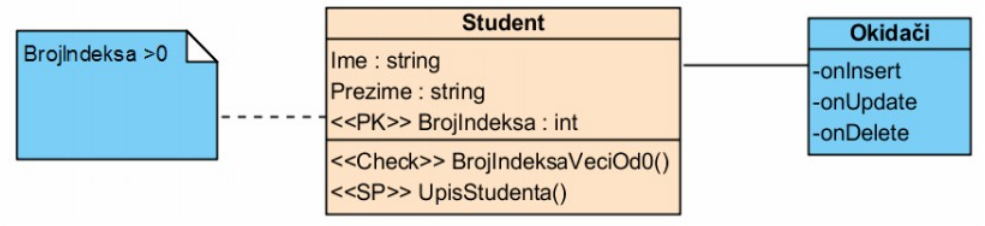
Na fizičkom nivou u centru pažnje su performanse, pa je potrebno navesti sve sa prethodnog nivoa, ali sa potpuno preciznim tipovima atributa, dodatnim elementima vezanim za indekse i slično.

89. Kako se u dijagramima klasa označavaju ključevi?

Oni se označavaju atributskim stereotipovima, najčešće <<PK>> za atribute primarnog ključa i <<FK>> za atribute stranog ključa.

90. Kako se u dijagramima klasa označavaju uslovi integriteta?

Uslovi integriteta se mogu navoditi u vidu dodatnih elemenata (npr. u okviru komentara) povezanih isprekidanom linijom za tabelu na koju se odnose, ili čak i u odeljku elementa za tabelu koji se obično, prilikom modeliranja softvera sa standardnim UML dijagramima klasa, koristi za navođenje metoda objekata.



91. Objasniti specifičnosti predstavljanja odnosa na dijagramu klasa domena (u odnosu na uobičajen dijagram klasa).

Glavna razlika je u tome što velika većina odnosa u dijagramu klasa domena (strukturalnom dijagramu klasa) ima strukturalnu prirodu, koja, za razliku od funkcionalnih odnosa (gde imamo semantiku da klasa A radi nešto za klasu B), ima semantiku gde klasa A predstavlja nešto za klasu B (pa onda, možda, objekti klase A mogu sadržati objekte klase B).

92. Objasniti različite aspekte primene dijagrama klasa domena na različitim nivoima modeliranja baze podataka.

??? (pitanje 88.)

Isti odnosi se često mogu modelirati na više različitih načina dijagramom klasa, a najbolji izbor, između ostalog, zavisi i od nivoa modeliranja na kom se ova dijagramska tehnika koristi.

Naime, na konceptualnom nivou je najvažnije očuvanje same suštine domena problema, tj. semantike odnosa iz njega. Na logičkom nivou je najvažnije ostvarivanje stabilne i jednoznačne strukture, i ona dosta zavisi od konkretnog modela podataka koji će se koristiti (relacioni, objektni, ...). Na fizičkom su, dodatno, cilj i dobre performanse.

93. Objasniti osnovne koncepte relacionog modela podataka.

Relacioni model podataka je najzastupljeniji model podataka u praksi. Verovatno najveća njegova prednost u odnosu na ostale modele podataka jeste potpuna matematička formalizacija ovog modela. Naime, svi podaci i veze između podataka su predstavljeni relacijama, koje su precizno definisani matematički objekti. Formalizacija ovog modela je pružena kroz relacionu algebru i relacioni račun.

Formalizacija kao (dobre) posledice ima neprotivrečnost relacionog modela, nedvosmislenost u primeni, mogućnost za automatsku obradu i optimizaciju, i još toga.

94. Šta je "strukturni deo relacionog modela"? Objasniti ukratko.

Strukturni deo relacionog modela se bavi načinom modeliranja podataka iz nekog domena koji posmatramo (kako se entiteti i odnosi između njih preslikavaju u relacije).

95. Šta je "manipulativni deo relacionog modela"? Objasniti ukratko.

Manipulativni deo relacionog modela pruža formalni način rukovanja modeliranim podacima iz strukturnog dela. Ovo je pruženo kroz formalne, apstraktne jezike za rukovanje podacima (relaciona algebra i relacioni račun).

96. Šta je "integritetni deo relacionog modela"? Objasniti ukratko.

Integritetni deo relacionog modela pruža način obezbeđivanja valjanosti, ispravnosti, stabilnosti (izmenom nekog podatka ne gubimo njegovu usklađenost sa ostalim podacima u bazi) i konzistentnosti podataka u bazi.

97. Navesti primer modeliranja skupa iz posmatranog domena odgovarajućom relacijom.

Prvo, formalna definicija:
Neka je *r* n-arna relacija i njen domen *Dom(r)*. Tada je model relacije *r* sledeći skup:
$model(r) = \{(a_1, ..., a_n) \mid (a_1, ..., a_n) \in Dom(r) \land r(a_1, ..., a_n)\}$.
Za sve n-torke iz *Dom(r)* važi $r(a_1, ..., a_n) <=> (a_1, ..., a_n) \in model(r)$, pa se veoma često *model(r)* i sama relacija *r* koriste u istom kontekstu, tj. uzima se da je $r = model(r)$. Primetimo da je model podskup domena. Intuitivno – model je skup torki iz domena za koje važi data relacija.

Primer:
Neka je dat skup osoba predstavljenih njihovim imenom, *O = {Marko, Luka, Ana}* i skup predmeta, takođe predstavljenih njihovim imenom, *P = {Programiranje, Analiza, Engleski}*. Posmatrajmo binarnu relaciju *PoložilaPredmet(O, P)* koja je zadovoljena ukoliko je osoba *O* položila predmet *P*. Tada je domen ove relacije, tj. *Dom(PoložilaPredmet)* jednak Dekartovom proizvodu skupova *O* i *P*, tj. $Dom(PoložilaPredmet) = O \times P$. Neka je model ove relacije zadat sa *model(PoložilaPredmet) = {(Marko, Analiza), (Marko, Programiranje), (Ana, Engleski), (Luka, Analiza)}*.

Tada važi:
- Iskaz Marko je položio Analizu je tačan.
- Iskaz Ana je položila Engleski je tačan.
- Iskaz Ana je položila Analizu nije tačan (jer *(Ana, Analiza)* ne pripada *model(PoložilaPredmet)*).
- Iskaz Nebojša je položio Engleski nije tačan (jer (Nebojša, Engleski) ne pripada *Dom(PoložilaPredmet))*.

98. Navesti i ukratko objasniti osnovne pojmove u vezi strukturnog dela relacionog modela.

- Entiteti – različiti postojeći elementi sistema koje modeliramo bazom podataka (na primer – Student, Predmet, ...).
- Odnosi – veze između entiteta (na primer – "polagao predmet", ...). I entiteti i odnosi se modeliraju relacijama (oni se ne mogu do kraja razdvojiti – neke stvari iz jednog ugla izgledaju kao entiteti, neke kao odnosi, a iz drugog ugla suprotno). Na primer, odnos "polagao predmet" bi mogli da modeliramo entitetom Ispit.
- Atributi – karakteristike entiteta koje ga opisuju. Formalno posmatrano funkcije nad entitetom.
- Relacije – u kontekstu relacionog modela često nazivamo i tabelom. Kolone ove tabele odgovaraju atributima, nazivi kolona nazivima atributa, a slogovi tabele (torke relacije) odgovaraju entitetima. Još neki pojmovi su model relacije i domen relacije (pogledati ostala pitanja).
- Ključevi i natključevi – jednistveno određuju torke u relaciji.

99. Šta su entiteti? Kako se formalno definišu atributi i relacije?

Entiteti su različiti postojeći elementi iz stvarnog domena koje modeliramo bazom podataka (npr. Student, Predmet, ...). Entitete karakterišu atributi, tj. formalno:

Skup entiteta E se karakteriše konačnim skupom atributa $A(E) = \{A_1, ..., A_n\}$ *akko:*

1. Svaki atribut A_i predstavlja funkciju koja preslikava entitete u domen atributa A_i, u oznaci D_i, tj. $A_i: E \rightarrow D_i$.
2. Svaki atribut A_i ima jedinstveni naziv t_i.

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

Skup svih atributa skupa entiteta E određuje funkciju $a(e \in E) = (A_1(e), ..., A_n(e))$. Slika skupa entiteta funkcijom a je skup $R = a(E)$. Ukoliko je ovako opisana funkcija a injektivna (funkcija a je injektivna ukoliko važi $a(x) = a(y)$ *akko* $x = y$ – za svaki par različitih ulaznih vrednosti dobijamo različite slike), onda kažemo da skup atributa $\{A_1, ..., A_n\}$ dobro karakteriše skup entiteta E.

Formalna definicija relacije – pitanje 97.

Svojstvo injektivnosti funkcije a je važno jer je relacija skup (ne može sadržati duplikate). Ukoliko je a injektivna funkcija i svi atributi su atomični (zasebni atributi su skalari – nisu nekakav Dekartov proizvod), onda je slika $R = a(E)$ relacija.

100. Objasniti pojmove ključ i natključ. U čemu je njihov značaj?

Natključ K neke relacije R je podskup njenih atributa, tj. $K \subseteq A(R)$, za koji važi da on jednoznačno određuje torke relacije (kad bi izvršili projekciju funkcije atributa a za koju važi $R = a(E)$ na atribute K, ona bi ostala injektivna).

Ključ (ključ kandidat) relacije R je natključ K za koji ne postoji pravi podskup koji je takođe natključ relacije R.

Važno je napomenuti da svaki natključ sadrži barem jedan ključ (jer i sam ključ je natključ), a i svaka relacija ima najmanje jedan ključ (u najgorem slučaju, to je skup svih njenih atributa).

Njihov značaj je u tome što nam pružaju mogućnost jednoznačnog određivanja torki relacije. Oni se, u relacionom modelu, koriste za primarne ključeve (konkretno ključevi kandidati).

101. Šta je relaciona baza podataka? Šta je relaciona shema?

Relaciona baza podataka je skup relacija. Relaciona shema je skup opisa relacija koje čine bazu podataka, gde opis jedne relacije čine njen domen i nazivi atributa.

102. Kako se modeliraju entiteti posmatranog domena u relacionom modelu?

Entiteti se modeliraju relacijama. Naime, neka je a injektivna funkcija nad skupom entiteta E, tj. $a(e \in E) = (A_1(e), ..., A_n(e))$, tada je slika $R = a(E)$ relacija sa atributima $A(R) = \{A_1, ..., A_n\}$, domenom relacije $Dom(R) = D_1 x ... x D_n$ i nazivima atributa $Kol(R) = (t_1, ..., t_n)$. Skup entiteta E sa atributima $A(E) = \{A_1, ..., A_n\}$ se modelira relacijom R.

103. Kako se modeliraju odnosi u posmatranom domenu u relacionom modelu?

Odnosi se u relacionom modelu modeliraju na isti način kao i entiteti – relacijama. U slučaju binarnih odnosa, postupak je sledeći (analogno važi i za odnose veće arnosti):

Neka su data dva entiteta, $e \in E$ i $f \in F$ i neka su oni modelirani kao $e=(x_1, ..., x_n)$ i $f =(y_1, ..., y_m)$. Dalje, neka su oni u nekom odnosu (bez dodatnih atributa) w. Tada ovaj odnos možemo opisati relacijom $r(e, f)$ čiji je domen $Dom(r) = ExF$. Ovaj odnos se modelira sa $r(e, f) = (x_1, ..., x_n, y_1, ..., y_m)$.

Ukoliko su K_E i K_F redom ključevi relacija E i F, onda je skup atributa $K_R = K_E \cup K_F$ ključ relacije r (jednoznačno određuju sve atribute iz nje). Tada se odnos može opisati "suženom" relacijom r' koja sadrži samo atribute iz skupa K_R.

Dodatno, nekada se odnos između dva entiteta može integrisati u okviru relacije jednog od njih (1-1 ili 0-1 veze ukoliko dopuštamo nedefinisane vrednosti). Na primer, relacija Student može sadržati atribute relacije StudijskiProgram (ili samo ključ te relacije), čime se opisuje koji studijski program je student upisao – ovo istovremeno predstavlja i odnos između studenta i studijskog programa.

104. Šta čini manipulativni deo relacionog modela?

Manipulativni deo relacionog modela pruža formalni način rukovanja modeliranim podacima iz strukturnog dela. Ovo je pruženo kroz formalne, apstraktne jezike za rukovanje podacima (relaciona algebra i relacioni račun) koji omogućavaju zadavanje upita nad podacima. Formalno definisano, upit je izračunavanje nove relacije na osnovu već poznatih relacija baze podataka.

105. Objasniti ukratko relacionu algebru.

Relaciona algebra je jedan od upitnih jezika koji pružaju mogućnost formalnog rukovanja modeliranim podacima u okviru relacione baze podataka. Ona predstavlja proširenje skupovne algebre – zadržane su osnovne skupovne operacije (unija, presek, razlika i proizvod), a dodate su operacije projekcije, restrikcije, prirodnog spajanja, slobodnog spajanja i deljenja.

106. Objasniti osnovne operacije relacione algebre.

Iz skupovne algebre su potpuno preuzete operacije unije, preseka, razlike i proizvoda. Naravno, da bi se izvršila unija, presek ili razlika nad neke dve relacije, npr. R i Q, potrebno je da one imaju isti skup atributa (i domene, i nazive).

Dodatno, za operaciju proizvoda važi (A x B) x (C x D) = A x B x C x D, što sledi iz atomičnosti atributa (A, B, C i D su skupovi atributa relacija). U slučaju proizvoda (i još nekih operacija), moguće je da se poklapaju nazivi atributa dve relacije nad kojima se ta operacija izvršava. U tom slučaju, naziv atributa se dopunjava nazivom relacije (sa leve strane) iz koje potiče:
R(A, B) x Q(B, C) = W(A, R_B, Q_B, C).

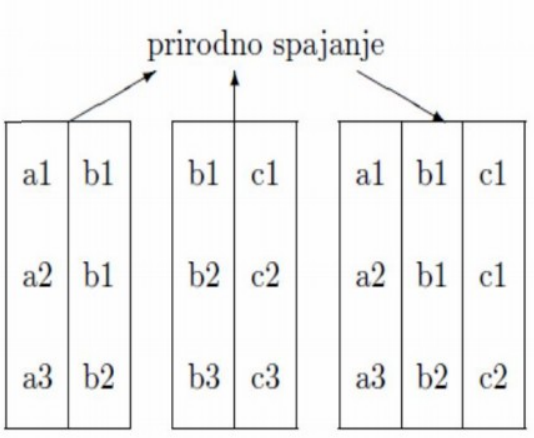
U slučaju da je u upitu potrebno iskoristiti istu relaciju više puta, moguće je uvesti alias da ne bi dolazilo do kolizija: DEFINE ALIAS R1 FOR R.

Operacija projekcije se definiše kao izbor kolona X jedne relacije (u SQL-u, to bi bila SELECT naredba): $R[X] = \{x \mid x \in Dom(X) \wedge (\exists y \in Dom(Y))\ (x, y) \in R\}$, gde je Y = A(R) \ X (ostatak atributa koji ne učestvuju u projekciji). Dobijena relacija će, pošto se radi o skupovima n-torki, imati manje ili jednako n-torki od polazne relacije R (ukoliko je X natključ relacije R, broj n-torki će biti isti). Primer: RADNIK[ime, prezime] = {(x.ime, x.prezime) | x ∈ RADNIK}.

Operacija restrikcije se definiše kao izbor torki relacije R u kojima dati atribut X ∈ A(R) ima datu vrednost x ∈ Dom(X) (u SQL-u, to bi bio WHERE): R[X=x] = {e | e ∈ R ∧ X(e) = x}. Uopštenije, X može biti proizvoljan skup atributa relacije R, a kao uslov se može koristiti proizvoljni predikat. Primer: RADNIK[org_jed_id = 40] = {x | x ∈ RADNIK ∧ x.org_jed_id = 40}.

107. Objasniti osnovne vrste spajanja u relacionoj algebri.

Prirodno spajanje relacija R i Q koje imaju zajednički podskup atributa X definiše se kao:
R * Q = {(e, f) | e ∈ R ∧ f ∈ Q ∧ X(e) = X(f)}[X ∪ Y ∪ Z] = {(x, y, z) | (x, y) ∈ R ∧ (x, z) ∈ Q}, gde su Y = A(R) \ X i Z = A(Q) \ X. ^---- projekcija



U suštini, spajaju se torke jedne relacije sa torkama druge relacije ukoliko one imaju zajedničke vrednosti atributa iz skupa atributa X. Rezultat će sadržati samo jednu kopiju zajedničkih atributa, imaće najmanje nula torki (ukoliko nema torki sa istim vrednostima zajedničkih atributa X) i najviše |R| * |Q| torki (ukoliko sve torke relacije imaju iste vrednosti zajedničkih atributa X – npr. sve imaju vrednost 1 po atributu X po kojem se vrši spajanje, pa će se svaki spojiti sa svakim).

U suštini, moguće je izvršiti prirodno spajanje i po podskupovima atributa X (relacije R) i Y (relacije Q) koji nisu isti, ali imaju iste domene i iste su veličine.

Slobodno spajanje relacije R i Q se vrši po proizvoljnom binarnom predikatu p: RxQ -> { ⊤ , ⊥ } i definiše se kao: R[p]Q = {(e, f) | e ∈ R ∧ f ∈ Q ∧ p(e, f)}. Dakle, spajaju se one torke iz jedne relacije sa torkama druge relacije ukoliko one zajedno ispunjavaju neki uslov. Slobodno spajanje je opštije od prirodnog, tj. prirodno se može izraziti preko slobodnog spajanja.

108. Objasniti operaciju deljenja relacione algebre.

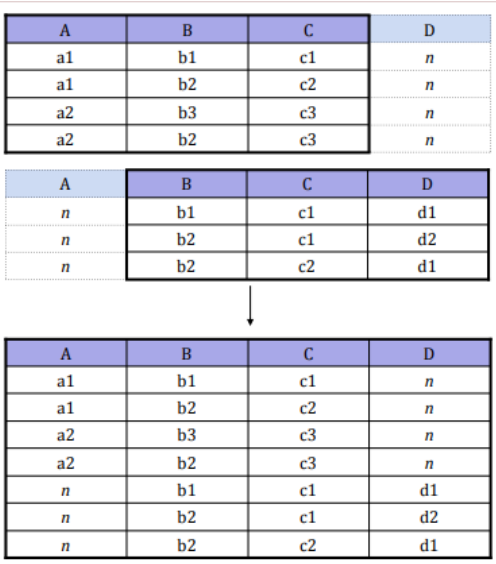
Deljenje relacije R relacijom Q po podskupu atributa $Y \subseteq A(R)$, gde je $Z = A(Q)$ skup atributa koji po broju i domenu odgovara skupu Y, definiše se kao: $R[Y\%Z]Q = \{x \mid x \in R[X] \wedge \{x\}xQ[Z] \subseteq R\}$, gde je $X = A(R) \setminus Y$ i $Z = A(Q)$ skup atributa koji po broju i domenu odgovara skupu Y.

Rezultat ove operacije je maksimalni podskup relacije R[X] takav da je njegov proizvod sa relacijom Q podskup relacije R.

Primer – izdvojiti radnike koji su radili na svim projektima (postoji rešenje za njihov rad na njemu): RESENJE[RAD_ID, PROJ_ID] [PROJ_ID % PROJEKAT_ID] PROJEKAT[PROJEKAT_ID]

109. Objasniti proširene skupovne operacije relacione algebre.

U proširenoj relacionoj algebri, uvodi se i mogućnost rada sa nedefinisanim vrednostima. Uz to, i skup istinitosnih vrednosti se proširuje nepoznatom vrednošću.



A	B	C	D
a1	b1	c1	*n*
a1	b2	c2	*n*
a2	b3	c3	*n*
a2	b2	c3	*n*

A	B	C	D
n	b1	c1	d1
n	b2	c1	d2
n	b2	c2	d1

A	B	C	D
a1	b1	c1	*n*
a1	b2	c2	*n*
a2	b3	c3	*n*
a2	b2	c3	*n*
n	b1	c1	d1
n	b2	c1	d2
n	b2	c2	d1

Proširene skupovne operacije, gde je X zajednički presek atributa relacija R i Q, su:

- Spoljašnja unija – $R \mathbin{\mathring{\cup}} Q = (R \times (Z{:}n)) \cup ((Y{:}n) \times Q)$, gde je $Y = A(R) \setminus X$ i $Z = A(Q) \setminus X$, a (A:n) relacija sa atributima A koja sadrži tačno jednu torku sa nedefinisanim vrednostima za sve atribute.
- Spoljašnji presek – $R \cap_o Q = (R \times (Z{:}n)) \cap ((Y{:}n) \times Q)$
- Spoljašnja razlika – $R \setminus_o Q = (R \times (Z{:}n)) \setminus ((Y{:}n) \times Q)$

Važno je naglasiti da operacije spoljašnjeg preseka i spoljašnje razlike nemaju puno smisla – rezultat spoljašnjeg preseka će uvek biti prazna relacija (jer će poređenje sa nedefinisanim vrednostima uvek biti $\perp$), a iz toga sledi da će rezultat spoljašnje razlike uvek biti levi operand.

Proširenja ostalih operacija – pogledati prezentacije, ne mogu više...

110. Objasniti ukratko relacioni račun. Koje vrste relacionog računa smo razmatrali?

Relacioni račun je apstraktni upitni jezik koji pruža mogućnost formalnog rukovanja modeliranim podacima u okviru relacione baze podataka. Za razliku od relacione algebre, koja je bazirana na teoriji skupova, relacioni račun je baziran na predikatskom računu i njegovoj primeni na relacijama. On je, svakako, ekvivalentan relacionoj algebri. Dve vrste koje smo razmatrali su:

- Relacioni račun torki
- Relacioni račun domena

111. Šta je relacioni račun n-torki? Objasniti osnovne karakteristike.

Relacioni račun n-torki je vrsta relacionog računa (formalnog apstraktnog jezika za zadavanje upita). Izraz računa n-torki je $\{(t_1, t_2 , ... , t_k) : f\}$, gde su:

- Promenljive $t_1, ... , t_n$ n-torne promenljive ili indeksirane n-torne promenljive oblika t[i], gde je i redni broj atributa torke t. Skup ovih promenljivih čini tzv. ciljnu listu.
- Formula f je tzv. kvalifikacioni izraz.
- Slobodne promenljive u kvalifikacionom izrazu su one i samo one koje čine ciljnu listu.

Kvalifikacioni izraz se gradi na sledeći način:

- (a1) Ako je *R* relacija i *s* n-torna promenljiva vezana za relaciju *R*, onda oznaka *R(s)* označava to vezivanje i naziva se atom pripadnosti.
- (a2) Ako su *s* i *p* n-torne promenljive, *a* neka konstanta i *Θ* operacija poređenja, onda oznake *s[i] Θ p[j]* i *s[i] Θ a* predstavljaju poređenje i-te komponente promenljive *s* sa j-tom komponentom promenljive *p* ili sa konstantom *a*, redom, i nazivaju se atomi poređenja.
- (f1) Svaki atom je formula i sve n-torne promenljive atoma su slobodne u formuli.
- (f2) Ako su f i g formule, onda su formule i: f AND g, f OR g, NOT f – pojavljivanje promenljive u formuli je tačno onakvo kakvo je u f i g.
- (f3) Ako je f formula, onda su formule i: (∃s)(f) i (∀s)(f). Promenljiva s je vezana u toj formuli, a slobodna u f.
- (f4) Ako je f formula, onda je formula i (f).
- (f5) Ništa drugo nije formula relacionog računa n-torki.

Primer: {(x[3], x[4], x[6]) : RADNIK(x) AND x[7] >= 40000} – imena, prezimena i datumi zaposlenih sa zaradom bar 40000. Za rad na računaru se koristi prilagođena sintaksa, pa bi odgovarajući upit u tom obliku bio: radnik.ime, radnik.prezime, radnik.dat_zaposlenja WHERE radnik.osnova_plate >= 40000. Ovakav oblik ima dosta sličnosti sa SQL-om.

112. Šta je relacioni račun domena? Objasniti osnovne karakteristike.

Relacioni račun domena je vrsta relacionog računa (formalnog apstraktnog jezika za zadavanje upita). Za razliku od relacionog računa torki, ovde se promenljive vezuju za domene atributa, a ne za domene torki. Izraz relacionog računa domena je $\{x_1, ..., x_k \mid f\}$, gde su:

- Promenljive $x_1, ... , x_k$ domenske promenljive.
- f je formula relacionog računa domena čije su slobodne promenljive $x_1, ..., x_k$.

Izrazi relacionog računa domena se grade na sledeći način:

- (a1) Ako je relacija *R* stepena n (tj. ima n atributa) i $y_1, ..., y_n$ konstante ili domenske promenljive nad domenima odgovarajućih atributa relacije *R*, onda je $R(y_1, ..., y_n)$ atom čije je značenje da vrednosti $y_1, ..., y_n$ moraju biti takve da postoji odgovarajuća n-torka u relaciji *R*.
- (a2) Ako su *x* i y promenljive, *a* neka konstanta i *Θ* operacija poređenja, onda su *x Θ y* i *x Θ a* atomi poređenja i značenje im je da vrednosti x i y moraju da budu takve da je ovakvo poređenje tačno.
- (f1) – (f5) kao kod relacionog računa n-torki. (pitanje 111.)

Primer upita: {zqs | (∃x)(∃y)(∃w)(∃t) RADNIK(xyzqwst) AND t >= 40000} – imena (z), prezimena (q) i datumi (s) svih radnika čija je zarada bar 40000.

113. U čemu je značaj formalnih upitnih jezika?

Sa teorijske strane, značaj formalizovanja upitnih jezika je u mogućnosti dokazivanja da se jednim takvim jezikom može iskazati bilo kakav upit. Međutim, sa praktične strane, dodatan značaj je u mogućnosti optimizacije upita – analizom upita se pronalazi što efikasniji put za njegovo izračunavanje, a korektnost te optimizacije je moguće formalno dokazati.

114. Kako se formalizuju operacije ažuriranja sadržaja baze podataka?

Da bi se formalizovale operacije ažuriranja, potrebno je razdvojiti pojam sadržaja relacije i relacione promenljive. Naime, relacionu promenljivu čini sve što nije sadržaj (npr. shema relacije), te se ažuriranjem sadržaja ona ne menja. Sa druge strane, sadržaj relacije će biti promenjen. Isto tako, ovi pojmovi se mogu proširiti i na čitavu bazu podataka u vidu promenljive baze podataka (niz relacionih promenljivih) i sadržaja baze podataka (sadržaj promenljive baze podataka). Nakon ovog razdvajanja, ažuriranje je moguće definisati kao zamenjivanje vrednosti, tj. sadržaja, baze podataka sa novom vrednošću (promenljivost se onda apstrahuje kao sekvenca takvih transformacija, tj. novih vrednosti).

115. Šta su integritet i konzistentnost? Zašto su značajni za baze podataka?

Konzistentnost baze je opštiji pojam od integriteta. Naime, konzistentnost se odnosi na saglasnost sadržaja u bazi podataka u odnosu na stvarni prostor koji se modelira tom bazom (tačnost ili ispravnost podataka u bazi). Na primer, ukoliko je u bazu potrebno uneti podatke o osobi koja se zove Ivan, ali je greškom uneseno ime Jovan, u tom slučaju je narušena konzistentnost baze.

Sa druge strane, integritet predstavlja one aspekte konzistentnosti koji se mogu automatski proveriti, tj. predstavlja logičku ispravnost sadržaja baze podataka. Na primer, Ivan ne može polagati ispit na predmetu koji nije prethodno upisao – u suprotnom, integritet bi bio narušen.

Ovi pojmovi su značajni jer, ukoliko baza ne opisuje ispravno stvarni prostor koji bi trebalo da modelira, upitno je koliko je ta baza uopšte i korisna.

Napomena – dodatni vid konzistentnosti – transakciona konzistentnost...

116. Šta čini integritetni deo relacionog modela?

Integritetni deo relacionog modela čine svi koncepti i mehanizmi koji omogućavaju automatsku proveru zadovoljivosti određenih uslova, tj. proveru logičke ispravnosti sadržaja baze. Ovaj deo zapravo čini jednu od primarnih prednosti relacionog modela u odnosu na neke druge nerelacione modele, u kojima je automatska provera integriteta dosta slabija, ili čak nepostojeća.

Ovi uslovi se modeliraju predikatima (istinitosnim formulama) nad relacijama (tabelama) u bazi.

117. Navesti osnovne vrste opštih uslova integriteta u relacionoj bazi podataka.

Opšti uslovi integriteta se odnose na one uslove koji se, suštinski, javljaju u svakoj bazi podataka (u nekom parametrizovanom obliku), tj. ne definišu se eksplicitno za konkretnu relaciju ili atribut, već se, pri određivanju strukture podataka, određuju konkretni parametri tih uslova.

Osnovne vrste opštih uslova integriteta su integritet domena, integritet primarnog ključa i integritet stranog ključa.

118. Navesti osnovne vrste specifičnih uslova integriteta u relacionoj bazi podataka.

U zavisnosti od opsega podataka na koji se postavlja dati specifični uslov integriteta, imamo specifične uslove integriteta na atributu, torki, relaciji ili bazi podataka.

119. Objasniti integritet domena u relacionom modelu.

Integritet domena nalaže da svaka relacija, kao i svaki njen atribut, moraju imati tačno određen domen (koji je, dodatno, u slučaju atributa, skalaran) – u relaciji se ne smeju naći vrednosti koje ne pripadaju njenom domenu.

U praksi, ovo podrazumeva zadavanje tipova atributa (tip podatka, dužinu podatka, opcionu deklaraciju jedinstvenosti i deklaraciju podrazumevane vrednosti, mogućnost nedefinisane vrednosti) u okviru svake relacije (tabele) baze podataka.

```
CREATE TABLE <ime tabele> (
        ...
        <ime kolone> <tip kolone> ...,
        ...
)
```

120. Objasniti integritet primarnog ključa u relacionom modelu.

Integritet primarnog ključa nalaže da svaka relacija ima bar jedan ključ (skup atributa K koji jedinstveno određuje svaku torku relacije (natključ) i za koji ne postoji pravi podskup koji je takođe natključ).

U suštini, za svaku relaciju mora da postoji tačno jedan ovakav ključ – tzv. primarni ključ. Potreba za ovim uslovom je u tome što bi, u suprotnom, duplikati u relaciji bili mogući, a to ne bi smelo da se desi, jer relacija je skup.

```
CREATE TABLE <ime tabele> (
        ...
        [CONSTRAINT <ime ključa>]
                PRIMARY KEY (<lista imena kolona>) ...
        ...
) -- napomena: u praksi se ime primarnog ključa retko kad zadaje, jer svakako postoji tačno jedan.
```

121. Objasniti integritet jedinstvenog ključa u relacionom modelu.

Integritet jedinstvenog ključa nalaže da izabrani jedinstveni ključ treba da jedinstveno određuje torke relacije. Za razliku od primarnog ključa, jedinstvenih ključeva može biti više. Takođe, u nekim implementacijama moguće je da jedinstveni ključ sadrži nepoznate vrednosti (neke ne nude ovu mogućnost direktno, ali omogućavaju zaobilazni put definisanjem indeksa koji, uz indeksiranje, obavlja i proveru jedinstvenosti).

```
CREATE TABLE <ime tabele> (
        ...
        [CONSTRAINT <ime ključa>]
                UNIQUE (<lista imena kolona>) ...
        ...
)
```

122. Objasniti referencijalni integritet u relacionom modelu.

Referencijalni integritet se odnosi na uslove o međusobnim odnosima koje torke dve relacije moraju zadovoljavati. Naime, ukoliko se zanemare nepoznate vrednosti, postoje dva uslova:

- Ne sme se obrisati torka na koju se odnosi neka torka neke relacije u bazi podataka, niti da se izmeni na takav način da ta referenca postane nevalidna.
- Ne sme se dodati torka sa neispravnom referencom, tj. referencom na neku drugu torku koja ne postoji.

Dodatno, ukoliko se u obzir uzmu i nedefinisane vrednosti, dodaje se i sledeći uslov:

- Referenca koja sadrži nedefinisane vrednosti je ispravna (i ne referiše ni na šta) akko je u potpunosti nedefinisana, tj. svi njeni atributi su nedefinisane vrednosti.

Međutim, većina implementacija ipak dozvoljava da su samo neki atributi nedefinisane vrednosti, i tada se pretpostavlja da ta referenca ne referiše ni na šta.

123. Objasniti integritet stranog ključa u relacionom modelu.

Integritetetom stranog ključa se ostvaruje referencijalni integritet. Skup atributa relacije R, u oznaci FK, je njen strani ključ koji se odnosi na baznu relaciju B akko važe sledeći uslovi (ne uzimamo u obzir nedefinisane vrednosti):

- Relacija B ima primarni ključ PK.
- Domen ključa FK je identičan domenu ključa PK.
- Svaka vrednost stranog ključa FK u torkama relacije R je identična ključu PK bar jedne torke relacije B.

U ovom slučaju, za relaciju R se kaže da je zavisna od bazne relacije B, njene roditeljske relacije.

Ukoliko se u obzir uzmu i nedefinisane vrednosti, tada se koriste sledeći uslovi:

- Relacija B ima primarni ključ PK.
- Domen ključa FK je identičan domenu ključa PK, ili je proširen nedefinisanim vrednostima.
- Svaka vrednost stranog ključa FK u torkama relacije R je ili nedefinisana ili je identična ključu PK bar jedne torke relacije B.
- Vrednost stranog ključa FK u torkama relacije R je nedefinisana akko sve vrednosti atributa stranog ključa imaju nedefinisanu vrednost (mada je u realnim implementacijama ovo relaksirano, slično kao i kod referencijalnog integriteta, da je dovoljno da bar jedna vrednost bude nedefinisana da bi ceo strani ključ FK bio nedefinisan).

124. Objasniti pravila brisanja i ažuriranja kod integriteta stranog ključa u relacionom modelu.

Postoji više vrsta pravila brisanja, ali bira se samo jedno. Ovo pravilo se "pokreće" u slučaju pokušaja brisanja torke bazne relacije B za koju postoji zavisna torka relacije R (koja ima strani ključ koji odgovara nekoj torki iz B). Slede pravila:

- Aktivna zabrana brisanja – RESTRICT – ovakvo brisanje je potpuno zabranjeno.
- Pasivna zabrana brisanja – NO ACTION – ovakvo brisanje je zabranjeno, ali se provera odlaže do samog kraja brisanja (ukoliko su neke druge reference koristile kaskadno brisanje, moguće je da usled njih dođe do brisanja torke zavisne relacije R, pa će biti moguće i brisanje torke iz bazne relacije B).
- Kaskadno brisanje – CASCADE – briše se torka iz bazne relacije B, ali se brišu i sve njoj zavisne torke iz relacije R.
- Postavljanje nedefinisanih vrednosti – SET NULL – u zavisnim torkama relacije R, vrednosti atributa odgovarajućeg stranog ključa se postavljaju na nedefinisanu vrednost.
- Postavljanje podrazumevane vrednosti – SET DEFAULT – isto kao i kod postavljanja nedefinisanih vrednosti, samo što se umesto NULL postavlja neka podrazumevana vrednost.

Pravila ažuriranja se "pokreću" u slučaju pokušaja izmene primarnog ključa torke bazne relacije B za koju postoji zavisna torka relacije R. Pravila su suštinski ista kao i kod brisanja (samo što se sada torke ne brišu, već ažuriraju).

```
CREATE TABLE <ime tabele> (
        ...
        [CONSTRAINT <ime ključa>]
                FOREIGN KEY (<lista imena kolona>)
                REFERENCES <ime tabele> [(<lista imena kolona>)]
                [ON DELETE <pravilo brisanja>]
                [ON UPDATE <pravilo ažuriranja>]
        ...
)
```

125. Objasniti specifične uslove integriteta relacionog modela.

Specifični uslovi integriteta se ostvaruju eksplicitnim navođenjem odgovarajućih logičkih uslova na nivou pojedinačnih atributa, torki, relacija ili čitave baze podataka.

126. Šta su uslovi integriteta na atributu i torki?

Uslovi integriteta na atributu su lokalni uslovi koji se zadaju nad vrednostima nekog konkretnog atributa (neke relacije). Često se koriste kao dopuna opštem uslovu domena. Na primer, za godinu rođenja radnika, čiji je domen skup celih brojeva, može se zadati uslov da je u nekom konkretnom opsegu (npr. od 1980 do 2100). U ovakvim uslovima, sme se referisati samo na taj jedan atribut za koji se definišu.

```
CREATE TABLE <ime tabele> (
        ...
        <ime atributa> <tip atributa>
                CHECK (<logički uslov>)
        ...
)
```

Uslovi integriteta na torkama se odnose na vrednosti jedne torke. Uglavnom se koriste za proveru vrednosti atributa koji bi trebalo da budu na neki način međusobno zavisni – na primer, ukoliko imamo atribut za godinu rođenja i atribut za jmbg, ovakvim uslovom bi mogli da proveravamo da li određena polja jmbg-a odgovaraju godini rođenja.

```
CREATE TABLE <ime tabele> (
        ...
        CONSTRAINT [<ime uslova>]
                CHECK (<logički uslov>)
        ...
)
```

127. Šta su uslovi integriteta na relaciji i bazi podataka?

Uslovi integriteta relacije se odnose na sve torke jedne relacije (mogu se koristiti svi atributi te relacije). Oni se koriste za proveru ispravnosti složenijih saglasnosti u okviru jedne relacije – na primer, provera da li je broj radnika u nekoj kompaniji manji od nekog gornjeg kapaciteta.

```
CREATE TABLE <ime tabele> (
        ...
        CONSTRAINT [<ime uslova>]
                CHECK (<logički izraz sa podupitima>)
        ...
)
```

Uslovi integriteta na bazi podataka se odnose na čitav sadržaj baze podataka, tj. na sadržaj različitih relacija. U praksi, oni se najčešće ne implementiraju zbog neefikasnosti (zamislimo da se nakon svake naredbe proverava da li je broj redova neke relacije veći od broja redova neke druge relacije, ili slično).

```
CREATE ASSERTION <ime> CHECK (<logički izraz sa podupitima>)
```

128. Kada se proveravaju uslovi integriteta? Da li i kako to može da se promeni?

Uslovi integriteta se, po podrazumevanom načinu rada, proveravaju pri svakom pojedinačnom pokušaju promene sadržaja baze podataka. Međutim, nekada je potrebno da se oni proveravaju tek na kraju transakcije, što veliki broj RSUBP omogućava naredbom SET CONSTRAINTS ALL DEFERRED, kojom se sve provere u tekućoj transakciji odlažu do njenog završetka (naravno, veći broj opcija je moguć za ovu naredbu, kao npr. izbor tabela i uslova koji se odlažu).

129. Objasniti aktivno održavanje integriteta u relacionim bazama podataka.

Aktivno održavanje integriteta se odnosi na njegovo održavanje reagovanjem na određene promene tako što će se pokrenuti eksplicitno definisan programski kod koji će praviti neophodna usklađivanja podataka. Mehanizam za aktivno održavanje integriteta su okidači (*triggers*).

130. Objasniti ulogu i princip rada okidača na tabelama relacione baze podataka.

Uloga okidača jeste da pruže mogućnost aktivnog održavanja integriteta u relacionoj bazi podataka. Oni su definisani relacijom na kojoj prate promene, vrstom promene na koju treba da reaguju, programskim kodom koji treba da izvrše, trenutkom izvršavanja (pre ili posle promene), granularnošću izvršavanja (da li će se kod izvršavati po jednom za svaku promenjenu torku ili jednom za čitavu naredbu menjanja) i podacima koji se referišu u datom programskom kodu.

U suštini, okidač će pratiti određene promene na nekoj tabeli i izvršavati neki zadati kod kada ta promena nastane.

Programski kod koji se izvršava ne mora striktno vršiti samo proveru integriteta. On takođe može da radi na održavanju ispravnih vrednosti podataka (dakle umesto samo provere integriteta, mi i ispravljamo narušenu vrednost – ovo je najčešće loša ideja i signal da u bazi postoje redundantni podaci, ali postoje neke situacije kada se može opravdati zbog dobitka na performansama, npr. umesto konstantnog računanja prosečne ocene na nekom predmetu, možemo imati kolonu za nju i onda da je održavati okidačima).

U slučaju samo provere integriteta, ukoliko se on naruši, biva prijavljena greška – ovo, međutim, iako se često koristi, često i nije dobra ideja zbog transakcionog rada (jedna naredba unutar transakcije može narušiti integritet i aktivirati okidač, ali celovita transakcija ne mora nužno narušiti integritet – primer, prebacivanje novca sa jednog računa na drugi).

Dodatno, i najzanimljivije, programski kod može izvoditi i akcije van same baze podataka – na primer, slanje mejla za naručivanje ukoliko je stanje nekog proizvoda u prodavnici nisko.

131. Objasniti ulogu i princip rada okidača na pogledima relacione baze podataka.

Pogledi (upiti – virtuelne relacije), kao glavno sredstvo razdvajanja spoljašnjeg i konceptualnog nivoa relacionog modela, imaju jedno veliko ograničenje – ukoliko su definisani nad više relacija, ili ukoliko koriste kolonske funkcije, oni se ne mogu ažurirati. Okidači nad pogledima prevazilaze taj problem. Naime, ovi okidači se izvršavaju umesto naredbi za dodavanje, brisanje ili menjanje nad pogledom. Ovo omogućava preusmeravanje izmena sa pogleda na stvarne relacije koje učestvuju u njemu, kao i sakrivanje potencijalno veoma složenih operacija od korisnika spoljašnje sheme.

132. U čemu je razlika između logičkog i konceptualnog modela baze podataka?

Za razliku od logičkog, konceptualni model baze podataka je potpuno posvećen domenu koji se modelira, tj. njegovoj semantici, dok logički model u obzir uzima konkretan model podataka koji će se koristiti u implementaciji baze (npr. relacioni).

Primer razlike između konceptualnog i logičkog modela bi bio to što se u konceptualnom modelu ne navode surogat ključevi i strani ključevi, dok je u logičkom modelu to obavezno.

133. Šta je logički model baze podataka?

Logički model baze podataka predstavlja opis strukture podataka koji će se čuvati u bazi podataka i njihovih odnosa, pritom uzimajući u obzir neki konkretan model podataka (npr. relacioni).

Ovo u suštini obuhvata logičku shemu svih trajnih objekata (relacija u slučaju relacionog modela podataka), specifikaciju svih uslova i ograničenja (što obuhvata i uslove stranih ključeva), sve ključeve i surogat atribute, kao i način implementacije svih odnosa (napomena da, zavisno od modela podataka, pa u nekim situacijama i od konkretnog SUBP-a, način implementacije odnosa se može razlikovati).

134. U čemu je osnovna razlika između konceptualnog i logičkog modela (i modeliranja)?

Isto kao 132.

135. Šta je očekivani rezultat logičkog modeliranja?

Očekivani rezultat logičkog modeliranja je detaljan logički model koji predstavlja logičku shemu svih trajnih objekata (relacija u slučaju relacionog modela podataka), specifikaciju svih uslova i ograničenja (što obuhvata i uslove stranih ključeva), sve ključeve i surogat atribute, kao i način implementacije svih odnosa (napomena da, zavisno od modela podataka, pa u nekim situacijama i od konkretnog SUBP-a, način implementacije odnosa se može razlikovati).

136. Zašto nije dobro preskočiti logički model i praviti fizički model na osnovu konceptualnog?

Iako je ovo moguće izvesti u nekim situacijama (manja baza podataka, SUBP koji će se koristiti poznat unapred, ...), preskakanje logičkog modela može dovesti do neželjenih posledica u vidu gubitka kompletnosti, integriteta, fleksibilnosti, efikasnosti i upotrebljivosti kod finalne implementacije baze.

137. Kako teče postupak pravljenja logičkog modela?

Pravljenje logičkog modela je iterativan proces. Naime, prva iteracija se dobija prevođenjem konceptualnog modela u logički, dok se svaka naredna pravi prečišćavanjem prethodne sheme, tj. izmenama prethodne iteracije, sve dok se ne zadovolje određene karakteristike koje logički model treba da poseduje.

138. Na osnovu kojih kriterijuma se pristupa menjanju logičkog modela?

Kriterijumi na osnovu kojih se vrši iterativni postupak logičkog modeliranja, tj. prečišćavanja sheme, su:

- Ispunjenost funkcionalnih zahteva modela (da li su prevedeni svi entiteti i odnosi)
- Ispunjenost nefunkcionalnih zahteva modela (više se odnose na implementaciju)
- Kompletnost modela
- Garancija integriteta podataka modelom
- Fleksibilnost modela
- Efikasnost modela
- Upotrebljivost modela
- Usklađenost modela sa ciljnim modelom podataka (npr. relacionim)

139. Kakva je izražajnost logičkog modela u odnosu na konceptualni? Šta može da se vidi u konceptualnom modelu a obično ne može u logičkom?

Konceptualni model bi trebalo da ima bolju izražajnost semantike entiteta i odnosa samog domena koji se modelira. Semantika odnosa se teže prepoznaje u logičkom modelu. Tako, na primer, u relacionom (logičkom) modelu, i entitete i odnose modeliramo relacijama (dodatno, odnose modeliramo i stranim ključevima), te se ponekad ne može ni jasno videti razlika između entiteta i odnosa.

140. Šta su "dijagrami tabela (relacija)"? Kada se koriste?

Dijagrami tabela su dijagramski alat / tehnika za projektovanje koji je posebno prilagođen za projektovanje na logičkom i fizičkom nivou relacionog modela. Kao što je već pomenuto, primarno se koristi za logičko i fizičko projektovanje kod relacionog modela.

141. U čemu se dijagrami tabela suštinski razlikuju od ER dijagrama? Koji se kada koriste?

Suštinska razlika je u tome što dijagrami tabela ne čuvaju dovoljno semantičkih informacija (o entitetima i odnosima između njih u stvarnom sistemu), pa su, iz tog razloga, ER dijagrami pogodniji za upotrebu kod konceptualnog projektovanja, dok se dijagrami tabela više koriste za logičko i fizičko projektovanje kod relacionog modela. Dodatno, ni oznake nisu iste – u dijagramima tabela koristimo termine poput tabela, kolona i ključeva, a i kardinalnosti se drugačije označavaju (po uzoru na UML, suprotna strana od ER dijagrama). Pored ovoga, nema oznaka za hijerarhije i agregacije.

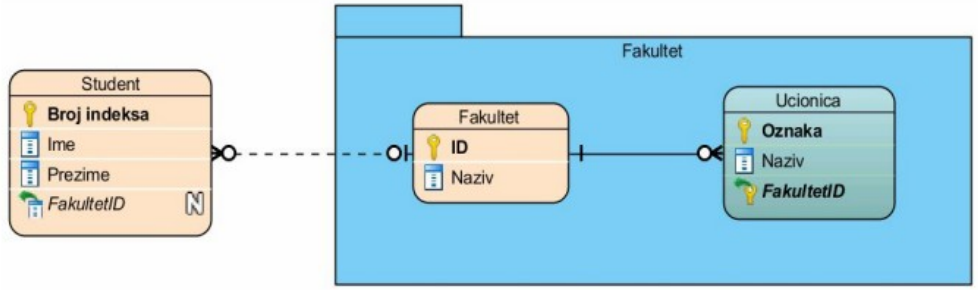
142. Objasniti osnovne elemente dijagrama tabela.

Tabele su predstavljene (zaobljenim) pravougaonicima koji su podeljeni na dva dela horizontalnom linijom – u gornjem delu se navodi naziv, dok se u donjem navode opisi kolona (i eventualno još neke informacije).

Kolone koje pripadaju primarnom ključu se mogu označavati na različite načine, npr. oznakom PK, crtežom ključa ili simbolom +. Kolone koje pripadaju stranom ključu se, slično, označavaju oznakom FK, crtežom strelice ili simbolom #. Dodatno, ukoliko kolona takođe pripada primarnom ključu, može se, na primer, označiti sa PF (PKFK). Dodatno, ukoliko kolona može da sadrži nedefinisanu vrednost, može se dodati simbol koji asocira na slovo N.

Odnosi između tabela (gde je osnovna vrsta odnosa u vidu stranog ključa) se označavaju linijama – standardno punim, a isprekidanim ukoliko se označava odnos slabog entiteta sa jakim. Kardinalnosti se označavaju u duhu UML-a, tj. na strani entiteta (tabele) stoji broj tih entiteta koji mogu biti u odnosu sa jednim entitetom sa druge strane odnosa. Ove kardinalnosti se označavaju grafičkim oznakama – 0 sa kružićem, 1 sa upravnom linijom, * sa tri kratke linije prema tabeli, a ostale kardinalnosti se izražavaju kombinacijom ovih simbola.

Takođe, tabele se mogu grupisati (slično kao paketi u UML-u). Najčešće se grupišu tabele koje čine neku logičku celinu.



143. Objasniti dodatne elemente dijagrama tabela.

Dodatni elementi u dijagramima tabela se mogu koristiti za preciznije opisivanje modela. Oni se najviše koriste za projektovanje na fizičkom nivou. Neki dodatni elementi koji se mogu prikazati su:

- Pogledi (mogu se označiti, na primer, <<view>> stereotipom u delu za naziv)
- Okidači
- Uslovi integriteta (najčešće prikazani u vidu komentara)
- Prostori tabela i drugi elementi fizičkog modela
- Indeksi
- Ugrađene procedure
- ...

144. Kako se označavaju ključevi u dijagramima tabela?

Kolone koje pripadaju primarnom ključu se mogu označavati na različite načine, npr. oznakom PK, crtežom ključa ili simbolom +. Kolone koje pripadaju stranom ključu se, slično, označavaju oznakom FK, crtežom strelice ili simbolom #. Dodatno, ukoliko kolona takođe pripada primarnom ključu, može se, na primer, označiti sa PF (PKFK).

145. Kako se označavaju različite vrste odnosa i kardinalnosti odnosa u dijagramima tabela?

Odnosi između tabela (gde je osnovna vrsta odnosa u vidu stranog ključa) se označavaju linijama – standardno punim, a isprekidanim ukoliko se označava odnos slabog entiteta sa jakim. Kardinalnosti se označavaju u duhu UML-a, tj. na strani entiteta (tabele) stoji broj tih entiteta koji mogu biti u odnosu sa jednim entitetom sa druge strane odnosa. Ove kardinalnosti se označavaju grafičkim oznakama – 0 sa kružićem, 1 sa upravnom linijom, * sa tri kratke linije prema tabeli, a ostale kardinalnosti se izražavaju kombinacijom ovih simbola.

146. Objasniti potencijalne razlike u dijagramima tabela na konceptualnom / logičkom / fizičkom nivou.

Na konceptualnom nivou se upotreba dijagrama tabela uglavnom ne preporučuje, a ako se ipak koristi, onda je to najčešće za veoma jednostavne baze podataka. U ovom slučaju, dijagrami su pojednostavljeni, tj. sadrže samo osnovnu strukturu podataka – ne sadrže dodatne kolone za surogat ključeve, za strane ključeve, oznake ključeva, tipove kolona i slično. Dodatno, na ovom nivou se mogu predstaviti više-na-više veze (što, naravno, zbog relacionog modela, nije moguće predstaviti direktno na logičkom i fizičkom nivou).

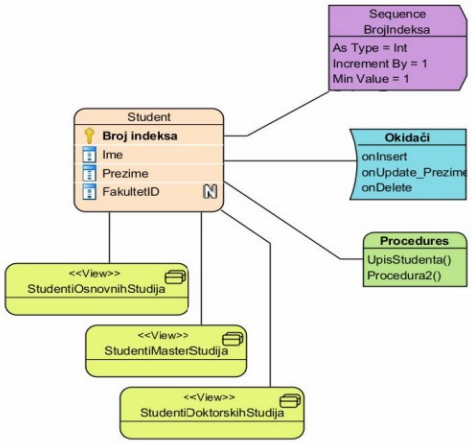
Na logičkom nivou se uglavnom koriste (sa eventualnim manjim dodacima) dijagrami tabela u svom osnovnom obliku (sve kolone označene, označeni svi ključevi, tipovi kolona, odnosi više-na-više se zamenjuju dodatnom tabelom, ...).

Na fizičkom nivou se prikazuju i dodatni elementi – pogledi, indeksi, okidači, prostori za tabele...

147. Kako se u dijagramima tabela označavaju pogledi, okidači i drugi elementi?

Označavaju se, zavisno od elementa, različitim oblicima (mada opet slično i tabelama – oblik približan pravougaoniku podeljen na deo za naziv i za delove elementa koje je potrebno prikazati) ili dodatnim oznakama (stereotipovima, npr. <<view>> za poglede ili grafičkim oznakama).

Ovi elementi se, najčešće, punom linijom povezuju sa tabelom (tabelama) na koju se odnose, tj. koju koriste.



148. Kako se konceptualni model "prevodi" u logički?

Tačan postupak prevođenja zavisi od konkretnih tehnika izražavanja ovih modela (npr. ER model ili UML), izbora modela podataka za logički nivo (npr. relacioni) kao i od same semantike i konteksta koji se javljaju u konceptualnom modelu. Suštinski cilj ovog koraka je prebacivanje entiteta i odnosa na jezik izabranog logičkog modela, ali na taj način da oni i dalje poštuju semantiku koja je bila opisana konceptualnim modelom.

Uglavnom važi da su neki aspekti (najčešće većina njih) prevođenja relativno jednostavni i pravolinijski (npr. svaki entitet se prevodi u relaciju ukoliko koristimo relacioni model), te se za njih većim delom mogu koristiti ustanovljena pravila prevođenja. Međutim, postoje i složeniji slučajevi koji se ne mogu prevesti tako jednostavno, ali je njihovo ispravno prevođenje od presudne važnosti za rezultujući model.

149. Opisati postupak prevođenja konceptualnog modela u logički model u slučaju relacionog modela podataka.

Postupak prevođenja konceptualnog modela u logički model sa relacionim modelom podataka se odvija u dve faze:

- Prevođenje entiteta – entiteti iz konceptualnog modela se prevode u relacije
  - Skalarni atributi tih entiteta se prevode u atribute odgovarajućih relacija
  - Složeni atributi se prevode u više atributa relacije (u nekim situacijama je bolje prevesti ih u zasebne relacije, ali to se najčešće ostavlja za naredne iteracije kreiranja logičkog modela)
  - Atributi sa više vrednosti (npr. lista upisanih predmeta) se prevode u relacije
  - Dodatno, odnosi 1-* i 0..1-* se mogu direktno modelirati kao strani ključevi odgovarajućih relacija već u ovoj fazi
- Prevođenje (ostalih) odnosa – prevode se odnosi koji nisu trivijalno rešeni stranim ključevima u prethodnom koraku
  - Prvo se prevode odnosi koji kao proizvod imaju nove relacije (složeni odnosi, tj. oni sa više od dva učesnika, kao i binarni odnosi *-*)
  - Nakon toga, prevode se preostali odnosi i vrši se doterivanje

150. Kako se entiteti konceptualnog modela prevode u logički model (RM)?

Svaki entitet konceptualnog modela se prevodi u relaciju. Prilikom prevođenja, potrebno je prepoznati primarni ključ (ili, ukoliko ne postoji prirodni primarni ključ, uvesti surogat ključ), kao i prevesti ostale atribute. Ovo uključuje i elemente koji pripadaju hijerarhijama, mada tu postoji više taktika (svaki element u posebnu relaciju, svaki list u posebnu relaciju ili čak i cela hijerarhija u jednu relaciju).

Dodatno, slabi entiteti i odnosi u kojima učestvuju se najčešće modeliraju jednom relacijom koja sadrži strani ključ prema jakom entitetu sa kojim je u odnosu.

151. Kako se atributi konceptualnog modela prevode u logički model (RM)?

Skalarni atributi tih entiteta se prevode u atribute odgovarajućih relacija, a složeni se prevode u više atributa te relacije (svaka komponenta u zaseban atribut, mada je u nekim situacijama bolje prevesti ih u zasebne relacije, ali to se najčešće ostavlja za naredne iteracije kreiranja logičkog modela).

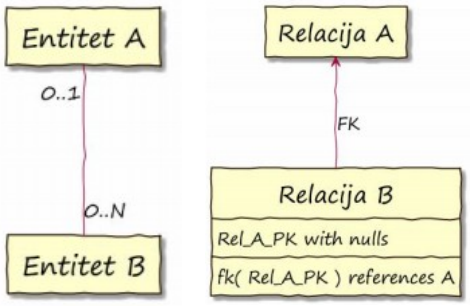
Sa druge strane, atributi sa više vrednosti (npr. lista upisanih predmeta) se prevode u relacije.

152. Kako se odnosi konceptualnog modela prevode u logički model (RM)?

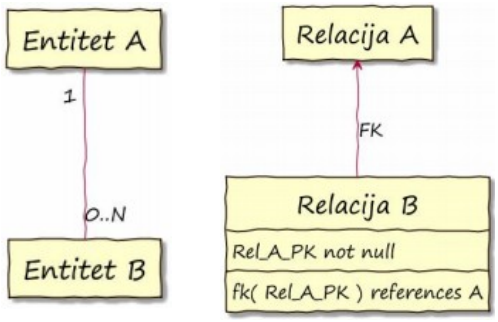
Zavisno od vrste odnosa (da li je u pitanju složen odnos, generalizacija, ...) i njegove kardinalnosti, odnosi se prevode ili u strane ključeve (npr. u slučaju binarnih 0-* i 1-* odnosa) ili u nove relacije (npr. u slučaju binarnih *-* odnosa), gde je u nekim situacijama potrebno i postavljanje dodatnih uslova integriteta. Dodatno, u izboru tačnog načina prevođenja može učestvovati i konkretan SUBP koji će se koristiti, jer neki podržavaju određene načine prevođenja, a drugi ne (detaljno objašnjeno u narednim odgovorima).

153. Kako se u logičkom (relacionom) modelu modeliraju odnosi 0..1 – 0..*, 1 – 0..*, 0..1 – 1, 0..1 – 0..1?

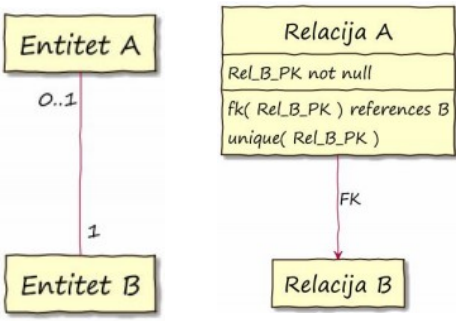
*0..1 – 0..** – ovo je najčešći slučaj kod odnosa agregacije, jednostavno se modelira stranim ključem (koji može imati NULL vrednost) prema celini na strani entiteta koji predstavlja deo.



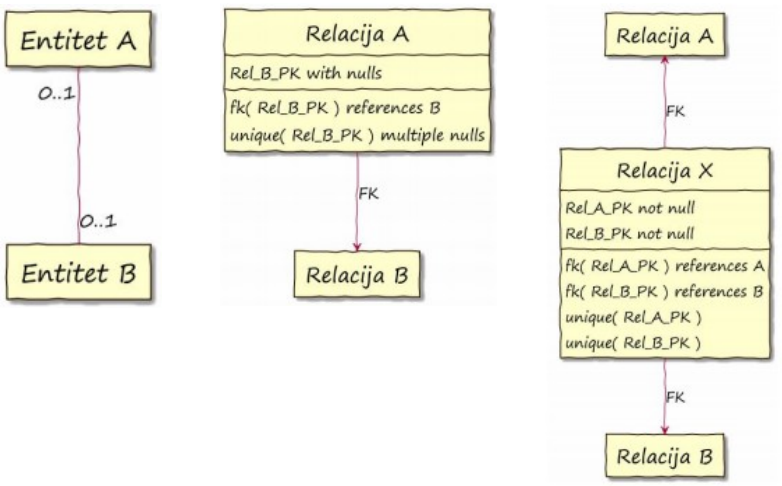
*1 – 0..** – najčešći slučaj kod odnosa kompozicije, suštinski se modelira isto kao i agregacija, samo što strani ključ ne može imati NULL vrednost.



0..1 – 1 – tipična kardinalnost za odnose koji imaju semantiku "nadređeni – podređeni" ili "celina – opcioni deo". Primer bi bilo odeljenje (entitet A) kojim mora da rukovodi neki službenik (entitet B), ali taj službenik ne mora da bude rukovodilac, a takođe može rukovoditi i najviše jednim odeljenjem. U tom slučaju, odnos se modelira dodavanjem JEDINSTVENOG stranog ključa na strani odeljenja koji referiše na nekog službenika i koji ne može biti NULL. Uslov jedinstvenosti je potreban da bi jedan službenik rukovodio najviše jednim odeljenjem.

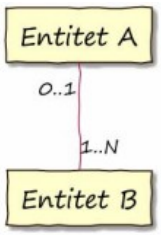


0..1 – 0..1 – kardinalnost koja predstavlja dvosmerni opcioni odnos. Primer bi bio projekat (entitet A) koji ima najviše jednog zaposlenog (entitet B) kao vođu projekta, s tim što zaposleni (entitet B) ne mora biti vođa ni jednog projekta, a može biti vođa najviše jednog. Ovaj odnos se može modelirati na dva standardna načina – ukoliko je u SUBP-u podržan uslov jedinstvenosti sa višestrukim nedefinisanim vrednostima (sve vrednosti, osim ukoliko su NULL, se mogu javiti najviše jednom), onda se može modelirati dodavanjem JEDINSTVENOG stranog ključa sa dozvoljenim NULL vrednostima na jednu od strana (koji referiše na drugu stranu). Ukoliko SUBP to ne podržava, opštije rešenje je uvođenje nove, vezne, relacije (u ovom slučaju, strani ključevi unutar vezne relacije su jedinstveni i ne smeju biti nedefinisani – ukoliko odnos ne postoji, vezna relacija jednostavno neće sadržati red koji odgovara datom odnosu).

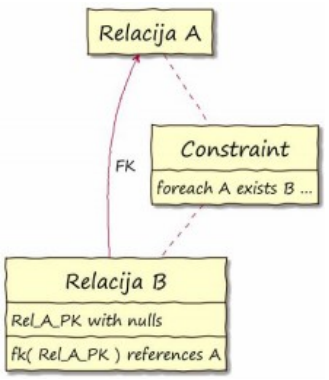


154. Kako se u logičkom (relacionom) modelu modeliraju odnosi 0..1 – 1..*, 1 – 1..*, 1 – 1?

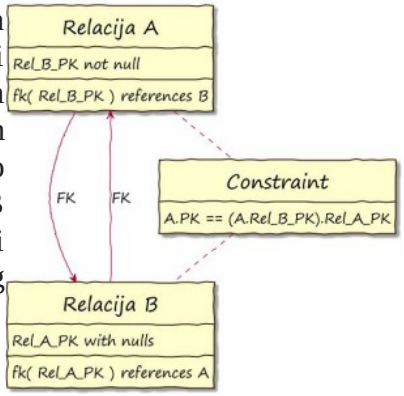
*0..1 – 1..** – ovakva kardinalnost odgovara semantici agregacije sa obaveznim delovima (delovi mogu postojati i zasebno, ali celina mora sadržati barem jedan deo). Za modeliranje ovakvog odnosa postoji više alternativa:



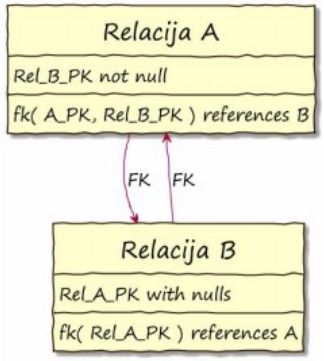
- Modeliranje kao 0..1 – 0..* sa dodatnim uslovom – relacija koja odgovara delu ima strani ključ (koji može biti NULL) prema celini, ali se dodaje i uslov da za svaku celinu A postoji deo B koji joj odgovara. Iako je ovakvo modeliranje datog odnosa jednostavno, provera datog uslova je izuzetno skupa jer zahteva prolazak kroz čitavu relaciju B.



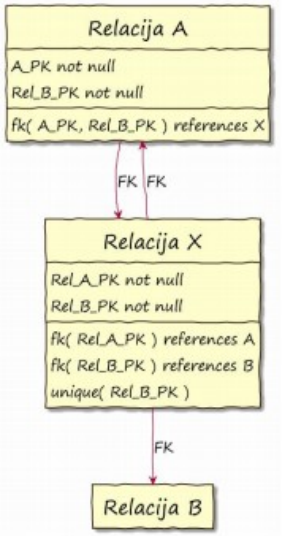
- Modeliranje sa delom predstavnikom – relacija koja odgovara delu se modelira kao i u prethodnom slučaju (strani ključ koji referiše na celinu i koji može biti NULL), a relacija koja odgovara celini sadrži obavezan strani ključ ka jednom izabranom delu B. Dodatno, da bi se osigurali da ne dođe do slučaja gde celina A pokazuje na neki izabrani deo B, a deo B pokazuje na neku drugu celinu, potrebno je dodati i dodatni uslov koji to proverava. Međutim, za razliku od prethodnog načina modeliranja, ovaj uslov je jeftiniji za proveru.



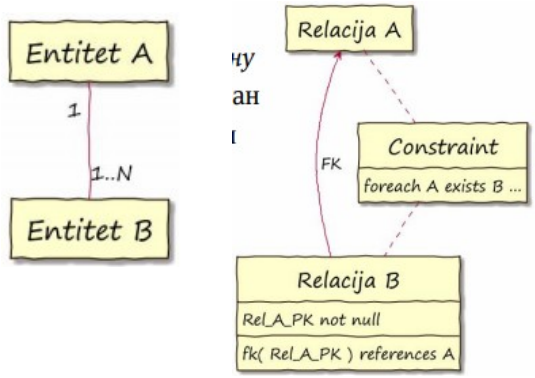
- U nekim SUBP gde je dozvoljeno referisanje po stranom ključu prema skupu atributa koji ne pripadaju primarnom ključu, uslov iz prethodnog načina modeliranja se može zameniti kompozitnim stranim ključem u relaciji celine.



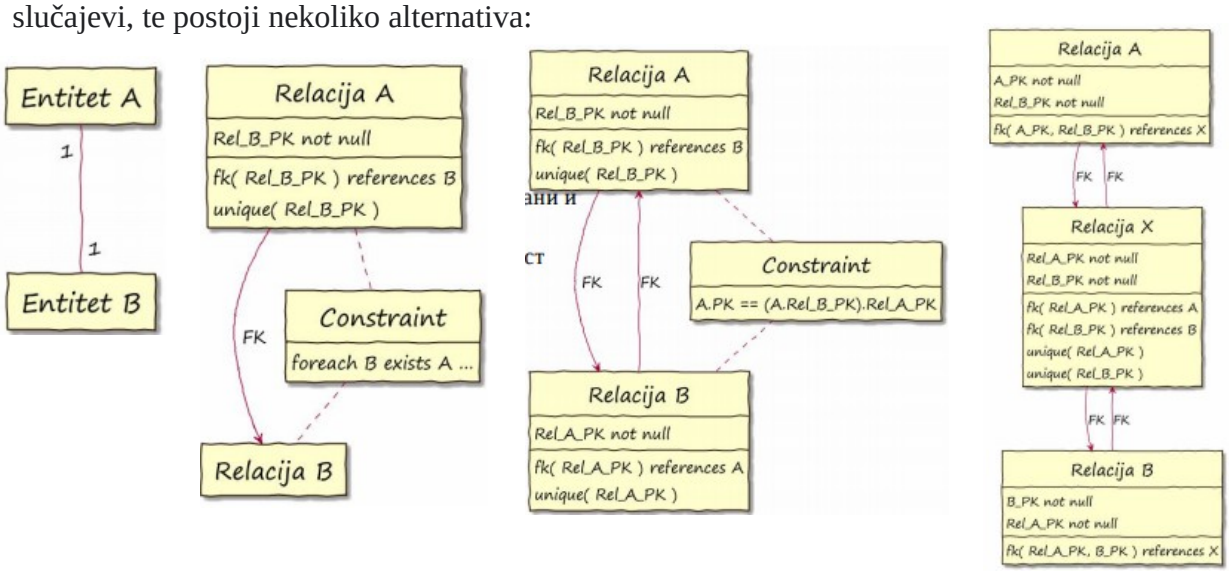
- Modeliranje sa veznom relacijom – dodatni uslovi, kao i problem stranog ključa koji referiše na atribute koji nisu deo primarnog, se mogu ukloniti uvođenjem vezne relacije. Dodavanjem vezne relacije X koja referiše na relacije A i B se dobija odnos 0..* – 0..*. Postavljanjem uslova jedinstvenosti za strani ključ prema relaciji B se dobija odnos 0..1 – 0..*. Na kraju, dodavanjem (obaveznog) stranog ključa ka veznoj relaciji u relaciju celine A dobijamo odnos 0..1 – 1..*.



*1 – 1..** – odgovara kompoziciji sa obaveznim delovima. Primer bi bilo odeljenje A koje ima najmanje jedno parking mesto B, gde svako parking mesto mora pripadati tačno jednom odeljenju. Modelira se potpuno isto kao i u slučaju 0..1 – 1..*, s tim što odgovarajući strani ključevi u relaciji B ne mogu biti nedefinisani.



1 – 1 – ovakvi odnosi predstavljaju ekskluzivno obostrano pridruživanje (pa se u ovim situacijama može postaviti i pitanje da li su to uopšte i različiti entiteti). Modelira se slično kao i prethodni slučajevi, te postoji nekoliko alternativa:

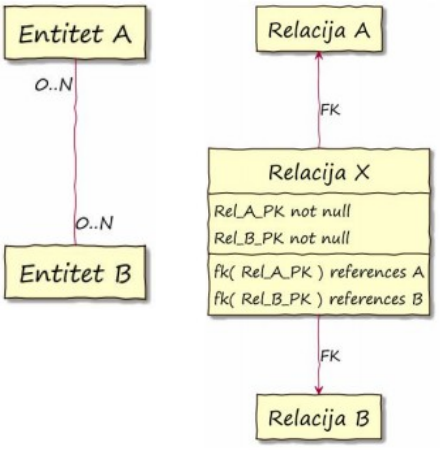


Dodatna alternativa, slična drugoj, ali kojom se izbegava uslov provere da li ključevi odgovaraju jedan drugom, jeste da se i u relaciji A i u relaciji B uvede isti primarni ključ (koji uključuje strane ključeve do druge relacije).

Važno je napomenuti da se ponekad (mada najčešće tek na fizičkom nivou) namerno vrši razbijanje entiteta na dva entiteta koja su u 1 – 1 odnosu radi povećanja efikasnosti. Naime, u slučaju da neka relacija ima ogroman broj atributa, od kojih se samo jedan manji skup najčešće koristi, taj češće korišćeni manji skup se može izvući u zasebnu relaciju i napraviti 1 – 1 odnos sa ostatkom.

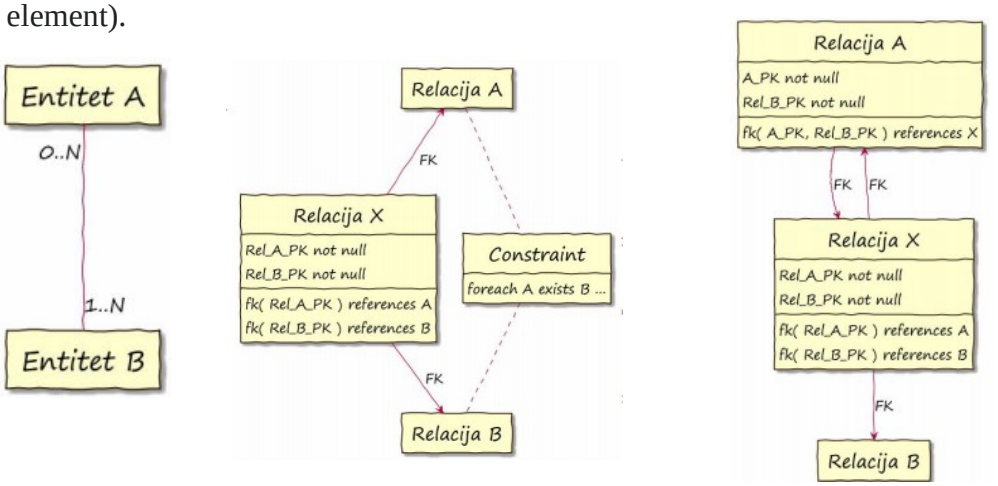
155. Kako se u logičkom (relacionom) modelu modeliraju odnosi 0..* – 0..*, 0..* – 1..*, 1..* – 1..*?

0.. – 0..** – ovakav odnos odgovara asocijaciji bez nekih dodatnih ograničenja. Modelira se veznom relacijom sa atributima koji predstavljaju strane ključeve ka relacijama A i B (i uz to su ti atributi i primarni ključ vezne relacije).

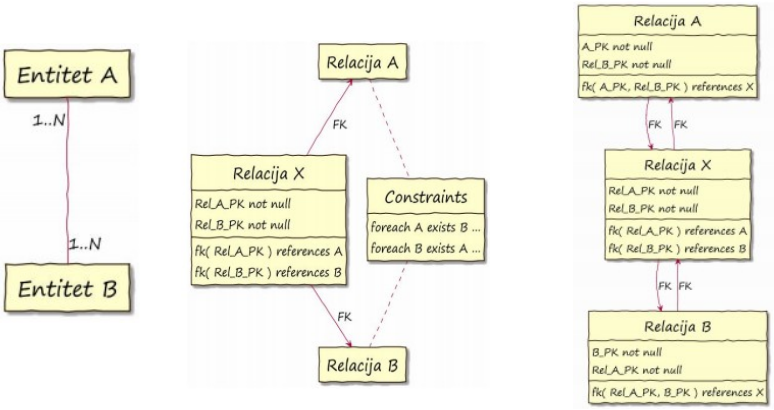


0.. – 1..** – dva standardna načina modeliranja, oba preko vezne tabele:

- Isto kao i u slučaju 0..* – 0..*, sa dodatnim uslovom da za svaki element relacije A postoji bar jedan dodeljeni element relacije B (što je jako skupo za proveru).
- Isto kao i u slučaju 0..* – 0..*, sa dodatnim (obaveznim) stranim ključem u relaciji A koji referiše na neki izabrani element relacije B preko vezne tabele. Ovako se izbegava dodatni uslov, ali je ažuriranje malo složenije (šta ako se, na primer, ažurira ili izbriše taj izabrani element).



1.. – 1..** – modelira se suštinski isto kao i prethodni slučaj (uz pomoć vezne relacije), samo što sada imamo dodatni uslov za obe strane (ili, alternativno, strani ključ ka izabranom elementu sa obe strane).

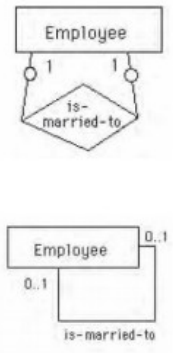


Prethodna dva slučaja mogu odgovarati agregacijama gde delovi mogu pripadati više različitih celina.

156. Kako se u logičkom (relacionom) modelu modeliraju binarni ciklični odnosi?

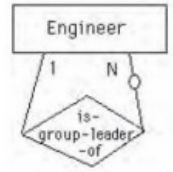
Konkretan način modeliranja binarnog cikličnog odnosa zavisi od njegove kardinalnosti.

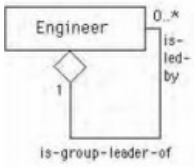
U slučaju cikličnog odnosa 1-1 (0..1-1, 0..1-0..1), odnos se modelira dodavanjem najobičnijeg stranog ključa koji će pokazivati na neki red te iste relacije.




```
create table employee (
     emp_id char(10),
     emp_name char(20),
     spouse_id char(10),
     primary key( emp_id ),
     foreign key( spouse_id )
            references employee
            on delete set null
            on update cascade
);
```

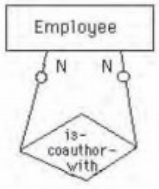
U slučaju cikličnog odnosa 1-*, taj odnos se takođe modelira dodavanjem stranog ključa, samo što se ovaj put on obavezno uvodi na strani *.

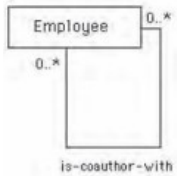





```
create table engineer (
     emp_id char(10),
     leader_id char(10) not null,
     primary key( emp_id ),
     foreign keg( leader_id )
            references engineer
            on delete set default
            on update cascade
);
```

U slučaju cikličnog odnosa *-*, potrebno je uvesti novu, veznu, relaciju koja će sadržati po strani ključ za oba učesnika u odnosu (koji, ujedno, i čine primarni ključ vezne tabele – potpuno isto modeliranje kao u slučaju klasičnog *-* binarnog odnosa).






```
create table employee (
     emp_id char(10),
     emp_name char(20),
     primary key( emp_id )
);
create table coauthor (
     author_id char(10) not null,
     coauthor_id char(10) not null,
     primary key( author_id, coauthor_id ),
     foreign key( author_id )
            references employee
            on delete cascade
            on update cascade,
     foreign key( coauthor_id )
            references employee
            on delete cascade
            on update cascade
);
```

U slučaju cikličnih odnosa gde u ciklusu učestvuje više relacija (npr. A -> B -> C -> A), odnosi se (u većini slučajeva) modeliraju kao standardni odnosi, a ne kao binarni ciklični odnosi.

157. Kako se u logičkom (relacionom) modelu modeliraju odnosi sa više učesnika?

Odnosi sa više učesnika se, u većini slučajeva, modeliraju pomoću nove relacije koja, između ostalog, sadrži strane ključeve ka učesnicima u datom odnosu. Zavisnosti između učesnika se uređuju dopuštanjem NULL vrednosti i postavljanjem uslova jedinstvenosti nad odgovarajućim ključevima. U slučaju odnosa sa tri učesnika, razlikujemo glavne slučajeve 1-1-1, 1-1-*, 1-*-* i *-*-*.

Alternativno, u nekim situacijama (što zavisi od konkretnih zavisnosti između učesnika u odnosu) je moguće izbeći novu relaciju direktnim postavljanjem stranih ključeva u relacije koje odgovaraju učesnicima.

Pogledati slajdove za detaljniji opis.

158. Na koje se sve načine u logičkom (relacionom) modelu može predstaviti hijerarhijski odnos (generalizacija, specijalizacija...)?

Tri osnovna načina na koja se to može izvesti su:

- Prevođenje svakog entiteta hijerarhije u zasebnu relaciju
- Prevođenje svakog lista hijerarhije u zasebnu relaciju
- Prevođenje čitave hijerarhije u jednu relaciju

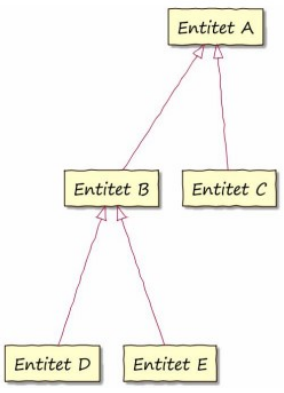
Uz to, ove metode je moguće kombinovati za različite delove hijerarhije.

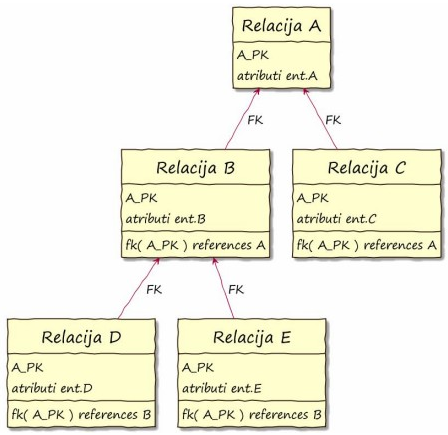
159. Objasniti prevođenje hijerarhija u logički model (RM) primenom principa "svaki entitet posebno". Šta su osnovni kvaliteti i slabosti?

U slučaju prevođenja hijerarhije tako što se svaki njen entitet prevodi u zasebnu relaciju, princip prevođenja odnosa specijalizacije je maltene isti kao i u slučaju klasične agregacije ili asocijacije. Naime, relacija svakog izvedenog entiteta sadrži samo svoje atribute i strani ključ ka svojoj neposrednoj baznoj relaciji.

Prednosti ovog pristupa su to što je relativno jednostavan, nema redundantnosti i dobro funkcioniše ukoliko je u pitanju hijerarhija bez prekrivanja (ne mora svaki red baznog entiteta da ima odgovarajući red u relaciji entiteta nekog od listova) i ukoliko je dozvoljeno preklapanje (moguće je da na jedan red relacije baznog entiteta referiše više redova iz različitih izvedenih entiteta).

Sa druge strane, velika mana je što su potrebna česta spajanja prilikom čitanja (ukoliko je potrebno da pročitamo atribute koji pripadaju različitim nivoima hijerarhije, potrebno je da spojimo sve odgovarajuće relacije), što može biti neefikasno. Dodatno, ukoliko je u pitanju hijerarhija sa prekrivanjem, potrebno uvesti dodatne provere (da li svaki red baznog entiteta ima odgovarajući red u nekom od listova), kao i u slučaju da preklapanje nije dozvoljeno (takođe je potrebno uvesti dodatne provere, konkretno da se na jedan red baznog entiteta ne može referisati iz više izvedenih entiteta).



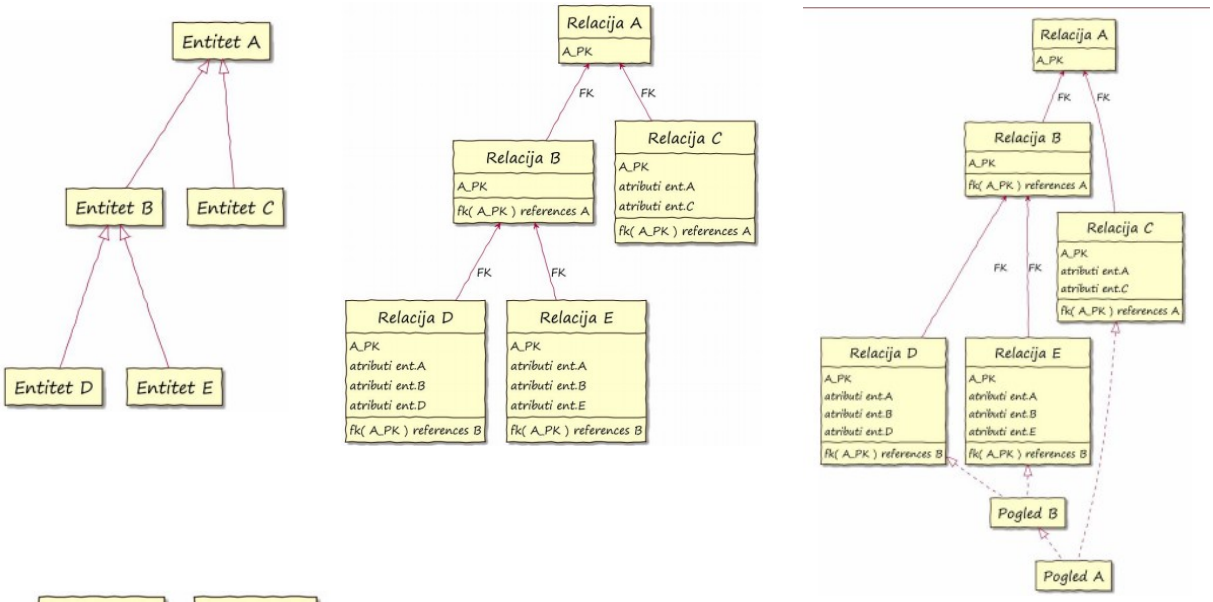


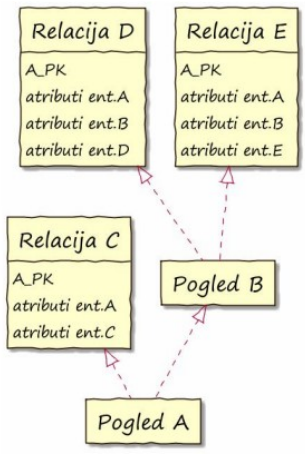
160. Objasniti prevođenje hijerarhija u logički model (RM) primenom principa "svaki list posebno". Šta su osnovni kvaliteti i slabosti?

U slučaju ovog principa prevođenja hijerarhije, svi listovi hijerarhije se prevode u zasebne relacije koje sadrže, kako svoje, tako i atribute svih entiteta na putu do (korene) bazne relacije, i uz to i strani ključ ka toj korenoj baznoj relaciji. Ostali entiteti se ili uopšte ne prevode kao relacije, ili se prevode kao relacije koje sadrže samo strani ključ ka baznoj relaciji. Za sve ne-list relacije se mogu praviti pogledi koji se dobijaju kao unija svih njima izvedenih relacija.

Glavna prednost ovog principa je izuzetno efikasana upotreba entiteta-listova (sve atribute već imamo u datim relacijama, pa nije potrebno vršiti spajanja). Dodatno, ukoliko je hijerarhija bez prekrivanja (što možda i nema nekog smisla u ovom slučaju), nemamo dodatnih problema.

Jedna od mana je u tome što je upotreba apstraktnih entiteta (unutrašnjih) neefikasna zbog potrebe pravljenja unije svih specijalizacija. Pored ovoga, ukoliko je u pitanju hijerarhija sa prekrivanjem, potrebno je koristiti dodatne uslove da svaki red baznog entiteta ima odgovarajući red u nekom od listova (međutim, ukoliko se uopšte ne prave relacije za apstraktne, tj. unutrašnje entitete, ovo nije problem, jer onda red koji nije u listu ne može ni postojati). Takođe, ukoliko preklapanje nije dozvoljeno, potrebne su dodatne provere, a ukoliko jeste, uvodi se redundantnost (nasleđeni delovi entiteta se mogu ponavljati u više listova).





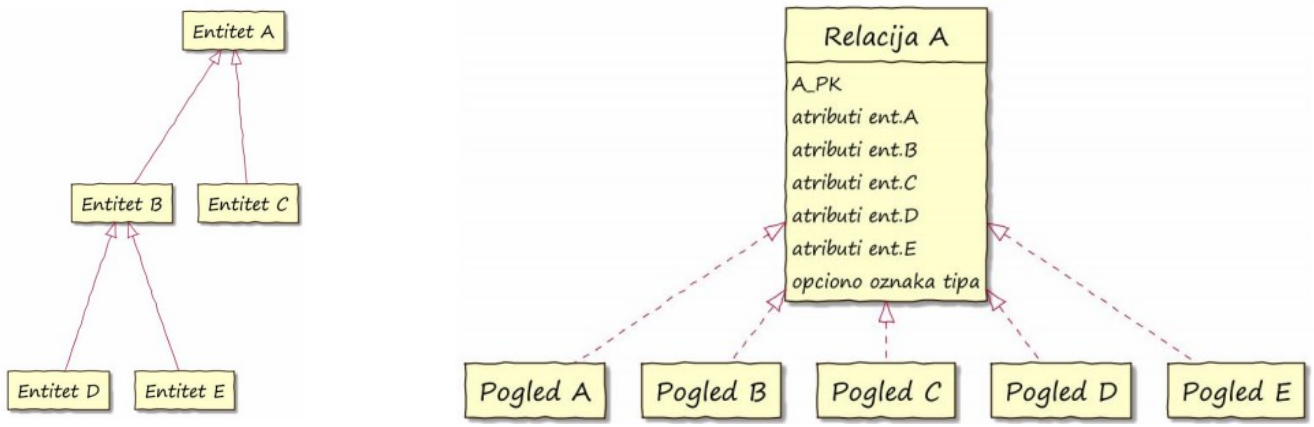
<-- alternativa u kojoj uopšte ne pravimo relacije za apstraktne entitete, već samo poglede koji se dobijaju unijom odgovarajućih listova

161. Objasniti prevođenje hijerarhija u logički model (RM) primenom principa "sve u jednu relaciju". Šta su osnovni kvaliteti i slabosti?

Primenom ovog principa, kreira se jedna relacija koja sadrži sve atribute koji sa javljaju unutar hijerarhije, s tim što će se za svaki od entiteta u hijerarhiji koristiti samo oni atributi koji su za njega definisani (a ostali će biti NULL). Pripadnost nekoj vrsti entiteta se može rešiti ili na osnovu vrednosti atributa (proveravanjem da li su određeni atributi NULL ili ne), ili na osnovu specijalnog surogat atributa koji će identifikovati vrstu entiteta. Nad ovom relacijom se dalje mogu praviti pogledi koji odgovaraju svakom od entiteta hijerarhije.

Glavna prednost ovog pristupa je izuzetno efikasna upotreba (nema potrebe za bilo kakvim spajanjem jer imamo samo jednu relaciju), kao i to što nisu potrebne bilo kakve dodatne provere u slučaju hijerarhije bez prekrivanja.

Sa druge strane, potencijalna mana je loša efikasnost kod memorijskog zauzeća, mada moderni SUBP-ovi sasvim dobro rešavaju ovaj problem. Pored toga, ukoliko je hijerarhija sa prekrivanjem, potrebno je dodati provere, mada su one poprilično jednostavne (provera da li su atributi nekog lista definisani za svaki red, ili jednostavna zabrana da oznaka tipa odgovara nekoj apstraktnoj klasi ukoliko se koristi pristup sa oznakama kod razlikovanja entiteta). Dodatno, ukoliko preklapanje nije dozvoljeno, to je relativno jednostavno rešivo proverom atributa ili oznake, a ukoliko jeste dozvoljeno, potrebno je preuzeti složeniji pristup prilikom označavanja entieta kojem red pripada. Naime, pošto u tom slučaju red može pripadati većem broju entiteta, potrebno je da oznaka podržava svaku od mogućih kombinacija. Jedan način da se to izvede je uvođenje binarnog atributa za svaki od entiteta umesto jedne oznake.



162. Šta je prečišćavanje sheme? Kako se odvija?

Prečišćavanje sheme je korak pri logičkog modeliranju baze podataka u kojem se logički model, inicijalno dobijen prevođenjem konceptualnog modela, dodatno usaglašava i prilagođava izabranom modelu podataka (npr. relacionom), pri čemu se vodi računa o, između ostalog, održavanja potpunosti modela. Proces prečišćavanja je iterativan, tj. svaka naredna iteracija sheme se pravi na osnovu prethodne njenim analiziranjem i izmenama pojedinačnih elemenata sheme po potrebi.

163. Šta su ciljevi prečišćavanja sheme?

Primarni cilj je potpuno prilagođavanje logičkog modela izabranom modelu podataka (npr. relacionom), ali pri tome je potrebno da budu ispunjeni i neki dodatni kriterijumi (čiji značaj zavisi od konkretnog modela podataka i konkretnog slučaja):

- Ispunjenost funkcionalnih zahteva modela (da li su prevedeni svi entiteti i odnosi)
- Ispunjenost nefunkcionalnih zahteva modela (više se odnose na implementaciju)
- Kompletnost modela
- Garancija integriteta podataka modelom
- Fleksibilnost modela
- Efikasnost modela
- Upotrebljivost modela

164. Objasniti probleme koji nastaju usled redundantnih podataka.

Osnovni problemi koji se pojavljuju usled prisustva redundantnih podataka su:

- Redundantno čuvanje podataka – u suštini neefikasna iskorišćenost memorijskih kapaciteta.
- Anomalije ažuriranja – usled postojanja više kopija jednog istog podatka, prilikom ažuriranja jedne kopije, potrebno je ažurirati i ostale, jer u suprotnom će se baza naći u nekonzistentnom stanju.
- Anomalije dodavanja – dodavanje jednog podatka može zahtevati i dodavanje još podatka bez kojih će se baza naći u nekonzistentnom stanju.
- Anomalije brisanja – brisanje jednog podatka može zahtevati i brisanje još podataka bez čega će se baza naći u nekonzistentnom stanju.

Primer redundantnosti – relacija oblika UpisanKurs(broj indeksa, ime i prezime studenta, šifra predmeta, naziv predmeta, ime i prezime nastavnika) – šta ukoliko se promeni ime studenta, naziv predmeta ili ime nastavnika? Morali bi da se menjuaju svi redovi u datoj relaciji koji sadrže neki od promenjenih podataka.

165. Kakva je uloga nedefinisanih vrednosti u rešavanju problema redundantnosti.

Upotreba nedefinisanih vrednosti u nekim slučajevima može pomoći u rešavanju anomalija dodavanja i brisanja koje se mogu javiti u prisustvu redundantnosti.

U slučaju dodavanja podataka, ukoliko ne znamo druge potrebne podatke, možemo ih popuniti NULL vrednostima. Na primer, posmatrajući relaciju oblika UpisanKurs(broj indeksa, ime i prezime studenta, šifra predmeta, naziv predmeta, ime i prezime nastavnika), ukoliko želimo da dodamo novog studenta, ali i dalje ne znamo ime nastavnika, to polje možemo popuniti sa NULL vrednosti.

Sa druge strane, ukoliko želimo da obrišemo neke redundantne podatke, ali bez brisanja čitavih redova, možemo te podatke zameniti NULL vrednostima. Na primer, posmatrajući relaciju oblika UpisanKurs(broj indeksa, ime i prezime studenta, šifra predmeta, naziv predmeta, ime i prezime nastavnika), ukoliko želimo da obrišemo podatke za datog nastavnika, ali bez brisanja podataka o studentima, možemo jednostavno postaviti NULL vrednost umesto brisanja celih redova.

Ipak, ovakav pristup može dovesti do nekih problema.

166. Objasniti postupak dekompozicije kao alat za otklanjanje redundantnosti.

Dekompozicija, tj. razbijanje relacije sa većim brojem atributa na manje relacije, se može koristiti kao sredstvo pri otklanjanju redundantnosti tako što će se atributi koji nemaju prirodne veze (što je najčešći uzrok redundandnosti) raspodeliti na razdvojene relacije.

Prilikom dekompozicije, potrebno je razmisliti o tome koji problem je uopšte potrebno i rešiti, da li predloženim načinom dekompozicije taj problem i rešavamo i da li možda vršenjem te dekompozicije uvodimo neke nove probleme. Kao pomoć u davanju odgovora na ova pitanja koristi se analiza funkcionalnih zavisnosti i teorija o normalnim formama relacija.

Primer jedne dekompozicije, u slučaju relacije UpisanKurs(broj indeksa, ime i prezime studenta, šifra predmeta, naziv predmeta, ime i prezime nastavnika), bio bi njeno razbijanje na relacije Student, Predmet, Nastavnik i UpisanKurs, gde bi se dalje koristili strani ključevi za modeliranje odgovarajućeg odnosa.

167. Šta su funkcionalne zavisnosti i zašto su važne za projektovanje baza podataka?

Funkcionalna zavnisnost predstavlja odnos između dva skupa atributa na skupu torki jedne relacije. Formalno – neka je data relacija R i neka su X i Y neprazni skupovi atributa te relacije. Tada skup torki r relacije R zadovoljava funkcionalnu zavisnost $X \rightarrow Y$ akko za svaki par torki t_1 i t_2 iz r važi $t_1.X = t_2.X \Rightarrow t_1.Y = t_2.Y$.

Važno je razlikovati pojam funkcionalne zavisnosti na nivou sadržaja relacije i na nivou sheme relacije. Naime, na nivou sadržaja relacije moguće je da se zaključe neke zavisnosti koje zapravo nemaju domenskog smisla (naručito u slučaju malih skupova podataka). Sa druge strane, ako posmatramo funkcionalne zavisnosti na nivou sheme relacije (možemo reći i samo na nivou relacije), važi da je sadržaj relacije ispravan akko ga čini skup redova za koje važe date funkcionalne zavisnosti, pa tako posmatrano možemo reći da funkcionalne zavisnosti na nivou relacije predstavljaju pravila integriteta.

Osim kao pravila integriteta, funkcionalne zavisnosti su od izuzetnog značaja u analizi i eliminaciji redundantnosti (složene FZ ukazuju na redundantnost), a takođe mogu ukazati i na mogućnost da se relacija može dekomponovati bez gubitka informacija.

168. Navesti i objasniti osnovne osobine ("aksiome") funkcionalnih zavisnosti.

Takozvane Armstrongove aksiome (suštinski nisu ni aksiome jer se trivijalno izvode iz same definicije funkcionalnih zavisnosti) su kompletne i zatvorene na skupu svih funkcionalnih zavisnosti (sve funkcionalne zavisnosti koje važe možemo izvesti od polaznih na osnovu osnovnih funkcionalnih zavisnosti pomoću ovih osobina). Za sve skupve atributa X, Y i Z neke relacije važe:

- Refleksivnost: $X \rightarrow X$, ili, ekvivalentno, $Y \subseteq X \Rightarrow X \rightarrow Y$, što suštinski znači da skup atributa jednoznačno određuje svaki njegov podskup
- Proširivost: $X \rightarrow Y \Rightarrow (\forall Z)\ XZ \rightarrow Y$, ili, ekvivalentno, $X \rightarrow Y \Rightarrow (\forall Z)\ XZ \rightarrow YZ$, što suštinski znači da ukoliko neki skup atributa X jednoznačno određuje drugi Y, onda i unija X sa bilo kojim skupom atributa Z takođe određuje Y
- Tranzitivnost: $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$

Dodatna svojstva su:

- Aditivnost: $X \rightarrow Y \wedge X \rightarrow Z \Rightarrow X \rightarrow YZ$
- Projektivnost: $X \rightarrow YZ \Rightarrow X \rightarrow Y \wedge X \rightarrow Z$
- Pseudotranzitivnost: $X \rightarrow Y \wedge YZ \rightarrow W \Rightarrow XZ \rightarrow W$

169. Objasniti odnos funkcionalnih zavisnosti i integriteta ključa.

Prvo je potrebno definisati pojmove natključa i ključa kandidata:

- Skup atributa X relacije R je natključ akko je X^+ = Attr(R), tj. akko je zatvorenje tog skupa atributa nad svim funkcionalnim zavisnostima (i direktnim i indirektnim) jednako skupu svih atributa relacije.
- Natključ X relacije R je ključ-kandidat akko za svaki drugi natključ Y važi $Y \subseteq X \Rightarrow Y = X$, tj. ukoliko ne postoji njegov podskup koji je takođe natključ.

Uslovu ključa onda odgovara funkcionalna zavisnost oblika {atributi ključa} -> {svi ostali atributi}. Međutim, ovde dolazi do problema, jer šta ukoliko neki atributi mogu da imaju nedefinisanu vrednost? Sama definicija funkcionalnih zavisnosti počiva na poređenju vrednosti atributa redova, a poređenje nedefinisanih vrednosti nema očigledno rešenje.

Iz prethodnog razloga, za primarni ključ se bira ključ-kandidat koji ne sme da ima nedefinisane vrednosti. Ovo odgovara integritetu primarnog ključa.

Sa druge strane, uz integritet primarnog ključa (PRIMARY KEY), možemo posmatrati i integritet jedinstvenog ključa (UNIQUE), koji se, zavisno od načina definisanja, može posmatrati kao slabiji oblik integriteta primarnog ključa. Naime, i u ovom slučaju je potrebno da važi uslov {atributi jedinstvenog ključa} -> {svi ostali atributi}, ali, pored toga, i zavisno od definicije, važi jedan od uslova:

- Atributi jedinstvenog ključa ne smeju da imaju nedefinisane vrednosti (kao kod integriteta primarnog ključa).
- Atributi jedinstvenog ključa smeju da imaju nedefinisane vrednosti, a za njih se, pri poređenju, smatra da su međusobno identične.
- Atributi jedinstvenog ključa smeju da imaju nedefinisane vrednosti, a za njih se, pri poređenju, smatra da su međusobno različite (ovo suštinski znači da se redovi koji sadrže nedefinisane vrednosti, čak iako postoje identični takvi redovi, ne porede, tj. smatraće se da su jedinstveni).

170. Objasniti odnos funkcionalnih zavisnosti i potpunosti dekompozicije.

Dekompozicija relacione sheme je proces zamene te relacione sheme sa dve (ili više njih) relacione sheme koje zajedno sadrže sve atribute polazne sheme. Ona je potpuna ukoliko odgovarajuće (rezultujuće) relacije sadrže sve one informacije koje bi sadržala i polazna relacija. Formalnije, kaže se da je dekompozicija potpuna akko polazna relacija može da se rekonstruiše spajanjem rezultujućih relacija.

Osobina potpunosti dekompozicije se može svesti i na očuvanje funkcionalnih zavisnosti na sledeći način:

- Dekompozicija čuva funkcionalne zavisnosti ako se iz funkcionalnih zavisnosti na dekomponovanim relacijama mogu rekonstruisati sve funkcionalne zavisnosti na polaznoj relaciji.
- Ako dekompozicija čuva funkcionalne zavisnosti, onda je ona potpuna (implikacija, ne ekvivalencija, tj. može biti potpuna čak i u nekim slučajevima gde se ne čuvaju FZ).

Formalnije, teorema glasi:

- Neka je Attr(R) skup atributa neke relacije R i neka su X i Y neprazni podskupovi tih atributa. Neka u polaznoj relaciji važi funkcionalna zavisnost X -> Y. Tada važi sledeća jednakost – R = R[XY] * R[XZ], gde je Z = Attr(R) \ XY. U suštini, ovo znači da ako iz polazne relacije izdvojimo novu relaciju koja modelira jednu FZ, a pri tome u polaznoj relaciji ostavimo domen te FZ (u ovom slučaju X), onda je takva dekompozicija potpuna čak iako ne garantuje očuvanje FZ.

171. Objasniti odnos problema redundantnosti i konceptualnog modela baze podataka.

Najčešće, neke funkcionalne zavisnosti nisu očigledne u konceptualnom modelu, što kao za posledicu ima mogućnost uvođenja redundantnosti na logičkom modelu u koraku prevođenja. Da bi se ovo izbeglo, tj. da bi se potencijalne redundantnosti uočile još u konceptualnom modelu, model bi morao da se neprekidno posmatra izuzetno pažljivo i da se, suštinski, deo po deo prevodi u logički (u našem slučaju u relacioni) model. Ovako nešto je, međutim, neostvarivo pre koraka integrisanja pogleda (jer pre toga nemamo potpune informacije koje bi nam omogućile da uvidimo redundantnosti), a čak i tada se retko radi – jednostavnije je ostaviti uklanjanje redundantnosti za korak prečišćavanja baze (u suprotnom, morali bi više puta da suštinski pravimo logički model, popravljamo konceptualni na osnovu uviđenih redundantnosti, ponovo pravimo logički, itd.).

172. Šta su normalne forme? Objasniti suštinu pojma i njegov značaj za projektovanje baza podataka.

Normalne forme su specijalni oblici relacija, tj. shema relacija, u odnosu na funkcionalne zavisnosti koje važe za njih, koji garantuju neke poželjne osobine relacija – u suštini, pružaju garanciju da neće biti redundantnosti određene vrste, što i predstavlja njihov glavni značaj u projektovanju baza podataka. Relacije se, određenim transformacijama, mogu dovesti do tih normalnih formi.

173. Navesti poznate normalne forme u uobičajenom redosledu. Šta predstavlja taj redosled?

- 1. normalna forma (1NF)
- 2. normalna forma (2NF)
- 3. normalna forma (3NF)
- Normalna forma elementarnog ključa (EKNF)
- Bojs-Kodova normalna forma (BCNF)
- 4. normalna forma (4NF)
- Normalna forma esencijalnih torki (ETNF)
- Normalna forma bez redundansi (RFNF)
- Normalna forma superključeva (SKNF)
- 5. normalna forma (5NF)
- Normalna forma domena i ključa (DKNF)

I, dodatno, 6. normalna forma (6NF). U prethodno navedenom redosledu (neračunajući 6NF), svaka naredna normalna forma podrazumeva da važe i sve prethodne. 6NF predstavlja alternativu za DKNF i ne može se neposredno uporediti sa ostalim normalnim formama.

174. Objasniti normalne forme 1NF, 2NF i 3NF.

- 1NF – relacija je u prvoj normalnoj formi akko svaki njen atribut može da ima samo atomične vrednosti. U relacionom modelu podataka, atomične vrednosti se pretpostavljaju, ali one ne moraju važiti u nekim drugim modelima (npr. ER model), pa zato kod prevođenja sa konceptualnog na relacioni logički model atribute koji mogu imati višestruke vrednosti uvek prevodimo u zasebne relacije. Na primer, u relaciji "predmet" posmatrajmo atribut "nastavnici" koji predstavlja listu svih nastavnika na tom predmetu. Takva relacija nije u 1NF, a to se može rešiti ili razbijanjem atributa "nastavnici" na atomične atribute "nastavnik1", "nastavnik2", ..., što je užasno loše rešenje, ili pravljenjem nove relacije.
- 2NF – relacija je u drugoj normalnoj formi akko je u prvoj normalnoj formi i ukoliko nijedan neključni atribut (atribut koji ne pripada nijednom kandidatu ključu) nije funkcionalno zavisan od nekog PRAVOG PODSKUPA atributa nekog kandidata ključa. U suštini, potrebno je da, u slučaju složenog ključa, svi ostali atributi zavise isključivo od celog tog ključa, a ne od nekog njegovog dela. Primer koji nije u 2NF je relacija UpisanPredmet(**indeks**, **šifrapred**, ime, prezime, nazivpred, kojiput) jer važi {indeks} -> {ime, prezime}, kao i {šifrapred} -> {nazivpred}, a, kao što možemo videti, indeks i šifrapred su pravi podskupovi ključa. Rešenje je podela na relacije UpisanPredmet(**indeks**, **šifrapred**, kojiput), Student(**indeks**, ime, prezime) i Predmet(**šifrapred**, nazivpred). Narušavanje ove NF ukazuje na pogrešno grupisanje atributa unutar relacije gde data relacija modelira više nepovezanih FZ. Potpuna dekompozicija do 2NF je uvek moguća.
- 3NF – relacija je u trećoj normalnoj formi akko je u drugoj normalnoj formi i ukoliko je svaki neključni atribut netranzitivno zavisan od svakog ključa. Alternativna definicija glasi:
  - Neka je R relacija, X neki podskup njenih atributa i A neki njen atribut, tada je R u 3NF akko za svaku funkcionalnu zavisnost X -> A na relaciji R važi jedno od:
    - $A \in X$ (trivijalna zavisnost)
    - X je natključ (X je ključ ili sadrži ključ)
    - A je deo nekog ključa u relaciji

  Primer jedne relacije koja je u 2NF ali nije u 3NF je Student(**indeks**, ime, prezime, studprogram, nivo, trajanje). Naime, atributi "nivo" i "trajanje" zavise od neključnog atributa "studprogram", pa je potrebno dekomponovati relaciju na Student(**indeks**, ime, prezime, studprogram) i Program(**studprogram**, nivo, trajanje). Narušavanje ove NF ukazuje na to da je u relaciji modelirano više relativno nezavisnih FZ. Kao i kod 2NF, potpuna dekompozicija do 3NF je uvek moguča.

175. Objasniti normalnu formu elementarnog ključa i BKNF. U čemu je njihov značaj? Navesti primer relacije koja je u NFEK a nije u BKNF.

Pre definisanja NFEK, potrebno je definisati pojmove elementarne funkcionalne zavisnosti i elementarnog ključa:

- FZ X -> Y je elementarna akko ne postoji FZ Z -> Y ni za koji pravi podskup atributa $Z \subset X$.
- Ključ je elementaran akko odgovara domenu bar jedne elementarne funkcionalne zavisnosti.

Tada važi:

- NFEK – relacija je u NFEK akko je u 3NF i za svaku elementarnu FZ X -> Y na datoj relaciji važi jedno od:
  - X je ključ
  - Y je elementarni ključ ili deo elementarnog ključa
- BKNF (Bojs-Kodova NF) – neka je R relacija, X neki podskup njenih atributa i A neki njen atribut, tada je R u BKNF akko za svaki FZ X -> A na relaciji R važi jedno od:
  - $A \in X$ (trivijalna zavisnost)
  - X je natključ (X je ključ ili sadrži ključ)

Suština BKNF je da je svaki atribut relacije ili deo ključa, ili zavisi od CELOG ključa (za razliku od 3NF), i uz to ne postoje zavisnosti među neključnim atributima, kao ni tranzitivne zavisnosti, što nas osigurava od određenih tipova redundantnosti. Sa druge strane, značaj NFEK je u tome što je ona poslednja normalna forma (nalazi se odmah pre BKNF) do koje se relacija sigurno može dovesti, a da se pritom očuvaju sve funkcionalne zavisnosti (ovo ponekad nije moguće sa BKNF, što ne znači da, iako pri potencijalnoj dekompoziciji zavisnosti nisu očuvane, data dekompozicija nije kompletna (setimo se, očuvane zavisnosti => kompletna dekompozicija, nije ekvivalencija)).

Sledi primer relacije koja je u NFEK, ali ne i u BKNF:

PrijavaIspita(**student**, **ispitni rok**, predmet), sa funkcionalnim zavisnostima:

- {student, ispitni rok} -> {predmet} // student u jednom ispitnom roku može da prijavi najviše jedan predmet
- {predmet} -> {ispitni rok} // svaki predmet se polaže u po jednom ispitnom roku (npr. Analiza 1 u roku Jun 1, Programiranje 1 u roku Jun 2, ... - svaki predmet ima svoj rok)

Data relacija nije u BKNF zbog druge funkcionalne zavisnosti, ali jeste u NFEK (obe FZ su elementarne, s tim što je prva od celog ključa (ispunjen prvi opcioni uslov), a druga ka delu ključa (ispunjen drugi opcioni uslov)). Relaciju je moguće dekomponovati (potpuna dekompozicija) na relacije:

- PrijavaIspita(**student**, predmet)
- PredmetURoku(**predmet**, ispitni rok)

Međutim, nisu očuvane sve zavisnosti. Naime, nije očuvano ograničenje da u jednom ispitnom roku student može da prijavi ispit iz najviše jednog predmeta. Ovo se jedino može rešiti uvođenjem redundantnosti.

176. Koje normalne forme se definišu na osnovu funkcionalnih zavisnosti?

1NF, 2NF, 3NF, EKNF i BCNF.

177. Objasniti ograničenja funkcionalnih zavisnosti. Kako se prevazilaze?

Glavno ograničenje funkcionalnih zavisnosti je to što za vrednost jednog skupa atributa određuju tačno jednu vrednost drugog skupa atributa (oblik funkcije – jedna vrednost iz domena se preslikava u tačno jednu vrednost iz kodomena). Naime, u nekim situacijama postoje zavisnosti koje nisu ovog oblika, već za neki skup atributa određuju skup vrednosti iz drugog skupa atributa. Primer ovoga bi bilo to što indeks studenta određuje skup upisanih godina tog studenta, ili npr. {indeks, sk_godina} određuje skup predmeta koje je student sa datim indeksom upisao u datoj školskog godini.

Posledica ovoga je što očuvanje funkcionalnih zavisnosti, iako daje dovoljan uslov za potpunu dekompoziciju, ne daje i neophodan uslov za potpunu dekompoziciju, tj. važi implikacija, a ne ekvivalencija. Ovaj problem se rešava uvođenjem koncepta višeznačnih zavisnosti.

178. Šta su višeznačne zavisnosti? U čemu je njihov značaj?

Neformalno, višeznačne zavisnosti predstavljaju uopštenje funkcionalnih zavisnosti gde vrednost jednog skupa atributa može da određuje veći broj vrednosti iz nekog drugog skupa. Formalno, definicija glasi:

Između skupa atributa X i njemu zavisnog skupa atributa Y relacije R ($X, Y \subseteq Attr(R), Z = Attr(R) \setminus XY$) postoji višeznačna zavisnost, u oznaci X ->> Y, akko za svake dve torke t_1 i t_2 iz R takve da je $t_1[X] = t_2[X]$ postoji torka t_3 takva da je:

- $t_3[X] = t_1[X] = t_2[X]$,
- $t_3[Y] = t_1[Y]$,
- $t_3[Z] = t_2[Z]$.

Njihov značaj je u pružanju podloge za definisanje viših oblika normalne forme. Ovo je moguće jer je uspostavljena ekvivalentnost između očuvanja višeznačnih zavisnosti i potpunosti dekompozicije na dve relacije (za razliku od samo implikacije kod funkcionalnih zavisnosti).

179. Navesti i objasniti osnovne osobine višeznačnih zavisnosti.

Neka su X, Y, W i Z skupovi atributa na R i uz to $W = Attr(R) \setminus XY$. Tada važe osobine:

- Komplementarnost: X ->> Y => X ->> W
- Refleksivnost: $Y \subseteq X$ => X ->> Y
- Proširivost: X ->> Y => XZ ->> YZ
- Tranzitivnost: X ->> Y $\wedge$ Y ->> Z => X ->> Z \ Y

Dodatno, u odnosu na funkcionalze zavisnosti (FZ), za višeznačne zavisnosti (VZ) važi:

- Konvertibilnost: X -> Y => X ->> Y (FZ su specijalni slučaj VZ)
- Interaktivnost: X ->> Y $\wedge$ XY -> Z => X -> Z \ Y (pod pretpostavkom da su X, Y, Z neprazni)

Zatvorenje svih VZ (skup svih VZ nad datom relacijom) se može izvesti konačnom primenom navedenih osobina, kao i aksioma FZ.

Pored ovoga, specijalni, trivijalni slučajevi VZ su:

- X ->> $\varnothing$
- X ->> Attr(R) \ X

180. Kakav je odnos višeznačnih zavisnosti i potpunosti dekompozicije?

Odnos je dat narednom teoremom:

Neka su X, Y i Z neprazni skupovi atributa relacije R, pri čemu je Z = Attr(R) \ XY. Tada na relaciji R važi jednakost R = R[XY] * R[XZ] akko važi višeznačna zavisnost X ->> Y.

Značenje ove teoreme je da postoji ekvivalencija (za razliku od FZ, gde je postojala implikacija) između očuvanja višeznačnih zavisnosti i potpunosti dekompozicije jedne na dve relacije. Važno je naglasiti da ovo važi samo za dekompoziciju na dve relacije, a ne na više njih (u tu svrhu se uvodi pojam zavisnosti spajanja).

181. Šta su zavisnosti spajanja i u čemu je njihov značaj?

Zavisnosti spajanja predstavljaju najopštiju vrstu zavisnosti (opštiju i od višeznačnih zavisnosti):

Def 1: Neka su X_1, X_2, ..., X_n skupovi atributa relacije R, čija je unija jednaka Attr(R). U relaciji R važi zavisnost spajanja, u oznaci *{X_1, X_2, ..., X_n}, akko je R = R[X_1] * R[X_2] * ... * R[X_n].

Def 2: Zavisnost spajanja *{X_1, X_2, ..., X_n} je trivijalna akko za neki od skupova atributa X_i važi X_i = Attr(R).

Def 3: Zavisnost spajanja *{X_1, X_2, ..., X_n} je reducibilna akko neka komponenta X_i može da se isključi, a da i dalje imamo zavisnost spajanja *{X_1, ..., X_{i-1}, X_{i+1}, ..., X_n}. Svaka reducibilna zavisnost spajanja (ZS) se može svesti na kombinaciju nereducibilne ZS i trivijalnih ZS.

Iako ova definicija (Def 1) nije praktično upotrebljiva (suštinski kaže samo da važi relacija spajanja nad nekim skupovima atributa akko se može izvršiti potpuna dekompozicija na relacije nad tim atributima), ona je potrebna za definisanje nekih viših oblika normalne forme.

182. Koje normalne forme se definišu na osnovu višeznačnih zavisnosti?

Direktno na osnovu višeznačnih zavisnosti se definiše samo 4NF, dok se ostale iznad nje (ETNF, RFNF, SKNF, DKNF, 6NF) mahom definišu preko zavisnosti spajanja. Naravno, da bi ove normalne forme važile, mora važiti i 4NF, pa se može reći i da se one definišu na osnovu višeznačnih zavisnosti.

183. Objasniti 4. normalnu formu. Kako se definiše i zašto je značajna?

Def: Relacija je u 4NF akko za svaku višeznačnu zavisnost X ->> Y važi da je X ključ ili natključ.

Suštinski, narušena 4NF znači da je relaciju moguće potpuno dekomponovati na dve jednostavnije relacije. Primer bi bila relacija Film{**gledalac**, **naslov**, **glumac**} gde važe višeznačne zavisnosti {naslov} ->> {gledalac} i {naslov} ->> {glumac}. Međutim, {naslov} nije natključ, te je 4NF narušena. Da bi se ovo rešilo, relaciju je potrebno razbiti na FilmGledalac{**naslov**, **gledalac**} i FilmGlumac{**naslov**, **glumac**}.

Značaj 4NF je u tome što njeno nezadovoljavanje može dovesti do drastičnog pada performansi usled povećanja broja redova u (većim nego što je potrebno) tabelama. Na primer, u prethodnom slučaju, ukoliko postoji 50 gledalaca i 20 glumaca, u objedinjenoj relaciji Film bi moralo da se čuva 50 * 20 = 1000 redova, dok bi sa razdvojenim relacijama FilmGledalac i FilmGlumac bilo potrebno samo 50 + 20 = 70 redova.

184. Objasniti 5. normalnu formu. Kako se definiše i zašto je značajna?

Def: Relacija je u 5NF akko su sve komponente svake netrivijalne zavisnosti spajanja natključevi relacije (zavisnost spajanja logički implicirana ključevima).

Intuitivno, ovo znači da u svakom skupu atributa u zavisnosti spajanja imamo ključ kao osnovu, i deljenjem relacije po skupovima atributa u zavisnosti spajanja, svaka dobijena relacija će sadržati dati ključ "kao osnovu". Primer je relacija Student{**indeks**, ime, prezime} sa zavisnošću spajanja *{{**indeks**, ime}, {**indeks**, prezime}}. Posledica 5NF je da svaka komponenta zavisnosti spajanja "sadrži" isti broj redova.

Značaj 5NF je u tome što je, ukoliko zanemarimo RFNF i SKNF koje više predstavljaju teorijski osnov za ovo, prva "praktična" normalna forma u kojoj je uklonjena sva redundantnost.

185. Koje normalne forme su između 4. i 5. NF? Kako se definišu i u čemu je njihov značaj?

Između 4NF i 5NF se, redom, nalaze:

- Normalna forma esencijalnih torki (ETNF)
  - Def: Relacija R zadovoljava ETNF akko svaka torka ove relacija mora da bude esencijalna.
  - Def: Torka je esencijanlna akko nije ni potpuno ni delimično redundantna. Intuitivno to znači da je torka esencijalna ukoliko nosi suštinski novu informaciju koja bez nje ne može da se izračuna.
  - Def: Torka t je potpuno redundantna u relaciji R akko postoji neki skup torki S relacije R ($t \notin S$) takav da se primenom FZ i pravila integriteta na taj skup može logički dokazati da torka t mora da postoji u relaciji R. Intuitivno, torka je potpuno redundantna ako cela može da se izračuna iz drugih torki.
  - Def: Torka t je delimično redundantna u relaciji R akko u istoj relaciji postoji FZ X -> A i torka q ($q \neq t$) takva da su im projekcije na X jednake, tj. t[X] = q[X]. Intuitivno, torka je delimično redundantna ako neki njen deo može da se izračuna iz drugih torki i njenog preostalog dela.
  - Teorema: Neka je relacija R određena samo funkcionalnim zavisnostima (FZ) i zavisnostima spajanja (ZS). Relacija R je u ETNF akko je u BKNF i svaka zavisnost spajanja ima neku komponentu koja je natključ (setimo se – u 5NF, sve komponente moraju biti natključevi).
- Normalna forma bez redundansi (RFNF)
  - Def: Relacija R zadovoljava RFNF akko ne može da postoji instanca relacije (tj. sadržaj relacije) koja sadrži neku torku t i neki atribut A tako da je par (t, A) redundantan.
  - Def: Par (t, A), gde je t torka, a A atribut, je redundantan u relaciji R akko za bilo koju torku q, koja se od t razlikuje samo u vrednosti atributa A, instanca relacije koja bi se dobila kada bi se torka t zamenila sa q NE BI predstavljala ispravnu vrednost. Intuitivno, vrednost atributa neke torke je redundantna ako se u toj relaciji ne može zameniti samo ta jedna vrednost, a da relacija i dalje ostane ispravna.
- Normalna forma superključeva (SKNF)
  - Def: Relacija R zadovoljava SKNF akko je svaka komponenta svake nereducibilne zavisnosti spajanja natključ.

Značaj ovih normalnih formi je uglavnom teorijski – ETNF i RFNF definišu minimalne zahteve da ne postoji redundantnost (ETNF po torkama, RFNF i po atributima), još i pre 5NF.

186. Objasniti normalnu formu domena i ključa i 6. normalnu formu. U čemu je razlika?

Def: Relacija zadovoljava NFDK akko u njoj ne postoje drugi uslovi i ograničenja osim uslova domena (dopuštene vrednosti atributa) i uslova ključa (jedine dopuštene FZ su uslovi ključeva).

Def: Relacija zadovoljava 6NF akko ne zadovoljava nijednu zavisnost spajanja, tj. akko ne postoji dalja potpuna dekompozicija za nju.

6NF najčešće nije praktična – do nje (u 5NF) je svakako već eliminisana redundantnost, a dalje razlaganje bi dovelo do nepotrebno velikog broja relacija (npr. relacija sa ključem i 20 neključnih atributa bi dovela do 20 odvojenih relacija ({**indeks**, ime}, {**indeks**, prezime}, ...)). Ipak, u nekim specifičnim situacijama može imati smisla (kolonske nerelacione baze podataka ili vremenske baze podataka (vreme izmene torke za svaki pojedinačni atribut)).

NFDK i 6NF nisu u potpunom skladu zbog toga što se u 6NF vrši dekomponovanje na trivijalne relacije, te je moguće izgubiti mogućnost elementarne provere potpunosti entiteta (da li svaka od tih trivijalnih relacija sadrži odgovarajući red). Na primer, posmatrajmo sledeću relaciju:

Student{**indeks**, ime, prezime}, sa FZ indeks -> {ime, prezime}

Jedina FZ je uslov ključa, te je ova relacija već u NFDK. Međutim, ona nije u 6NF jer se može izvršiti dalje dekomponovanje na dve relacije:

StudentIme{**indeks**, ime}, StudentPrezime{**indeks**, prezime}

Ovde je, za razliku od slučaja sa NFDK, moguće da u StudentIme imamo ime studenta sa indeksom 227/2018, ali da u StudentPrezime nemamo odgovarajuće prezime. Ovo je moguće osigurati jedino poprilično skupim dodatnim proverama integriteta.

187. Šta je "normalizacija"? Kada se primenjuje? Zašto? Kako?

Normalizacija je postupak transformisanja logičkog modela baze podataka, tj. njegovih relacija, u ekvivalentan skup relacija koji zadovoljava određeni oblik, tj. neku normalnu formu. Dovođenje relacija u određenu normalnu formu kao rezultat ima garanciju da su, zavisno od konkretne normalne forme, određeni tipove redundantnosti eliminisani.

Normalizacija se najčešće primenjuje u sklopu koraka prečišćavanja sheme prilikom pravljenja logičkog modela baze podataka.

Što se tiče normalizacije do BKNF, dat je sledeći rekurzivni algoritam:

1. Neka je R relacija koja nije u BKNF
2. Neka u R važi funkcionalna zavisnost X -> A, gde je X pravi podskup atributa iz R koji nije natključ, a A jedan atribut
3. Ukoliko se time ne gube neke FZ, izvršiti dekompoziciju R na dve relacije sa atributima XA i Attr(R)\A
4. Ukoliko neka od prethodno dobijene dve relacije nije u BKNF, rekurzivno ponoviti postupak nad njom

Ponekad, međutim, nije moguće izvršiti dekompoziciju do BKNF, a da ona čuva sve funkcionalne zavisnosti (međutim, ovo je uvek moguće do 3NF). U tim situacijama je potrebno ili ostati u 3NF, ili žrtvovati problematičnu zavisnost ukoliko ona nije bitna, ili da promenimo semantiku funkcionalnih zavisnosti. // Za ostale algoritme pogledati slajdove

188. Koliko daleko se obično sprovodi normalizacija?

Normalizacija se najčešće sprovodi do BKNF ili 4NF, mada je i 5NF relativno često poželjna. Do 6NF se ide samo u jako specifičnim slučajevima.

189. Šta je fizički model baze podataka?

Fizički model je najniži model prilikom modeliranja baze podataka (ispod konceptualnog i logičkog modela) i kao takav treba da opisuje konkretnu implementaciju baze podataka (npr. kako su podaci zaista organizovani na računaru). U ovoj fazi se veliki značaj pridodaje optimizaciji – organizacija podataka, njihova struktura i upiti nad njima se optimizuju u skladu sa konkretnim zahtevima.

190. Šta čini fizički model baze podataka?

Elementi fizičkog modela su:

- Struktura podataka (strukturom podataka se primarno bavio i logički model, ali to se nastavlja i kod fizičkog modela – dodatno, ovde se obično govori o "tabelama" i "kolonama" umesto "relacija" i "atributa")
- Interna organizacija podataka (kako su podaci fizički organizovani na računaru – prostori tabela, kontejneri, stranice, baferi, ...)
- Pomoćne komponente (indeksi)

191. Kako se procenjuje opterećenje baze podataka? Koji podaci su potrebni?

Procena opterećenja baze podataka se vrši na osnovu (njihovim razmatranjem) raznih prikupljenih podataka, primarno na osnovu modela obrade podataka (kako se koja tabela koristi prilikom dodavanja, ažuriranja, brisanja i čitanja), ali i nestrukturnih zahteva, aplikativnih zahteva, zahtevanih performansi, matrice entiteta i procesa i slično.

192. Šta obuhvata model obrade podataka, koji se koristi radi procene opterećenja pri pravljenju fizičkog modela?

Model obrade podataka sadrži informacije o operacijama koje će biti izvršavane nad bazom podataka (npr. ko će, kada, i koliko često dodavati redove u neku određenu tabelu). Konkretno, on za neku konkretnu tabelu sadrži:

- Okolnosti dodavanja novih redova (koliko redova se dodaje u neku tabelu u proseku kroz neki vremenski period, npr. dnevno, koliko ih se dodaje pri najvećem opterećenju, da li su primarni ključevi tih redova međusobno slični i da li zavise od vremena)
- Okolnosti ažuriranja postojećih redova (slično kao i kod dodavanja – koliko redova se menja u proseku i pri najvećem opterećenju, dodatno, kolika je verovatnoća da se redovi sa sličnim primarnim ključevima istovremeno koriste, što može biti korisno za efikasnije zaključavanje)
- Okolnosti brisanja redova (koliko redova se briše u proseku, pri najvećem opterećenju, da li se oni brišu pojedinačno ili u grupi)
- Okolnosti čitanja redova (učestalost čitanja, koliko redova se u proseku čita jednim upitom, koje kolone se najčešće koriste za odabir redova prilikom upita (WHERE), koje druge tabele se često koriste zajedno sa posmatranom prilikom zadavanja upita (zbog spajanja))

193. Koji nestrukturni zahtevi se razmatraju pri pravljenju fizičkog modela baze podataka?

Osnovni nestrukturni zahtevi koji se razmatraju su:

- Trajanje podataka – koliko dugo se podaci zadržavaju u tabeli pre njihovog brisanja ili arhiviranja. Naime, ovde je potrebno napraviti razliku između operativno aktivnih i statistički aktivnih podataka (student koji je završio studije pre 10 godina više nije operativno aktivan u bazi podataka, ali jeste važan za neke statistike, pa se on može arhivirati). Ranije su ovi zahtevi predstavljali veće probleme, ali usled povećanja dostupnog prostora za skladištenje podataka, ovi problemi postaju manji.
- Obim podataka – koliko će otprilike redova biti u nekoj tabeli prilikom puštanja baze u rad i kako će se taj broj redova menjati tokom vremena.
- Raspoloživost podataka – zavisno od toga koliko se često neki podaci koriste i koliko su važni, može biti potrebno da su oni uvek raspoloživi i da je pristup njima brz. U tim situacijama, ovi podaci se mogu skladištiti na bržim i pouzdanijim uređajima.
- Ažurnost podataka – koliko ažurni moraju da budu podaci koji se koriste.
- Bezbednosni zahtevi.

194. Koji su osnovni metodi optimizacije baze podataka? Objasniti ukratko.

Optimizacija na nivou interne organizacije podataka – u ovom slučaju se optimizacija ostvaruje kroz promenu interne, tj. fizičke organizacije podataka (prostori tabela, stranice, baferi stranica), kao i upotrebu pomoćnih struktura (indeksi).

Optimizacija na nivou upita – optimizacija se ostvaruje pisanjem što efikasnijih upita. Međutim, veliki deo ovog posla obavljaju automatski optimizatori u sklopu SUBP-a, mada postoje i slučajevi gde manuelna optimizacija može doprineti performansama (ukoliko ona uključuje neko specifično znanje koje se ne može zaključiti automatski).

Optimizacija na nivou strukture podataka – optimizacija izmenom same strukture podataka u odnosu na onu ustanovljenu logičkim modelom (denormalizacija – namerno uvođenje redundantnosti radi povećanja efikasnosti).

195. Koji su osnovni elementi fizičke organizacije podataka (na primeru SUBP DB2)?

Počev od najvišeg elementa, to su: instanca SUBP-a, baze podataka, particije, prostori tabela, kontejneri, jedinice čitanja ("extent") i stranice (jedinice pisanja). Dodatno se može pomenuti i bafer stranica.

196. Šta je prostor za tabele? Čemu služi?

Prostor za tabele predstavlja najviši i osnovni nivo fizičke organizacije podataka u okviru jedne baze podataka. Naime, prostor tabela ne određuje direktno gde će se podaci koji se u njemu nalaze skladištiti, ali, između ostalog, služi za organizovanje tzv. kontejnera koji upravo to određuju. Pored toga, prostori tabela služe i za definisanje određenih parametara prilikom fizičkog skladištenja i obrade podataka – kolika je veličina fizičke stranice (najčešće 4, 8, 16 ili 32 KB), koja je veličina jedinice čitanja (veličina extent-a) u broju stranica (najčešće od 2 do 256 stranica), načini baferisanja stranica, kompresija čuvanih podataka, ...

Prostor tabela "logički" sadrži tabele, koje su fizički organizovane po kontejnerima (ne nužno u samo jednom, zapravo najčešće se jedna tabela čuva u više kontejnera).

197. Šta je stranica baze podataka? Čemu služi?

Stranica predstavlja osnovni element fizičkog zapisa tabele ili indeksa – suštinski predstavlja blok memorije određene veličine koji služi za čuvanje jednog ili više redova neke tabele (moguće čak i ni jednog u slučaju da su svi redovi unutar jedne stranice obrisani). Redovi unutar stranica ne moraju biti fiksne dužine (npr. usled prisustva VARCHAR kolona).

Osnovni parametar stranice je njena veličina (najčešće 4, 8, 16 ili 32 KB) i ona se definiše na nivou prostora za tabele. Stranica ne mora biti popunjena do kraja (u proseku, neiskorišćeni prostor na kraju stranice je veličine polovine jednog reda). Odabir najbolje veličine zavisi od konkretne situacije (npr. velike stranice mogu sadržati više redova, pa su bolje od manjih ako je potrebno često dohvatanje grupa susednih redova, dok je, u slučaju čestog dohvatanja izolovanih redova, to samo trošak).

Jedna stranica se sastoji od zaglavlja stranice i zapisa. Zaglavlje sadrži podatke o zapisima, kao npr. lokacija svakog zapisa (reda) u okviru stranice, a zapisi sadrže konkretne podatke.

198. Šta je bafer za stranice? Čemu služi?

Bafer za stranice predstavlja memorijski prostor (u radnoj memoriji) koji čuva neki broj stranica (tj. njihovih kopija sa diska) radi njihove upotrebe (čitanja ili pisanja). Svakom prostoru tabela odgovara tačno jedan bafer za stranice.

Naime, svi podaci, radi obrade, moraju u nekom trenutku preći sa diska u radnu memoriju (pa onda u registre procesora). Ovo znači da je, usled čitanja podataka, potrebno dohvatiti sve odgovarajuće stranice sa diska i smestiti ih u bafer za stranice, a usled pisanja, potrebno je izmeniti odgovarajuće (prethodno dohvaćene) stranice unutar bafera stranica, pa njihovo kasnije upisivanje na disk.

Glavni parametar bafera za stranice je njegova veličina – što je veći, to je broj pristupa disku manji, pa je idealan slučaj da cela baza podataka stane u jedan bafer stranica. Međutim, ovo nije moguće za bilo koju veću bazu podataka, ali u slučajevima da postoje neki podaci kojima se jako često pristupa, oni se mogu smestiti u posebne prostore tabela, sa, naravno, odvojenim baferom stranica.

199. Šta je materijalizovani pogled? Kada se koristi i zašto?

Materijalizovani pogled predstavlja tabelu (sačuvanu u okviru nekog prostora tabela – u slučaju DB2 SUBP, u prostoru tabela TEMPSPACE1 po podrazumevanom ponašanju) čija su struktura i sadržaj definisani nekim upitom.

Najčešće se koriste u svrhe ubrzavanja složenih upita, i to nad podacima koji se retko menjaju (ili se lako mogu ažurirati u slučaju promene). Razlog za ovakav pristup je što, u slučaju čitanja podataka, dobijamo ubrzanje jer upit nije potrebno ponovo izvršiti (izvršava se jednom – onda kada se mat. pogled kreira). Međutim, u slučaju ažuriranja, umetanja ili brisanja podataka koji pripadaju, ili treba da pripadaju datom materijalizovanom pogledu, uz same podatke u njihovim stvarnim tabelama (tj. odgovarajućim stranicama), potrebno je ažurirati i stranice koje odgovaraju materijalizovanom pogledu.

Materijalizovani pogledi se, zavisno od SUBP-a, mogu kreirati automatski ili manuelno (od strane korisnika).

200. Objasniti ukratko strategije izolovanja transakcija i razlike među njima.

Dve osnovne strategije izolovanja transakcija su:

- Pesimistička strategija – pretpostavka je da će više različitih transakcija koje koriste iste podatke raditi u isto vreme, pa se podaci, pre pristupa njima, zaključavaju pomoću sistema katanaca.
- Optimistička strategija – pretpostavka je da će se pristup različitih transakcija istim podacima dešavati jako retko, pa se ne vrši zaključavanje, već se, prilikom pokušaja čuvanja rezultata transakcije, proverava da li su oni u međuvremenu izmenjeni od strane druge transakcije, u kom slučaju se transakcija poništava.

Razlika u ovim strategijama je u celokupnom pristupu. Naime, pesimistička strategija vrši prevenciju stvaranja konflikata na najnižem nivou, tj. prilikom svakog pristupa zaključanim podacima, dok, sa druge strane, optimistička strategija suštinski ne vrši prevenciju, već dozvoljava transakciji potpunu slobodu u radu sa podacima, pa se tek na samom kraju vrši provera da li je došlo do nekog konflikta. Zbog ovakvih pristupa, pesimistička strategija se koristi u bazama podataka, dok se optimistička uglavnom koristi za aplikacije (koje potencijalno koriste bazu podataka).

201. Objasniti pesimističko izolovanje transakcija.

Pesimističko izolovanje se vrši pod pretpostavkom će više različitih transakcija koje koriste iste podatke raditi u isto vreme, pa se podaci, pre pristupa njima, zaključavaju pomoću sistema katanaca. Ovi katanci se čuvaju do kraja transakcije, čime se obezbeđuje njihova izolovanost. Ovakav pristup se koristi u bazama podataka.

202. Objasniti optimističko izolovanje transakcija.

Optimističko izolovanje se vrši pod pretpostavkom da će se pristup različitih transakcija istim podacima dešavati jako retko, pa se, za razliku od pesimističkog izolovanja, ne vrši zaključavanje podataka. Umesto toga, vrši se pamćenje stanja korišćenih podataka radi provere da li je došlo do izmena u tim korišćenim podacima u toku izvršavanja transakcije, u kom slučaju se transakcija prekida. Glavna cena u ovom pristupu je u slučaju da je došlo do konflikta, jer se tada gubi čitava transakcija. Ovaj pristup se uglavnom ne koristi u bazama podataka, već u aplikacijama koje potencijalno koriste bazu (npr. sistemi za ORM često koriste ovaj pristup). Ipak, optimističko zaključavanje se može koristiti i u bazi podataka prilagođavanjem njene fizičke strukture – npr. dodavanjem kolona sa vremenom poslednje izmene reda ili dodavanjem "tokena izmene".

203. Objasniti osnovne elemente mehanizma katanaca.

Osnovni elementi svakog katanca su:

- Objekat koji se zaključava – određuje granularnost katanca, tj. njegov opseg. Objekti koji se mogu zaključati su pojedine vrednosti (atributi), red tabele, stranica tabele, grupa stranica, cela tabela, prostor za tabele, indeks.
- Trajanje katanca – vremenski interval trajanja katanca. U teoriji, samo držanje katanca do samog kraja izvršavanja transakcije garantuje potpunu izolovanost transakcija, ali, radi povećanja nivoa konkurentnosti, uvode se različiti nivoi izolovanosti transakcije.
- Vrsta katanca – radi povećanja nivoa konkurentnosti, uvode se različite vrste katanaca koje potencijalno mogu biti aktivne istovremeno na istom objektu. Osnovne vrste su S (Shared) i X (eXclusive) katanci, ali uz njih se, zavisno od konkretnog SUBP, uvode i drugi, pa u slučaju DB2 imamo U (Update), IS (Intent Shared), IX (Intent eXclusive) i druge.

204. Šta je eskalacija katanaca? Zašto je značajna?

Eskalacija katanaca je proces zamene većeg broja katanaca manje granularnosti sa manjim brojem katanaca veće granularnosti.

Naime, prilikom pristupa podacima u prisustvu sistema za zaključavanje, pre pristupa je potrebno proveriti da li je traženi podatak zaključan i, ukoliko jeste, kojeg tipa je dati katanac, što, u prisustvu velikog broja katanaca, može znatno smanjiti performanse celog sistema. Iz tog razloga, broj katanaca se ograničava (bilo konkretnim brojem katanaca ili količinom memorije koju lista katanaca može zauzimati). U slučaju da neka transakcija zahteva katanac, a kapacitet liste katanaca je već dostignut, pokušava se sa eskalacijom katanaca radi oslobađanja prostora u listi. Ukoliko eskalacija ne uspe, a moguće je da se to desi (npr. dva različita katanca drže redove u istoj stranici, pa se ne može izvršiti eskalacija), transakcija mora biti poništena.

Primer eskalacije bi bila zamena više katanaca na pojedinim redovima neke stranice sa katancem na čitavoj stranici, ili zamena više katanaca na stranicama sa katancem na čitavoj tabeli.

205. Šta su indeksi? Kada se i kako koriste?

Indeksi su pomoćne strukutre podataka koje omogućavaju brz pristup podacima u okviru tabela na osnovu nekog unapred izabranog ključa (u nekim situacijama, tj. pri nekim upitima, može se i potpuno eliminisati potreba za pristup tabeli jer je dovoljan samo pristup indeksu).

Indekse je poželjno koristiti nad iole većim tabelama (u slučaju jako malih tabela, moguće je da oni zapravo smanje efikasnost) sa ključevima (ključevima indeksa) koji odgovaraju čestim upitima nad datom tabelom. Postoji više vrsta indeksa, a najbolji izbor je često potrebno utvrditi eksperimentalno.

206. Navesti poznate vrste indeksa i ukratko objasniti.

U slučaju strukture indeksa, to su:

- Indeksi sa strukturom B-stabla – koriste balansirano drvo za predstavljanje indeksa, s tim što su svi listovi tog stabla (koji referišu na stranice odgovarajuće tabele) na istoj dubini.
- Bit-mapirani indeksi – koristi se bit mapa nad vrednostima ključnih atributa indeksa, što omogućava brzo pronalaženje redova koji zadovoljavaju složenije uslove, tj. uslove u kojima učestvuje više atributa nad kojima je izgrađen indeks.
- Heš indeksi – implementira se kao heš tabela, što omogućava brži direktan pristup konkretnom redu u odnosu na B-stabla, ali sporiji sekvencijalni pristup većem broju redova.

U slučaju dodatnih svojstava, to su:

- Jedinstveni indeksi – ne dozvoljavaju ponavljanje više redova sa istim ključem, u DB2 se koriste za implementaciju integriteta jedinstvenosti.
- Grupišući indeksi – obezbeđuju da su redovi u tabeli sortirani u odgovarajućem poretku na osnovu izabranog ključa, čime se ubrzava izdvajanje sekvenci redova u nekom poretku (npr. sa grupišućim indeksom nad cenom izdvajamo sve proizvode cene između 1000 i 2000 rsd), ali može se postaviti samo jedan ovakav indeks na datu tabelu.
- Particionisani indeksi – ovakav indeks je particionisan kroz više prostora tabela, često se koristi za primenu nad particionisanim tabelama, ali njegove particije ne moraju nužno odgovarati particijama tabele.
- Dvosmerni indeksi – za razliku od jednosmernog, omogućava brzo pretraživanje podataka u oba smera (lako možemo pronaći i naredni i prethodni red na osnovu indeksa), što može potencijalno zakomplikovati implementaciju indeksa u odnosu na jednosmerne.

207. Šta su jedinstveni indeksi?

Jedinstveni indeksi su indeksi koji zabranjuju ponavljanje više redova sa istim (indeksnim) ključem u tabeli nad koju su postavljeni, pa se kao takvi, u nekim SUBP-ovima, koriste za implementaciju integriteta jedinstvenosti.

Postoje tri vrste u zavisnosti od toga kako rukuju nedefinisanim vrednostima:

- Indeksi koji zabranjuju nedefinisane vrednosti
- Indeksi koji dozvoljavaju nedefinisane vrednosti, ali se za njih smatra da su jednake (pa se one ne mogu ponavljati)
- Indeksi koji dozvoljavaju nedefinisane vrednosti, ali se za njih smatra da su različite (pa se mogu ponavljati, čak iako bi indeksni ključevi u tim slučajevima bili strukturno "isti")

208. Šta su grupišući indeksi? Implementacija? Karakteristike? Prednosti i slabosti u odnosu na ne-grupišuće indekse?

Grupišući indeksi su indeksi koji osiguravaju da će redovi u odgovarajućoj tabeli biti sortirani prema redovima indeksa. Zbog ovoga je moguće postaviti najviše jedan grupišući indeks nad nekom tabelom (jer kako bi njeni redovi bili sortirani prema dva indeksa?).

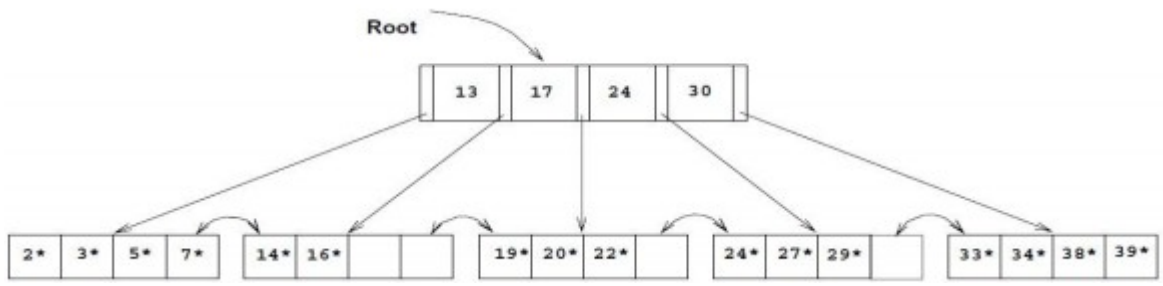
Implementacija, zbog toga što redovi tabele moraju biti sortirani, mora održavati tu sortiranost prilikom izmena (ili umetanja) redova u tabeli, pa je održavanje (usled operacija umetanja, brisanja i ažuriranja kolona koje pripadaju ključu indeksa) usporeno.

Glavna prednost ovakvog indeksa je to što omogućava izuzetno efikasno čitanje uzastopnih redova po nekom uslovu (npr. svi predmeti cene između 1000 i 2000 rsd). Međutim, slabosti su u tome što je održavanje usporeno i što ubrzanja suštinski nema prilikom pristupanja pojedinačnim redovima.

209. Šta su indeksi sa strukturom B-stabla? Implementacija? Karakteristike? Prednosti i slabosti?

Indeksi sa strukturom B-stabla koriste balansirano stablo gde je svaka stranica tabele referisana iz nekog lista tog stabla. Čvorovi stabla sadrže elemente u vidu ključeva za poređenje i pokazivača na čvorove narednog nivoa, ili, u slučaju listova, na stranice tabele. Veličina čvorova, tj. broj elemenata koje neki čvor može da sadrži, zavisi od veličine stranice prostora tabela u kojem se čuva dati indeks (jedan čvor odgovara jednoj stranici). Iz ovog razloga, i zbog toga što je stablo balansirano (pa je svaki list na istom nivou), vremenska složenost pristupa nekoj stranici tabele je proporcionalna logaritmu nad brojem stranica (čvorova u stablu) tabele, sa osnovom jednakoj broju elemenata po čvoru.

Ovakva struktura indeksa je u suštini podrazumevana u modernim SUBP-ovima. Prednosti su jednostavni i efikasni algoritmi za održavanje indeksa, ali je slabost smanjena efikasnost u slučaju da ima mnogo ponavljanja indeksnog ključa u tabeli (veliki broj redova, malo različitih vrednosti ključa).



210. Šta su bit-mapirani indeksi? Implementacija? Karakteristike? Prednosti i slabosti?

Bit-mapirani indeksi su indeksi strukturirani kao bit-mapa nad vrednostima indeksnog ključa. Naime, svakoj različitoj vrednosti ključa se dodeljuje niz bitova, gde svaki bit odgovara jednom od redova u tabeli. Vrednost bita je 1 akko odgovarajući red ima baš tu vrednost ključa.

Prednosti ovakve strukture indeksa su što, za razliku od indeksa sa strukturom B-stabla, efikasni u slučaju relativno malo različitih vrednosti ključa, kao i to što se efikasno kombinuju sa više indeksa (možemo, na primer, izvršiti konjukciju odgovarajućih nizova bitova ako se traže svi redovi koji zadovoljava jedan uslov i još neki drugi uslov). Sa druge strane, neefikasni su u slučaju velikog broja različitih vrednosti ključa, a dodatno je i održavanje indeksa često skuplje nego u slučaju B-stabla (dodamo red koji ima neku novu vrednost ključa (indeksnog), moramo da kreiramo čitav novi niz bitova samo zbog njega).

211. Šta su heš-indeksi? Implementacija? Karakteristike? Prednosti i slabosti?

Heš indeksi su indeksi koji su implementirani kao heš tabela sa datim ključem – heš vrednosti se računaju na osnovu ključnih atributa indeksa, pa se pomoću dobijene heš vrednosti pristupa odgovarajućim redovima u tabeli. Alternativna implementacija je da se umesto heš tabele koristi B-stablo, ali da se kao indeksni ključ koriste heš vrednosti.

Prednost ovakve strukture indeksa je što je pristup pojedinačnim redovima na osnovu tačno zadatog ključa izuzetno efikasna, ali je zato efikasnost izuzetno mala u slučaju sekvenvijalnog pristupa većem broju redova, ili ukoliko operator poređenja nije jednakost (npr. svi predmeti sa cenom između 1000 i 2000 rsd – za svaku cenu u tom rasponu se mora pojedinačno izračunati odgovarajuća heš vrednost). Dodatno, ukoliko je indeksni ključ složen, tj. čini ga više atributa, a u uslovu upita ne navodimo sve te atribute, već samo neke, heš indeks nam tu ne može pomoći.

Uglavnom se automatski pravi za pristupanje privremenim tabelama (tj. međurezultatima upita), naručito u slučaju velikog broja spajanja.

212. Šta su indeksi sa dodatnim kolonama? U čemu je njihov značaj?

Indeksi sa dodatnim kolonama su indeksi koji, pored kolona ključa, imaju pridodate i dodatne kolone koje ne čine uslov uređenja. Ovo omogućava izbegavanje pristupa stranicama tabele, već je dovoljan samo pristup indeksu, u slučajevima kada upit sadrži samo kolone čije vrednosti indeks već sadrži. Međutim, ovo, povećava fizičku veličinu indeksa i smanjuje dubinu B-stabla (zbog smanjenog broja elemenata po jednoj stranici usled povećanja memorijskog zauzeća jednog čvora).

213. Kada pravimo indekse, zašto i koliko? Da li uvek moramo da imamo indekse?

Indekse je poželjno praviti uvek kada je potreban efikasan pristup redovima neke tabele na osnovu određenih atributa (npr. kada jako često imamo upite nad nekim atributom). Dodatno, pored efikasnosti, moguće je da je potrebno, zavisno od SUBP-a, kreirati jedinstveni indeks radi ostvarivanja integriteta jedinstvenosti.

Konkretan broj indeksa koji se kreira, kao i njihova vrsta, zavisi od raznih faktora – vrste, namene, strukture, načina upotrebe konkretne tabele. Ipak, obično se, u slučaju transakcionih tabela, preporučuje 3 do 5 indeksa, dok se za analitičke tabele ne zadaje neko ograničenje (sve dok je ažuriranje, usled veće cene održavanja, prihvatljivo efikasno). Utvrđivanje najboljeg broja i vrste indeksa se vrši eksperimentalno.

Indekse nije uvek poželjno kreirati, konkretno u slučaju izuzetno malih tabela (koje staju u svega par stranica). U tim slučajevima, upotreba indeksa može čak i smanjiti efikasnost. Primer jedne ovakve tabele je tabela studijskih programa.

214. Šta su distribuirane baze podataka i distribuirani SUBP?

Distribuirana baza podataka je skup logički međuzavisnih baza podataka (nije dovoljno da to budu logički potpuno odvojene baze) koje su distribuirane na računarskoj mreži.

Distribuirani SUBP je softverski sistem za upravljanje distribuiranim bazama podataka, ali takav da je distribuiranost transparentna za korisnika, što suštinski znači da korisnik ne mora biti svestan da je i kako je baza distribuirana – time upravlja sam SUBP i detalje sakriva od korisnika.

215. Koji su osnovni doprinosi distribuiranih baza podataka?

Osnovni doprinosi distribuiranih baza podataka su povećane performanse i pouzdanost, veća skalabilnost, kao i manja cena opreme (zbog eksponencijalnog rasta cene u odnosu na potrebne performanse jednog računara).

216. Objasniti šta znači transparentno upravljanje distribuiranim i repliciranim podacima.

Transparentno upravljanje distribuiranim i repliciranim podacima podrazumeva skrivenost implementacionih problema niskog nivoa nastalih usled fragmentacije i replikacije podataka u distribuiranim bazama podataka, što u suštini znači da korisnik može da pristupa podacima potpuno nezavisno od načina implementacije distribuirane baze.

217. Koji su aspekti *transparentnosti upravljanja distribuiranim i repliciranim podacima*?

Aspekti su:

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

218. Objasniti *nezavisnost podataka* u kontekstu transparentnosti upravljanja distribuiranim i repliciranim podacima.

Aspekt nezavisnosti podataka se može posmatrati iz ugla logičke i iz ugla fizičke nezavisnosti podataka. Logička nezavisnost označava otpornost aplikacija (koje zavise od DSUBP-a) na promene logičke strukture podataka (dodavanje novih elemenata strukturi). Fizička nezavisnost označava potpunu skrivenost fizičke strukture podataka od aplikacija, što znači da aplikacije neće trpeti nikakve posledice u slučaju njene promene (npr. promene načina distribuiranja podataka).

219. Objasniti *mrežnu transparentnost* u kontekstu transparentnosti upravljanja distribuiranim i repliciranim podacima.

Apsekt mrežne transparentnosti označava skrivenost svih detalja mrežne komunikacije u okviru distribuiranog sistema od korisnika – iz ugla korisnika, sistem bi trebalo da se koristi potpuno isto kao i da je centralizovan, nezavisno od toga gde se podaci zaista nalaze (na primer, ukoliko se čuvaju na nekom udaljenom čvoru).

220. Objasniti *transparentnost replikacije* u kontekstu transparentnosti upravljanja distribuiranim i repliciranim podacima.

Aspekt transparentnosti replikacije označava da su detalji o replikaciji podataka (čuvanja više identičnih kopija podataka na više lokacija radi boljih performansi, veće pouzdanosti i raspoloživosti) potpuno sakriveni od korisnika – on ne bi trebalo da bude svestan da se podaci uopšte i repliciraju.

221. Objasniti *transparentnost fragmentacije* u kontekstu transparentnosti upravljanja distribuiranim i repliciranim podacima.

Aspekt transparentnosti fragmentacije označava da su detalji o fragmentaciji podataka (čuvanje različitih delova iste kolekcije podataka, bilo da je u pitanju horizontalna ili vertikalna replikacija, na više različitih lokacija radi boljih performansi, veće pouzdanosti i raspoloživosti) potpuno sakriveni od korisnika – on ne bi trebalo da bude svestan da su podaci uopšte fragmentisani (što se može postići, na primer, automatskim prevođenjem korisnikovog globalnog upita na skup manjih, lokalnih upita koji će se izvršavati nad pojedinačnim fragmentima, nakon čega se rezultat unira).

222. Šta i kako može biti nosilac transparentnosti upravljanja distribuiranim i repliciranim podacima?

Nosilac transparentnosti može biti:

- Upitni jezik – svaki (korisnikov) globalni upit se, u zavisnoti od konkretne organizacije distribuiranih podataka, automatski prevodi na odgovarajući način, čime se distribuiranost podataka skriva od korisnika.
- Operativni sistem – operativni sistem je zadužen za rešavanje problema koji nastaju pri distribuiranju podataka. Ovakav pristup se veoma retko koristi u potpunosti, ali DSUBP-ovi mogu koristiti određene mehanizme pružene od strane operativnog sistema za implementaciju nekih funkcionalnosti.
- DSUBP – najčešći pristup, SUBP rešava sve probleme koji nastaju pri distribuiranju podataka.

223. U čemu se ogleda unapređenje performansi usled distribuiranja?

Unapređenje performansi usled distribuiranja podataka se najčešće ogleda u tome što:

- Fragmentisani podaci se čuvaju bliže mestu upotrebe – troškovi mrežne komunikacije se time smanjuju, naručito u slučaju kada je kašnjenje, tj. vreme odziva mreže visoko.
- Globalni (korisnikovi) upiti su paralelizovani – usled fragmentacije podataka, globalni upiti se mogu podeliti na manje, lokalne upite koji se izvršavaju nad pojedinačnim fragmentima, nakon čega se rezultat unira.

Dodatno, čak i u situaciji da su potrebne još bolje performanse, distribuirane sisteme karakteriše veći nivo skalabilnosti, te se oni uvek mogu dodatno proširiti u skladu sa potrebama.

224. Koji su osnovni otežavajući faktori pri implementaciji DBP? Objasniti.

Osnovni otežavajući faktori prilikom implementacije distribuiranih baza podataka su:

- Velika složenost implementacije – uz sve probleme koje rešavaju centralizovane baze podataka, potrebno je rešavati i probleme koji nastaju usled distribuiranja podataka (kako osigurati transparentnost distribuiranja i replikacije usled fragmentacije, replikacije, upotrebe računarske mreže).
- Cena – iako je cena pojedinačnih hardverskih komponenti manja nego kod komponenti visokih performansi koje bi se koristile u centralizovanim sistemima (distribuirani sistem vs *mainframe* računar), potrebno je uračunati dodatnu cenu mrežne opreme, veću cenu održavanja sistema, kao i veću cenu samog softvera zbog njegove veće složenosti.
- Distribuirana kontrola – potrebno je rešiti problem sinhronizacije i koordinacije svih komponenti distribuiranog sistema.
- Bezbednost – umesto samo jednog čvora kod centralizovanih sistema, potrebno je obezbediti sve (potencijalno i više stotina, čak i hiljada) čvorove distribuiranog sistema, bilo u smislu sigurnosti, bilo u smislu prevencije nezgoda (požari i slično).

225. Navesti najvažnije probleme i teme istraživanja u oblasti DSUBP.

Neki važni problemi i teme istraživanja u oblasti distribuiranih SUBP-ova su:

- Projektovanje distribuirane baze podataka (kako optimalno fragmentisati i replicirati podatke)
- Distribuirano izvršavanje upita (kako paralelizovati globalni upit na različite fragmente)
- Distribuirano upravljanje metapodacima (dodatni podaci o strukturi, distribuiranosti i sadržaju baze podataka koji se moraju čuvati i održavati zarad njenog ispravnog funkcionisanja)
- Distribuirana kontrola konkurentnosti (sinhronizacija pristupa podacima usled konkurentnog izvršavanja procesa, pri čemu se održava integritet baze podataka)
- Distribuirano upravljanje mrtvim petljama (kako razrešiti mrtve u distribuiranim sistemima – prevencija ili prepoznavanje i oporavak)
- Pouzdanost distribuiranih SUBP
- Podrška u operativnim sistemima
- Heterogene baze podataka

226. Šta su heterogene baze podataka?

Heterogene baze podataka su baze, često nastale spajanjem već postojećih baza podataka, u kojima je narušena homogenosti softvera i/ili homogenost strukture. Narušavanje homogenosti softvera podrazumeva upotrebu različitih SUBP-ova u različitim delovima heterogene baze (različite baze od kojih je heterogena nastala, njihovim spajanjem, su mogle da koriste različite SUBP-ove). Narušavanje strukture podrazumeva neusaglašenost u strukturi različitih delova heterogene baze. Alternativni i deskriptivniji naziv za ovakve sisteme je "sistemi sa više baza podataka".

227. Navesti i ukratko objasniti najvažnije ciljeve pri pravljenju distribuiranih sistema.

Glavni ciljevi kojima se teži pri pravljenju distribuiranih sistema su:

- Konzistentnost (Consistency) – operacija čitanja mora dati isti rezultat na svim čvorovima distribuiranog sistema.
- Raspoloživost (Availability) – odziv sistema mora da postoji u nekim garantovanim vremenskim granicama.
- Prihvatanje razdvojenosti (Partition tolerance) – sistem mora da nastavi sa ispravnim funkcionisanjem čak i u slučaju njegove razdvojenosti (na primer nastale usled prekida komunikacija između čvorova).

228. Objasniti *konzistentnost* kao jedan od osnovnih ciljeva pri pravljenju distribuiranih sistema.

Konzistentnost, u kontekstu distribuiranih sistema, ima značenje da sve operacije čitanja, na svim čvorovima, moraju dati isti rezultat. Ovo u suštini znači da sve replike nekog podatka moraju biti identične na svim čvorovima. Važno je napomenuti da se ovako definisana konzistentnost razlikuje od one iz ACID osobina transakcija (gde svaka transakcija prevodi bazu iz jednog konzistentnog u drugo konzistentno stanje).

229. Objasniti *raspoloživost* kao jedan od osnovnih ciljeva pri pravljenju distribuiranih sistema.

Raspoloživost distribuiranog sistema je njegova mogućnost da uvek i na svaki zahtev da odziv u određenim vremenskim granicama. Najčešća situacija je da sistem nije raspoloživ upravo u onim trenutcima kada je najpotrebniji, tj. u trenutcima visoke opterećenosti sistema.

230. Objasniti *prihvatanje razdvojenosti* kao jedan od osnovnih ciljeva pri pravljenju distribuiranih sistema.

Prihvatanje razdvojenosti se odnosi na osobinu distribuiranog sistema da, u slučaju otkazivanja čvorova ili komunikacija, pa čak i kada to otkazivanje rezultuje particionisanjem sistema, ali osim u slučaju da čitav sistem otkaže, on nastavi sa radom bez proizvođenja negativnog odziva sistema (tj. bez poništavanja transakcije).

231. Koji su osnovni doprinosi koji se očekuju od distribuiranog SUBP?

Pitanje 215.

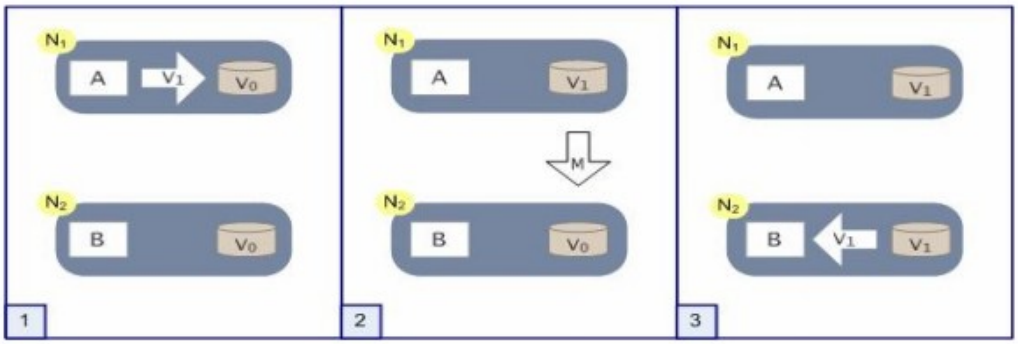
232. Koji su osnovni otežavajući faktori pri pravljenju distribuiranih baza podataka? Navesti i ukratko objasniti.

Pitanje 224.

233. Navesti teoremu CAP i ukratko je objasniti.

Teorema CAP nalaže da nije moguće definisati distribuirani sistem koji zadovoljava sva tri uslova od konzistentnosti, raspoloživosti i tolerancije na razdvojenost. Sa druge strane, moguće je zadovoljiti bilo koja dva uslova od ova tri.

Intuitivno, CAP teorema se može shvatiti na osnovu minimalnog primera. Neka su data dva čvora, N_1 i N_2. Transakcija A podatak v_0 na čvoru N_1 ažurira u v_1, nakon čega se ta promena propagira na čvor N_1 i na kraju čita od strane transakcije B.



Međutim, šta uraditi ukoliko dođe do prekida u komunikaciji između ova dva čvora? Postoje tri moguće opcije, ali svaka od njih remeti jednu od CAP osobina:

a) Transakcija A se poništava
   - Neuspehom ažuriranja transakcijom A usled razdvojenosti sistema, narušena je osobina tolerancije na razdvojenost.

b) Transakcija A se uspešno izvršava na čvoru N_1
   - Ažuriranje je izvršeno samo na čvoru N_1 (gde se sada nalazi podatak v_1), ali ne i na čvoru N_2 (gde se i dalje nalazi podatak v_0), pa sistem nije u konzistentnom stanju (osobina konzistentnosti je narušena).

c) Transakcija A čeka na uspešno slanje poruke M
   - Nikako se ne može znati kada će poruka M uspešno biti poslata, pa je vreme odziva nepoznato (ne može se garantovati vreme u kojem će transakcija biti završena), i time je narušena osobina raspoloživosti.

234. Koji vidovi kompromisa se prave radi prevazilaženja ograničenja koja proističu iz teoreme CAP?

Kompromisi koje prave su ili potpuno odbacivanje neke od CAP osobina (toleranciju razdvojivosti, raspoloživost ili konzistentnost), ili ublažavanje (pa čak i zasnivanje na potpuno drugim uslovima) originalnih CAP osobina (npr. odložena konzistentnost umesto stalne konzistentnosti).

235. Objasniti "odbacivanje prihvatanja razdvojenosti" kao posledicu teoreme CAP.

Odbacivanjem osobine tolerancije razdvojenosti se pristaje na to da, u slučaju razdvojenosti distribuiranog sistema, on neće raditi. Međutim, ovo najčešće nije prihvatljivo rešenje – u slučaju razdvajanja, pa čak i najmanjeg, ceo sistem prestaje sa radom do oporavka (primer osnovni ROWA protokol gde se čitanje zaustavlja ukoliko bilo koj čvor nije raspoloživ).

Neke alternative kojima se ovaj problem ublažava su upotreba centralizovane baze podataka (što najverovatnije nije prihvatljivo kada već pokušavamo sa distribuiranim sistemom) ili uvođenje velike redundantnosti u mrežu i time smanjivanje verovatnoće particionisanja sistema. Ipak, obe ove alternative su izuzetno skupe.

236. Objasniti "odbacivanje raspoloživosti" kao posledicu teoreme CAP.

Odbacivanje osobine raspoloživosti dovodi do toga da ne postoji garancija u vremenu odziva prilikom razdvojenosti sistema. Posledice ovoga se donekle mogu umanjiti pažljivim projektovanjem distribuiranog sistema, tj. uspostavljanjem što niže sprege između čvorova. Ipak, ovo rešenje je najčešće neprihvatljivo jer sistem koji nije raspoloživ je praktično neupotrebljiv, čak i više nego u slučaju netoleracije razdvojenosti (najčešće je bolje da transakcija odmah ne uspe nego da se neograničeno dugo čeka na njen uspešan završetak). Ovaj pristup se koristi jedino ukoliko su ostale dve osobine (konzistentnost i tolerancija razdvojenosti) od kritičnog značaja.

237. Objasniti "odbacivanje konzistentnosti" kao posledicu teoreme CAP.

Odbacivanjem konzistentnosti se omogućava da se različite replike istog podatka nalaze na različitim čvorovima u nekom trenutku, tj. da jedan upit da različite rezultate u zavisnosti od čvora na kojem se izvršava. Iako ovo, naizgled, zvuči nedopustivo, u velikom broju situacija ovo zapravo i ne predstavlja veliki problem, pa je, ukoliko se bira jedna od tri CAP osobine, ovo najčešća osobina za odbacivanje. Primer su transakcije u nekoj onlajn prodavnici – ukoliko se izmeni cena proizvoda, a ta izmena ne stigne da se propagira na sve čvorove, moguće je da će se neke transakcije izvršiti po staroj ceni (u zavisnosti od čvora na kojim se izvrše), ali to ne bi trebalo da predstavlja preveliki problem za firmu koja vrši prodaju. Ili, na primer, šta ukoliko se nekim korisnicima Twitter-a tvitovi prikažu kasnije nego drugim korisnicima? Ovo, zaista, nije veliki problem.

238. Na čemu počivaju alternativni skupovi uslova za projektovanje distribuiranih sistema, koji se uvode radi prevazilaženja posledica teoreme CAP?

Ovi alternativni skupovi uslova obično počivaju na standardnim osobinama transakcija u bazama podataka – takozvanim ACID osobinama (Atomicity, Consistency, Isolation, Durability). Primer alternativnog skupa u odnosu na ACID su tzv. BASE uslovi (Basically Available, Soft-state, Eventually consistent).

239. Navesti i objasniti izmenjen skup uslova integriteta baze podatka – BASE.

Uslovi integriteta BASE su:

- Basically Available (osnovna raspoloživost) – postoji gornje vremensko ograničenje za izvršavanje svake transakcije, bilo da je ona izvršena uspešno ili prekinuta po dostizanju tog ograničenja (nakon čega se korisniku šalje obaveštenje o neuspehu).
- Soft-state – stanje sistema se može menjati i van transakcija (na primer radi asinhrone replikacije).
- Eventually consistent (odložena konzistentnost) – stalna konzistentnost i izolovanost transakcija nije garantovana, ali zato je garantovano da će sistem doći u konzistentno stanje u nekom trenutku u budućnosti (verovatno asinhronom replikacijom).

240. Objasniti specifičnosti projektovanja baze podataka u odnosu na uslove BASE.

Pored funkcionalne dekompozicije podataka na fragmente, što je svakako potrebno izvršiti u okviru projektovanja distribuirane baze podataka, nevezano od skupa uslova integriteta, potrebno je uskladiti i implementaciju transakcija prema BASE uslovima (tačnije vreme izvršavanja operacija transakcije), kao i odrediti uslove replikacije.

Što se tiče fragmenata dobijenih dekompozicijom, u okviru njih (na jednoj replici) važe ACID uslovi, dok je između različitih fragmenata dovoljno da važe samo BASE uslovi. Ovo osigurava lokalnu konzistentnost svakog fragmenta.

Transakcije se, radi implementacije, mogu podeliti na sinhroni i asinhroni deo. Svaka transakcija odgovara jednom, matičnom fragmentu i izmena podataka od strane te transakcije u okviru matičnog fragmenta se implementira sinhrono (prateći standardne ACID osobine). Dodatno, sinhrono se, u okviru transakcije, te izmene i (samo) evidentiraju u drugim fragmentima, ali se same potrebne izmene na osnovu evidentiranih u drugim fragmentima sprovode asinhrono. Iz ovog razloga rezultati logički povezanih upita koji se izvršavaju na različitim fragmentima potencijalno mogu biti privremeno nekonzistentni.

Za uslove replikacije se pretpostavlja da između replika važe BASE uslovi i da se njihova sinhronizacija odvija van transakcija, tj. asinhronim procesom. Ovo je razlog zašto rezultati upita na različitim replikama potencijalno mogu biti privremeno nekonzistentni.

241. Objasniti pojam "konflikata" pri projektovanju baze podataka na osnovu uslova BASE i načine njihovog razrešavanja.

Konflikti mogu nastati u slučaju da se dve replike jednog istog podatka nezavisno menjaju u slučaju razdvojenosti sistema na više particija. Naime, pri oporavku sistema, očigledno je da se samo jedna od verzija podatka (iz pogođenih particija) može zadržati, a problem je izabrati koja.

Jedan od načina rešavanja ovog problema je potpuna zabrana menjanja podataka u slučaju particionisanja, što, sa druge strane, umanjuje raspoloživost sistema. Ovo se može postići, na primer, upotrebom nekih od protokola konsenzusa kvoruma – ukoliko je sistem particionisan, izmene će moći da se izvrše samo na većoj particiji, jer ona manja neće imati dovoljno čvorova za kvorum.

Kompleksniji načini rešavanja ovog problema uključuju definisanje hijerarhije među redundantnim kopijama podataka (npr. definisanjem primarnih i sekundarnih fragmenata za sve podatke, gde se transakcije izvode samo na primarnim, a dalje se propagiraju na sekundarne) ili vođenje evidencije o verzijama podataka (MVCC – Multiversion Concurrency Control) gde se konflikti mogu automatski prepoznati, ali ne i automatski otkloniti (ovaj pristup je povezan sa sistemom zaključavanja podataka i optimističkim načinom izolovanja transakcija).

242. Šta su "nerelacione baze podataka"?

Pod nerelacionim bazama se najčešće misli na sve one baze podataka, nastale nakon relacionih (dakle isključujući mrežne i hijerarhijske), koje se od relacionih razlikuju po nekim veoma bitnim karakteristikama – npr. nemaju strogu statičku strukturu podataka, nemaju iscrpnu proveru integriteta, ne koristi se upitni jezik SQL, ...

Alternativno, može se reći da su nerelacione baze podataka one baze koje ne počivaju na relacionom modelu podataka, lako se distribuiraju i horizonatlno su skalabilne (laka promena sheme, tj. strukture).

243. Objasniti motivaciju za razvijanje i korišćenje nerelacionih baza podataka.

Iako relacione baze podataka sasvim dobro pokrivaju veliki broj problema za koje je potrebno koristiti bazu podataka, postoje slučajevi kada je neka više specijalizovana, nerelaciona baza podataka bolji izbor. Primeri ovih situacija su potreba za fleksibilnom strukturom podataka (iako se ovo može rešiti zaobilaznim putevima u relacionim bazama, oni nisu baš udobni za rad i efikasni), potreba za distribuiranjem baze podataka (relacione baze se teško distribuiraju, najviše zbog njihovog svojstva da im je konzistentnost podataka primarna), potreba za efikasnijom upotrebom (npr. cena čitanja podataka može biti dosta velika u relacionim bazama zbog velikog broja spajanja) i slično.

Na kraju, postavlja se pitanje – ukoliko za konkretan problem konzistentnost podataka (koja se izuzetno često žrtvuje za odloženu konzistentnost u distribuiranim sistemima) nije primarna, zašto uopšte koristiti relacionu bazu kojoj je jedna od glavnih karakteristika visok stepen integriteta podataka?

244. Koje osnovne slabosti RBP pokušavaju da se prevaziđu nerelacionim bazama podataka?

Neki od glavnih slabosti relacionih baza podataka, većinski uzrokovane njihovom osobinom da na prvo mesto stavljaju integritet podataka, su:

- Nefleksibilna struktura i njena skupa promena
- Teško distribuiranje
- Visoka cena čitanja podataka usled čestog spajanja tabela (koje je potrebno usled neredundantnosti podataka)

245. Navesti osnovne vrste nerelacionih baza podataka i tipične probleme koji se njima rešavaju.

Neki od primera su:

- Podaci imaju složenu strukturu koja odgovara specifičnim domenima – objektne baze podataka mogu biti dobar izbor
- Neophodne su izuzetno visoke performanse – memorijske baze podataka
- Potrebno je skladištiti velike količine podataka niske složenosti (npr. za primene u bioinformatici) – različite vrste matričnih baza podataka
- Potrebno je čuvati dokumente – skladišta dokumenata
- Potreba za lakim distribuiranjem baze – raličite nerelacione, kao npr. baze parova ključeva i vrednosti ili baze sa proširivim slogovima
- Potrebna fleksibilnost u strukturi podataka – slično, veliki broj različitih nerelacionih baza
- ...

246. Navesti najčešće modele podataka nerelacionih baza podataka.

- Skladišta dokumenata,
- Baze parova ključeva i vrednosti (key-value store),
- Baze sa proširivim slogovima (wide-column database),
- Grafovske baze,
- Baze vremenskih serija,
- XML baze,
- Objektno orijentisane baze, ...

247. Baze parova ključeva i vrednosti – karakteristike, doprinosi, slabosti i primeri?

Baze parova ključeva i vrednosti (key-value store) su baze koje svaku kolekciju podataka (u kontekstu relacionih baza, to bi bila tabela) predstavljaju jednom heš tabelom. Ključevi te heš tabele ne mogu biti složene vrednosti (za razliku od primarnih ključeva u relacionim bazama koji se mogu sastojati iz većeg broja atributa), ali vrednosti mogu biti (torka za vrednost).

Ovakva (jednostavna) strukutra omogućava ovim bazama da skladište veoma velike skupove podataka, veliku brzinu pristupa pojedinačnim podacima, kao i jednostavno distribuiranje baze (većina ovih baza omogućava automatsko repliciranje i horizontalno particionisanje kolekcija).

Sa druge strane, i slabosti su brojne. Naime, osnovna mana je što je pretraga moguća isključivo po fiksnoj vrednosti ključa ili rasponu vrednosti ključa, što dalje ima za posledicu potrebu za uvođenjem kolekcija sa redundantnim podacima i drugim ključem radi pretraživanja po tom, novom, ključu. Ovakav pristup garantuje visok nivo redundantnosti. Dodatno, mehanizmi za očuvanje integriteta u ovakvim bazama su najčešće nepostojeći, do te mere da često ne obećavaju ni atomičnost transakcija. Još jedan (u suštini namerni) nedostatak je što jednom ključu odgovara tačno jedna vrednost – ali, ukoliko nam je potrebna mogućnost da ključu odgovara skup vrednosti, možemo jednostavno koristiti baze sa proširivim slogovima.

Neki primeri ovakvih baza podataka su Redis, Memcached, Riak, Oracle NoSQL i mnogi drugi.

248. Baze sa proširivim slogovima – karakteristike, doprinosi, slabosti i primeri?

Baze sa proširivim slogovima (wide-column database) su baze u kojima je svaka kolekcija podataka predstavljena "proširenom" heš tabelom sa slogovima oblika ključ-skup vrednosti. Same vrednosti su predstavljene parom ime-konkretna vrednost. Njihov broj po ključu je, u teoriji, neograničen.

Prednosti koje ovaj tip baza podataka donose su veoma slične kao i u slučaju sa bazama parova ključ-vrednost – moguće skladištenje veoma velikih skupova podataka, velika brzina pristupa pojedinačnim vrednostima (osim ukoliko vrednosti imaju veoma složenu strukturu) i jednostavno distribuiranje (većina ovih baza omogućava automatsko repliciranje i horizontalno particionisanje kolekcija). Velika prednost u odnosu na baze parova ključ-vrednost je u tome što u ovom slučaju jednom ključu može odgovarati skup vrednosti.

Slično kao i u slučaju prednosti, i mane ovih baza su slične onima u slučaju baza parova ključ-vrednost – pretraga samo po fiksnoj vrednosti ključa ili rasponu vrednosti ključa, što dalje ima za posledicu potrebu za uvođenjem kolekcija sa redundantnim podacima i drugim ključem radi pretraživanja po tom, novom, ključu. Ovakav pristup garantuje visok nivo redundantnosti. Dodatno, mehanizmi za očuvanje integriteta u ovakvim bazama su najčešće nepostojeći.

Neki primeri ovih baza su Cassandra, BigTable, Druid, Accumulo i drugi.

249. Baze dokumenata – karakteristike, doprinosi, slabosti i primeri?

Baze dokumenata su strukturno slične bazama parova ključ-vrednost, s tim što su u ovom slučaju vrednosti dokumenti – dakle, svakom dokumentu se pridodaje ključ. Ti dokumenti se mogu zapisivati u strukturiranim (npr. XML, JSON, YAML, ...) ili nestrukturiranim oblicima (npr. PDF). Pored samih dokumenata, čuvaju se i njihovi metapodaci koji su najčešće nestrukturirani ili sa veoma fleksibilnom strukturom. Interno, za potrebe skladištenja u okviru baze, telo dokumenta (ukoliko je taj dokument strukturiran) i njegovi podaci se automatski strukturiraju.

Ovakve baze omogućavaju veoma jednostavan i efikasan rad sa dokumentima, i uz to najčešće podržavaju barem poluautomatsku replikaciju i horizontalno particionisanje kolekcija, pri čemu je replikacija najčešće master-slave replikacija (pisanje se vrši samo na glavnoj replici, nakon čega se izmene propagiraju po ostalima, dok se čitanje može vršiti sa svih replikama).

Međutim, slabost ovih baza je ograničen domen primene – izuzetno su dobre za čuvanje dokumenata, ali ne i za mnogo toga drugog. Pored ovoga, izmena podataka, tj. dokumenata, zavisno od implementacije, često nije podržana.

Neki primeri ovih baza su MongoDB, CouchDB, MarkLogic, RavenDB, Google Cloud Datastore, ...

250. Grafovske baze podataka – karakteristike, doprinosi, slabosti i primeri?

Grafovske baze podataka stavljaju akcenat na odnose između podataka umesto na njihovu strukturu. To se postiže tako što samu bazu čine čvorovi (koji predstavljaju podatke sa veoma slobodnom strukturom) i veze između njih.

Glavna prednost ove vrste baza podataka je u izuzetno velikoj efikasnosti pri obavljanju operacija nad grafovima. Dodatno, zavisno od implementacije, neke baze ovog tipa podržavaju transakcije i ACID uslove, kao i jednostavnu replikaciju, ali najčešće samo master-slave tipa.

Sa druge strane, domen primene ovih baza je izuzetno ograničen – pogodne su samo za čuvanje podataka koji se mogu predstaviti grafovima i vezama između njih.

Neki primeri su Neo4J, OrientDB, GraphDB, ArangoDB, Virtuoso, Titan i druge.

251. Osnovne slabosti nerelacionih baza podataka?

Slabosti nerelacionih baza zavise od konkretnog tipa te baze. Međutim, jedan problem koji se javlja u skoro svim tipovima nerelacionih baza je izuzetno nizak stepen standardizacije i velika razlika između načina upotrebe, čak iako su u pitanju dve nerelacione baze sa istim modelom podataka (npr. dve baze sa parovima ključ-vrednost). Ovo dovodi do velikog problema u izboru konkretne baze podataka za upotrebu, kao i probleme pri prebacivanju sa jedne nerelacione baze na drugu (do te mere da je često lakše prebaciti se na relacionu sa neke nerelacione nego sa nerelacione na drugu nerelacionu sa istim modelom podataka).

Dodatno, nerelacione baze uvek odbacuju neke dobre osobine relacionih, a to često podrazumeva žrtvovanje visokog nivoa integriteta podataka i uvođenje redundantnosti u podatke.

252. Nerelaciona baza podataka Apache Cassandra – osnovne karakteristike.

Apache Cassandra je nerelacioni SUBP otvorenog koda koji počiva na modelu podataka proširivih slogova. Pogledati slajdove... (struktura – kolona, familija kolona, superkolona, familija superkolona, prostor ključeva, upitni jezik – CQL, tipovi podataka...)

253. Objasniti "izolovanje neispravnosti" i "toleranciju neispravnosti".

Dva glavna aspekta staranja o neispravnostima u distribuiranim sistemima su izolovanje neispravnosti i tolerancija neispravnosti.

Izolovanje neispravnosti je obezbeđivanje da neispravnost jedne komponente ne utiče na funkcionalnost drugih komponenti – one će nastaviti svoj rad u onoj meri koliko je to moguće bez neispravne komponente.

Tolerancija neispravnosti predstavlja princip da, u slučaju neispravnosti neke komponente, ostale komponente distribuiranog sistema mogu da preuzmu njenu funkciju i tako omoguće dalji rad sistema. Da bi se tolerancija ostvarila, uvođenje nekog nivoa redundantnosti je neizbežno.

254. Objasniti osnovne aspekte pouzdanosti sistema.

Dva osnovna aspekta pouzdanosti sistema su njegova raspoloživost i njegova odgovornost.

Raspoloživost sistema predstavlja njegovu sposobnost da prihvati zahtev i da obavesti ostatak sistema da je taj zahtev primio.

Odgovornost sistema predstavlja njegovu sposobnost da, nakon prihvatanja zahteva, taj zahtev i obavi.

Moguće je da sistem istovremeno ima visoku odgovornost, a malu raspoloživost (npr. neki čvor ispravno radi, čak i u slučaju retkih izuzetaka, ali veze do njega su prekinute), kao i obrnuto (neki čvor skoro uvek ispravno prihvata zahtev, ali, zbog neispravnosti u softveru, ne uspeva da izvrši zahteve).

255. Objasniti kako se mere osnovni aspekti pouzdanosti sistema.

- Odgovornost sistema – mera odgovornosti je verovatnoća da će sistem odgovoriti na (tj. obaviti) zahteve koje je primio: $R(t)=1-F(t)=1-\int_0^t f(x)\,dx$, gde je R(t) mera odgovornosti u intervalu dužine t, F(t) verovatnoća da će doći do greške u intervalu dužine t, a f(x) gustina raspodele verovatnoće greške. Uobičajena mera je i tzv. srednje vreme do neuspeha, $MTTF=R^{-1}(0.5)$, tj. vreme za koje će verovatnoća greške *R* dostići 0.5. Pored nje, ponekad se koristi i srednje vreme između neuspeha, u oznaci *MTBF*.
- Raspoloživost sistema – mera raspoloživosti, u oznaci *A(t)*, je verovatnoća da će neki sistem biti u stanju da primi zahtev u nekom trenutku *t*, što znači da će sistem ili ispravno raditi u intervalu *[0, t]*, ili će poslednji problem biti otklonjen u nekom trenutku *x*, *0 < x < t*. Dodatno, često se koristi i mera srednjeg vremena oporavka, u oznaci *MTTR*, koja predstavlja očekivano trajanje popravljanja sistema nakon greške. U praksi, često se koristi granična mera raspoloživosti $\lim_{t\to\infty} A(t)=\frac{MTTF}{MTTF+MTTR}$, kao i eksperimentalna mera dobijena za interval *[0, t]* uzorkovanjem podintervala u_i u kojima sistem nije raspoloživ na sledeći način: $A(t)=1-\frac{\sum_i u_i}{t}$.
- Pouzdanost sistema – neka je raspoloživost $A_s(t)$ verovatnoća događaja $I_s(t)$ da će servis biti uspešno iniciran u trenutku *t*, odgovornost $R_s(t, \tau)$ uslovna verovatnoća događaja $T_s(\tau)$ da će sistem uspešno obraditi zahteve na intervalu τ ukoliko je nastupio događaj $I_s(t)$ (zahtev primljen). Tada je pouzdanost sistema verovatnoća $D_s(t, \tau)$ koja predstavlja verovatnoću konjukcije događaja $I_s(t)$ i $T_s(\tau)$, tj. da će zahtev biti uspešno primljen i obrađen.
  - $A_s(t) = P[I_s(t)]$
  - $R_s(t, \tau) = P[T_s(\tau) \mid I_s(t)]$
  - $D_s(t, \tau) = P[I_s(t), T_s(\tau)] = A_s(t)R_s(t, \tau)$ // pretpostavka da su nezavisni događaji

256. Šta je replikacija podataka i koje su njene osnovne karakteristike?

Replikacija podataka je postupak čuvanja više identičnih kopija podataka (tzv. replika) na više lokacija radi postizanja boljih performansi i veće pouzdanosti čitavog sistema. Što se tiče performansi, ona se uglavnom poboljšava u slučaju čitanja (i do n puta efikasnije čitanje za n replika), dok se u slučaju pisanja ona smanjuje (n puta sporije usled potrebe za pisanja u svaku od n replika). Pouzdanost se podiže pomoću mehanizama prepoznavanja neispravnosti i ostvarivanja tolerancije na neispravnosti (njihovim maskiranjem i automatskim rekonfigurisanjem sistema).

257. Objasniti vrste izvora replikacije.

Predmeti replikacije mogu biti podaci, procesi, objekti i poruke, ali od najvećeg interesa je replikacija podataka (čitave baze, tabela, fragmenta tabele, globalnog kataloga baze, ...).

258. Objasniti vrste replikacije.

(???) Sinhrona i asinhrona.

259. Šta je sinhrona replikacija? Karakteristike?

(???)

260. Šta je asinhrona replikacija? Karakteristike?

(???)

261. Koji su osnovni ciljevi replikacije?

Osnovni ciljevi replikacije su podizanje performansi i pouzdanosti sistema. Što se tiče performansi, ona se uglavnom poboljšava u slučaju čitanja (i do n puta efikasnije čitanje za n replika), dok se u slučaju pisanja ona smanjuje (n puta sporije usled potrebe za pisanja u svaku od n replika), mada se ovo do neke mere može zaobići asinhronim strategijama replikacije. Pouzdanost se podiže pomoću mehanizama prepoznavanja neispravnosti i ostvarivanja tolerancije na neispravnosti (njihovim maskiranjem i automatskim rekonfigurisanjem sistema).

262. Koja su ograničenja replikacije?

Osnovna ograničenja replikacije nastaju usled dodatne cene prostora za skladištenje replika, dodatne cene pisanja podataka (sve replike se moraju izmeniti u nekom trenutku), veći obim prenosa podataka kroz mrežu usled replikacije, kao i veća celokupna cena obrade usled održavanja replika.

Dodatno, što se tiče raspoloživosti, važi da je raspoloživost čitavog sistema ograničena raspoloživosti pojedinačnih replika. Konkretna raspoloživost zavisi od modela replikacije (ROWA, konsenzus kvoruma, ...), ali, uopšteno, važi da raspoloživost ažuriranja neke lokacije ne može preći $A^{1/2}$ ukoliko je raspoloživost jedne replike jednaka A.

263. Koji su osnovni načini implementacije replikacije podataka?

ROWA (Read One, Write All), konsenzus kvoruma i konsenzus kvoruma u uređenim mrežama. Za svaki od ovih osnovnih modela replikacije postoje različiti podtipovi.

264. Šta je protokol ROWA? Kako se u osnovi implementira?

ROWA (Read One Write All) je protokol koji određuje pravila čitanja i pisanja u distribuiranom sistemu sa prisustvom replika. Glavna ideja iza ovog protokola je da svaki klijent čita sa samo jedne replike, dok, ukoliko dođe do pisanja, onda se ono vrši nad svim replikama. Postoji više varijanti ovog protokola, svaka sa svojim prednostima i manama.

265. Šta je ROWA? Koje su osnovne varijante ovog protokola?

ROWA (Read One Write All) je protokol koji određuje pravila čitanja i pisanja u distribuiranom sistemu sa prisustvom replika. Varijante, pored osnovnog ROWA protokola, su ROWA-A (sa pisanjem nad raspoloživim replikama), ROWA sa primarnom kopijom i ROWA sa tokenima pravih kopija.

266. Objasniti detaljno osnovni protokol ROWA.

U osnovnoj verziji protokola ROWA, svaka operacija čitanja se prevodi u jednu operaciju čitanja koja se izvršava nad jednom od postojećih replika, pri čemu svaki klijent može kontaktirati bilo koju od replika za čitanje, što čini proces čitanja i do N puta efikasnijim za N replika (globalnim posmatranjem sistema). Sa druge strane, svaka operacija pisanja se prevodi u N operacija pisanja, po jednu za svaku od N replika, pri čemu svaka operacija mora biti izvršena uspešno, ili da ne bude izvršena ni na jednoj replici. Pristup replikama se sinhronizuje pomoću kontrolera konkurentnosti na svakom od čvorova.

U slučaju otkaza nekog od čvorova, ili veze do njega, čitanje je i dalje moguće (sa nekog od drugih čvorova), dok pisanje nije moguće u čitavom sistemu do oporavka svih neispravnih čvorova. Međutim, u trenutku oporavka, nije potrebno vršiti nikakvo dodatno ažuriranje sadržaja jer je pisanje za vreme nedostupnosti čvorova bilo zabranjeno.

267. Objasniti detaljno protokol ROWA-A.

ROWA-A (Read One Write Available) protokol je modifikacija osnovnog ROWA protokola koja rešava problem potpune blokade pisanja usled nedostupnosti jednog čvora. Naime, za razliku od osnovnog ROWA koji zahteva da se pisanje izvrši nad svim replikama, ili ni nad jednom, ROWA-A vrši pisanje nad svim dostupnim replikama. Što ste tiče čitanja, ono funkcioniše isto kao i ROWA – klijent čita podatke sa jedne od dostupnih replika.

Međutim, ovakav način pisanja stvara dodatne probleme u vidu održavanja ažurnosti podataka na replikama. Naime, u slučaju oporavka nekog čvora, vrlo je verovatno da su se u međuvremenu vršila neka pisanja na ostalim replikama, te je sadržaj ovog čvora neažuran i on se ne može odmah vratiti u aktivno stanje za čitanje i pisanje, već se prvo mora izvršiti ažuriranje. Da bi se ovaj način rada omogućio, koristi se dvofazni algoritam validacije za potvrdu transakcija/pisanja:

- Validacija propuštenih pisanja – proverava se da li su čvorovi koji su, usled nedostupnosti, propustili neka pisanja i dalje nedostupni
  - Koordinator šalje specijalnu UNAVAIL poruku svim čvorovima od kojih još uvek nije dobio odgovor na zahtev za pisanje usled nedostupnosti
  - Ukoliko je neki od tih čvorova u međuvremenu postao aktivan, primiće ovu poruku i odgovoriti signalom za prekid transakcije (potrebno je prvo izvršiti ažuriranje sadržaja)
  - Ukoliko ni jedan čvor nije postao aktivan, tj. ukoliko koordinator nije primio signal za prekid transakcije, prelazi se na korak validacije pristupa
- Validacija pristupa – proverava se da li su sve kopije koje su bile dostupne i dalje dostupne
  - Koordinator ovakvim čvorovima šalje specijalnu AVAIL poruku
  - Svaka dostupna kopija prihvata ovu poruku i šalje potvrdu
  - Ako se, nakon nekog vremena, ne primi potvrda, smatra se da je kopija nedostupna i prekida se transakcija (samo za tu, nedostupnu kopiju)

Ipak, proces dvofazne validacije nije preterano efikasan, te je moguć gubitak vremena i značajno umanjenje odzivnosti sistema.

268. Koje su prednosti protokola ROWA-A u odnosu na osnovni protokol ROWA? Ograničenja?

Glavna prednost ROWA-A protokola u odnosu na osnovni ROWA protokol je u tome što operacija pisanja nije potpuno blokirana u slučaju otkaza jednog čvora (vršiće se na svim ostalim čvorovima). Međutim, ROWA-A protokol je relativno neefikasan usled procesa dvofazne validacije, koja, između ostalog, ima i korak čekanja potvrde neko određeno vreme, pa detekcija nedostupnosti neke kopije može biti relativno spora (naručito zato što se, čak iako je neka kopija nedostupna već duže vreme, a možda se uopšte neće ni vratiti u ispravno stanje, u svakom koraku proverava da li je postala dostupna slanjem UNAVAIL poruke i čekanjem na odgovor).

269. // Greška

270. // Greška

271. Objansiti detaljno protokol ROWA sa primarnom kopijom.

Protokol ROWA sa primarnom kopijom, za razliku od osnovnog ROWA protokola i ROWA-A protokola, definiše postojanje tzv. primarne kopije (replike) sa kojom svi klijenti vrše komunikaciju, i prilikom čitanja, i prilikom pisanja (u ostalim protokolima, svaki klijent je mogao da komunicira sa bilo kojom replikom). Čitanje se vrši samo iz primarne kopije, a pisanje se vrši i na primarnoj, a i na svim raspoloživim rezervnim kopijama. Posledica ovoga je da su performanse pisanja smanjene, a performanse čitanja nisu poboljšane, ali, sa druge strane, pouzdanost sistema je unapređena usled postojanja kopija podataka na različitim lokacijama.

U slučaju otkaza primarne kopije, potrebno je proglasiti neku od dostupnih rezervnih kopija za novu primarnu kopiju, što, u zavisnosti od sistema, može biti automatski ili manuelni proces. Ipak, potrebno je biti oprezan prilikom ovoga i razlikovati neoperativnost same primarne kopije i probleme u vezi (npr. u slučaju podele distribuiranog sistema na dve komponente usled prekida veza) da ne bi došlo do situacije da postoji više od jedne primarne kopije, što bi potpuno uništilo konzistentnost sistema.

272. Koje su prednosti protokola ROWA sa primarnom kopijom u odnosu na osnovni protokol ROWA? Ograničenja?

(?) Prednost se ogleda u jednostavnosti ovog protokola, kao i u tome što pisanje nije potpuno blokirano usled otkaza neke kopije (ukoliko rezervna kopija postane nedostupna, svakako se čita samo iz primarne kopije, pa neažurnost rezervne kopije nije problem, ona samo nije kandidat za primarnu kopiju u slučaju njenog otkaza sve dok se ne dovede u ažurno stanje).

Ipak, mana je u tome što su, isto kao i kod osnovnog ROWA algoritma, performanse pisanja smanjene, ali, za razliku od njega, performanse čitanja nisu povećane jer se ono vrši samo iz jedne, primarne kopije. Međutim, pouzdanost sistema je svakako povećana.

273. Objasniti detaljno protokol ROWA sa tokenima "pravih" kopija.

Protokol ROWA sa tokenima koristi koncept postavljanja ekskluzivnih (u slučaju pisanja) i deljenih (u slučaju čitanja) tokena na podacima u okviru kopija. Naime, svaki klijent može komunicirati sa bilo kojom kopijom. Međutim, prilikom pisanja, kopija koja treba da izvrši tu operaciju zahteva ekskluzivni token za konkretni podatak koji piše – ukoliko on već postoji na nekoj drugoj (ili istoj) kopiji za taj podatak, onda se taj token preuzima, a ukoliko ne postoji, lociraju se svi deljeni tokeni za dati podatak, poništavaju se i kreira se novi (jedan) ekskluzivni token za taj podatak umesto njih. U slučaju čitanja, kopija sa koje se vrši čitanje pokušava da locira deljivi token nad istim podatkom i zatim kopira njegovu vrednost, a ukoliko ne postoji deljivi token, onda se locira ekskluzivni token koji se potom konvertuje u deljivi, a onda, isto kao i u prethodnom slučaju, kopira. Ukoliko je to potrebno, uzimanjem tokena se vrši i ažuriranje kopije pre izvršenja same operacije.

Važno je napomenuti da je podatak, ukoliko se svi tokeni nad njim nalaze na nedostupnim čvorovima, nedostupan. Ponovnom aktivacijom nedostupnog čvora, svi ekskluzivni tokeni koje je taj čvor držao se zadržavaju (jer ti podaci u međuvremenu nisu mogli biti ažurirani jer ekskluzivni token nad njima nije mogao biti preuzet), a svi deljeni tokeni se, ukoliko oni postoje i na nekoj drugoj, operativnoj lokaciji (što znači da data kopija potencijalno više nije ažurna), poništavaju.

Dodatno, distribuiranje promena nakon pisanja se može vršiti i asinhronim procesom koji obilazi sve ekskluzivne tokene, konvertuje ih u deljene i zatim kopira izmenjene podatke, zajedno sa deljenim tokenima, na još neke dodatne lokacije (naravno, postoje i drugačiji mehanizmi).

274. Koji od protokola ROWA se najčešće upotrebljava u praksi? Zašto?

Iako jednostavan i sa, naizgled, mnoštvom mana (u vidu smanjene efikasnosti pisanja i nepromenjene efikasnosti čitanja), u praksi se veoma često koristi ROWA sa primarnom kopijom. Naime, u velikom broju slučajeva je primarna svrha distribuiranog sistema podizanje nivoa pouzdanosti, a ROWA sa primarnom kopijom pruža upravo to, i to na relativno jednostavan način.

275. Šta je konsenzus kvoruma? Po čemu se razlikuje od protokola ROWA?

Konsenzus kvoruma predstavlja skup protokola za čitanje i pisanje u okviru sistema sa prisustvom repliciranih podataka. Glavna ideja, a ujednom i razlika od ROWA protokola, je u tome što se vrši podela kopija na podskup za pisanje (kvorum pisanja) i podskup za čitanje (kvorum čitanja), ali na taj način da ovi podskupovi imaju neprazan i dostupan presek, što osigurava da će svako čitanje moći da se odvija sa ažurnim podacima (ustanovljavanje ažurne kopije iz kvoruma čitanja se vrši na osnovu oznake verzije ili timestamp-ova poslednjeg pisanja – uvek se bira onaj sa najnovijim). Način izbora ovih podskupova može biti statički (određeni glasanjem pri podizanju sistema) ili dinamički (moguće rekonfigurisanje u toku rada sistema).

Dodatno, sprovođenje ažuriranja kroz ostale čvorove (van kvoruma pisanja) se vrši asinhronim procesom, pa nisu potrebni nikakvi dodatni postupci u slučaju aktivacije prethodno nedostupnog čvora (on će, vremenom, biti ažuran usled izvršavanja asinhronog procesa).

276. Opisati opšte karakteristike konsenzusa kvoruma.

Protokoli konsenzusa kvoruma omogućavaju relativno efikasno, bar u odnosu na ROWA protokole, pisanje (usled toga što se pisanje, van asinhronog procesa, vrši samo na podskupu svih kopija), ali i ne toliko efikasno čitanje (pri svakom čitanju je potrebno proveriti oznake verzije ili timestamp-ove svih kopija u okviru kvoruma čitanja). Dodatno, ponovna aktivacija čvora u slučaju prethodne nedostupnosti je, načelno, jednostavna i efikasna (ne zahteva nikakve dodatne postupke poput dvofazne validacije).

277. Koje su vrste konsenzusa kvoruma? U čemu je osnovna razlika među vrstama?

- Konsenzus kvoruma sa uniformnom većinom
- Konsenzus kvoruma sa težinskom vežinom
- Konsenzus kvoruma sa težinskom većinom za direktorijume
- Uopšteni konsenzus kvoruma za apstraktne tipove podataka
- Hibridni protokol ROWA/konsenzus kvoruma

278. Objasniti detaljno konsenzus kvoruma sa uniformnom većinom.

Protokol konsenzusa kvoruma sa uniformnom većinom nalaže da operacija čitanja i operacija pisanja mogu biti uspešne akko većina čvorova (tj. kopija, replika) odobri tu operaciju. Ovo suštinski znači da skup svih kopija u sistemu mora biti podeljen na dva podskupa – kvorum za čitanje i kvorum sa pisanje, s tim da oba ova podskupa sadrže barem većinu od svih kopija (ukoliko je ukupan broj kopija n, tada mora važiti $|R| >= n / 2 + 1$ i $|W| >= n / 2 + 1$). Ovaj uslov osigurava da će se bar jedna kopija sa ažurnim podacima nalaziti u preseku ova dva kvoruma.

Operacija čitanja se vrši samo nad jednom kopijom iz kvoruma za čitanje, i to onom sa najažurnijim podacima. Međutim, ovo podrazumeva proveru ažurnosti između svih članova kvoruma, pa je efikasnost donekle smanjena. Ažurnost se obično ustanovljava na osnovu oznake verzije ili timestamp-a.

Operacija pisanja se vrši nad svim kopijama u kvorumu za pisanje, što je, iako efikasnije nego u ROWA protokolu, i dalje poprilično neefikasno. Izmene se mogu propagirati i na čvorove van kvoruma za pisanje asinhronim procesom.

Ovakvim postupkom je postignuta otpornost na neispravnosti, kako u samim lokacijama, tako i u slučaju prekida u komunikaciji i particionisanja mreže (sve dok postoji particija u okviru koje može da se komunicira i koja sadrži većinu čvorova).

279. Objasniti detaljno konsenzus kvoruma sa težinskom većinom.

Konsenzus kvoruma sa težinskom većinom predstavlja uopštenje uniformne verzije ovog protokola. Naime, kopijama (replikama) podataka, u oznaci *d*, se dodeljuju određene težine, u oznaci d_s. Pored toga, podatku *d* se dodeljuju "pragovi" *r* i *w*, koji, redom, predstavljaju prag čitanja i prag pisanja. Operacija čitanja, odnosno pisanja, se onda smatra uspešnom ukoliko je suma svih težina nad dostupnim kopijama podatka *d* veća ili jednaka *r*, odnosno *w*. Skupovi koji zadovoljavaju ove pragove formiraju kvorume.

Da bi se osigurala konzistentnost, potrebno je da važi uslov $r + w > u$, gde je *u* suma svih težina kopija d_s nad podatkom *d*. Ovaj uslov je ekvivalentan uslovu nepraznog preseka kvoruma kod uniformne verzije protokola. Dodatno, i to samo ukoliko se za označavanje ažurnosti kopija koriste oznake verzije, a ne timestamp-ovi, mora važiti i $w > u / 2$.

Operacija čitanja se prevodi u operaciju čitanja nad svim kopijama u nekom kvorumu čitanja, ali se kao rezultat vraća samo ona kopija sa najvećim brojem verzije.

Operacija pisanja se prevodi u operaciju pisanja nad svim kopijama u nekom kvorumu pisanja. Ovo, između ostalog, podrazumeva čitanje verzije svih kopija u tom kvorumu i postavljanje nove verzije na vrednost jednaku $nova\_verzija = max(stare\_verzije) + 1$.

Glavna prednost ovog protokola u odnosu na uniformnu verziju je njegova fleksibilnost – u zavisnosti od raspoloživosti svake od lokacija, parametri *r*, *w*, kao i težine d_s, mogu se menjati u svrhu maksimizacije raspoloživosti celog sistema. Ovo se često radi metaheurističkim algoritmima optimizacije, kao npr. pomoću simuliranog kaljenja. Dodatno, ukoliko se za paremetre uzmu vrednosti $w = u$, $r \leq min(d_s)$ (svi moraju biti raspoloživi da bi se pisalo, može se čitati iz bilo kog – mislim da je greška u slajdovima, obrnuto w i r), protokol se svodi na osnovni ROWA protokol, a u slučaju izbora jednakih težina d_s, on se svodi na uniformni konsenzus kvoruma.

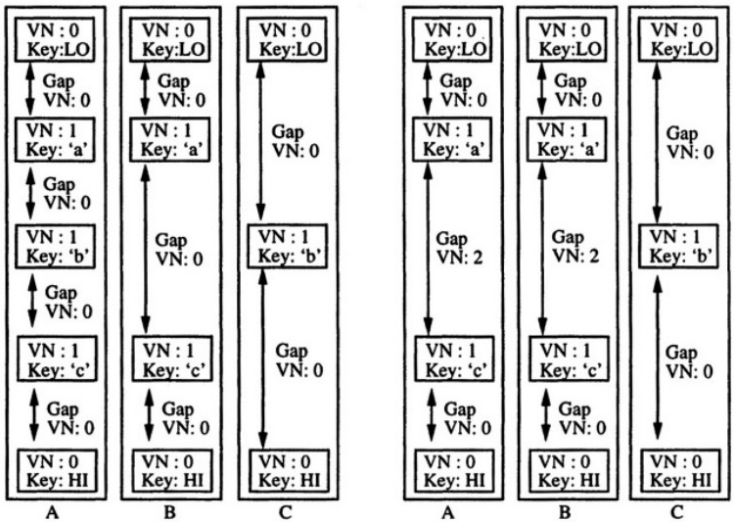
280. Objasniti razlike između konsenzusa kvoruma sa uniformnom većinom i konsenzusa kvoruma sa težinskom većinom.

Konsenzus kvoruma sa težinskom većinom predstavlja uopštenje konsenzusa kvoruma sa uniformnom većinom. Naime, u težinskoj verziji se svakoj kopiji nekog podatka dodeljuje određena težina d_s, dok bi, u slučaju uniformne verzije, te težine bile jednake. Ovakav pristup omogućava mnogo veću fleksibilnost težinske verzije protokola, što doprinosi boljim performansama usled bolje prilagođenosti sistema stvarnim potrebama (na primer, možemo čak i svesti protokol na običan ROWA ukoliko nam je potrebno efikasno čitanje).

281. Koje su specifičnosti konsenzusa kvoruma za direktorijume i apstraktne tipove podataka?

U slučaju direktorijuma, tj. preslikavanja prostora ključeva u prostor vrednosti (suštinski mapa) sa definisanim operacijama *Insert(Key k, Value v)*, *Update(Key k, Value v)*, *Delete(Key k)* i *Value Lookup(Key k)*, problematično je koristiti samo jednu oznaku verzije (ili timestamp) za praćenje ažurnosti podataka čitavog direktorijuma. Ovo smeta konkurentnosti prilikom izvršavanja operacija (zato što čitav direktorijum ima samo jednu pridodatu verziju, operacije se moraju izvršavati serijski). Rešenje za to je promena granulacije operacija zasebnim verzionisanjem delova direktorijuma.

Međutim, ovde je moguć problem usled brisanja ključeva. Naime, ukoliko se neki ključ obriše u nekoj kopiji, a u nekoj drugoj još uvek ne, kako ustanoviti koja je najnovija verzija kada su sve informacije, uključujući i verziju, obrisane prilikom brisanja ključa? Rešenje za ovo je ili praćenje verzija za sve moguće ključeve, čak i one koji nisu u direktorijumu, ili, efikasnije od toga, praćenje "razmaka" između ključeva u direktorijumu, pod pretpostavkom da su oni sortirani.



Što se tiče konsenzusa kvoruma za apstraktne tipove podataka, radi postizanja veće fleksibilnosti u zavisnosti od konkretnog tipa podataka, umesto replikacije samih podataka, repliciraju se "događaji" kojima se dati podaci menjaju, pa se kvorumi kreiraju tako da mogu da formiraju celokupnu istoriju izmena podataka.

282. Objasniti detaljno hibridni protokol ROWA/konsenzus kvoruma. Objasniti motivaciju za njegovu primenu.

Glavni problem protokola konsenzusa kvoruma je cena čitanja visoka, čak i onda kada sistem radi bez ikakvih smetnji. Naime, prilikom svakog čitanja je potrebno u najmanju ruku proveriti sve kopije u okviru kvoruma za čitanje i pročitati njihovu verziju, pa pročitati onu sa najvećom. Iz tog razloga se uvodi hibridni protokol ROWA/konsenzus kvoruma.

Osnovna ideja ovog protokola je da se sistem, u slučaju kada je potpuno raspoloživ, ponaša u skladu sa ROWA protokolom, ali u trenutku kada se detektuje otkaz u sistemu, prelazi se u režim rada po protokolu konsenzusa kvoruma. Ovime se postiže efikasno čitanje ROWA protokola (barem kada sistem radi potpuno ispravno), a izbegava se njegova mana neraspoloživosti za pisanje usled greške prelaskom u KK režim.

Da bi se ovaj način rada realizovao, potreban je način za prepoznavanje otkaza čvora. Ovo se izvodi kroz prepoznavanje tzv. propuštenih pisanja. Postoje dve situacije u kojima jedna transakcija T može da prepozna propušteno pisanje, a to su:

- Nakon čitanja neke kopije podataka d_s, što se relativno jednostavno može detektovati time što transakcija neće dobiti potvrdu o validnosti od strane lokacije *s* u određenom vremenskom intervalu. U ovom slučaju, transakcija T se prekida.
- Pre čitanja neke kopije podataka d_s, za šta bi, efektivno, transakcija T morala biti obaveštena o propuštenom pisanju od strane neke druge transakcije T', u kom slučaju prelazi u režim konsenzusa kvoruma, gde kvorumi sigurno obuhvataju bar jednu lokaciju koja nije propustila pisanje. Da bi se ovakav način obaveštavanja od strane druge transakcije omogućio, uz svaku kopiju podataka se vodi i lista propuštenih pisanja (MWL) u koju bi, u ovom slučaju, transakcija T' ostavila informaciju o propuštenom pisanju, a transakcija T bi zatim tu informaciju mogla da dobije iz liste i propagira je i na ostale kopije podataka koje posećuje.

Prelazak iz ROWA režima u KK režim je relativno jednostavan, ali obratno i nije zbog održavanja listi propuštenih pisanja (MWL-ova).

283. Šta je konsenzus kvoruma u uređenim mrežama? Motivacija?

Konsenzus kvoruma u uređenim mrežama predstavlja familiju protokola koja uvodi koncepte kvoruma u, na neki način, strukturirane mreže lokacija (replika, kopija). Ideja iza ovog pristupa je da se pažljivim uređenjem mreže lokacija i izborom kvoruma, veličina tih kvoruma može smanjiti, u slučaju neuređenih mreža, sa $n / 2 + 1$ čvora na mnogo manje veličine, čime bi se, pored pouzdanosti sistema, smanjila i neefikasnost usled veće cene čitanja i pisanja zbog uvedene redundantnosti.

284. Koji su osnovni algoritmi koji se upotrebljavaju u konsenzusu kvoruma u uređenim mrežama?

Osnovni algoritmi su:

- Algoritam √n
- Protokol matrice (GRID)
- Algoritmi asimptotski visoke raspoloživosti
- Drvo kvoruma
- Konsenzus kvoruma sa hijerarhijskom težinskom većinom
- Konsenzus kvoruma sa višedimenzionalnom težinskom većinom

285. Objasniti algoritam √n. Koje su mu osnovne karakteristike? Zašto se tako zove?

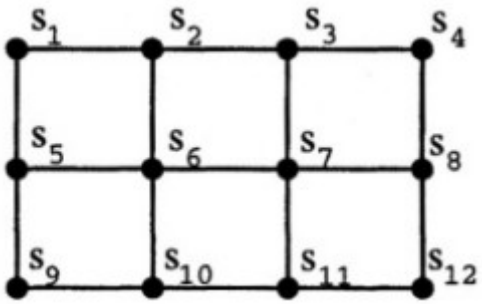
Algoritam √n je jedan od načina određivanja kvoruma u uređenim mrežama. Naime, ideja je da se svakoj lokaciji s_i dodeli kvorum (koji se može koristiti i za čitanje, i za pisanje) Q_i, ali, da bi se postiglo uzajamno isključenje (u konkurentnom radu), na takav način da važi:

- $Q_i \cap Q_j \neq \varnothing$ – svaka dva kvoruma moraju imati neprazan presek koji služi kao arbitar
- $s_i \in Q_i$ – lokacija s_i automatski dobija dozvolu pristupa od sebe, bez slanja ikakvih dodatnih poruka
- $|Q_i| = K$ – veličina svakog kvoruma je ista, što znači da svaka lokacija mora da pošalje i primi isti broj poruka da bi se postiglo uzajamno isključenje
- $|\{Q : s_i \in Q\}| = M$ – svaka lokacija je arbitar za isti broj lokacija, što znači da svaka lokacija učestvuje u istom broju kvoruma

Iz prethodnih uslova se može dokazati da važi $n = K(K – 1) + 1$, gde je n ukupan broj lokacija, kao i to da se uzajamno isključenje postiže sa $c\sqrt{n}$ poruka, gde je $3 \leq c \leq 5$. Dodatno, može se dokazati da je optimalna (minimalna) veličina kvoruma K jednaka upravo √n, odakle potiče i naziv ovog algoritma.

Najveća prednost ovog algoritma je relativno mala (korenska) veličina kvoruma, naručito u odnosu na standardni konsenzus kvoruma koji zahteva veličinu od bar $n / 2 + 1$. Međutim, pretpostavka algoritma je da ne može doći do greške u mrežnoj komunikaciji, jer bi particionisanje mreže prouzrokovalo potpuni kolaps u kvorumima dobijenim ovim algoritmom.

286. Objasniti protokol GRID. Koje su mu osnovne karakteristike?



Protokol GRID je jedan od mogućih protokola koji se mogu koristiti za implementaciju konsenzusa kvoruma u uređenim mrežama. Ideja je da se skup od *n* lokacija koje čuvaju replike uredi u matricu dimenzija *MxN*, pri čemu važi $M*N \leq n$. U takvom uređenju, kvorumi čitanja se dobijaju uzimanjem po jedne lokacije iz svake kolone matrice, dok se kvorumi pisanja dobijaju unijom nekog kvoruma čitanja i nasumično odabrane čitave kolone matrice. Primetimo da ovakvim izborom kvoruma uvek postoji neprazan presek između svakog kvoruma pisanja i svakog kvoruma čitanja u bar jednoj lokaciji. Oni, takođe, formiraju tzv. C-pokrivače, tj. skupove lokacija takve da svaka kolona matrice ima neprazan presek sa njima (u prevodu, imamo bar jedan element iz svake kolone).

Za operaciju čitanja se, redom, počev od nasumičnog reda *r*, u nasumičnom redosledu kolona (određenim nasumičnom permutacijom lokacija tog reda), pokušava sa postavljanjem katanaca za čitanje za svaki od elemenata tog reda. Ukoliko se ne uspe sa dobijanjem katanaca, prelazi se na naredni red, a ukoliko ne uspe ni za jedan red, operacija se prekida. Međutim, ukoliko se dobijanje katanaca obavi uspešno, oni formiraju jedan C-pokrivač i time garantuju da jedna od tih kopija ima ažurnu vrednost. Prethodno uvedena nasumičnost u odabiru reda i redosleda zaključavanja služi za raspodelu opterećenja na lokacijama.

Za operaciju pisanja se prvo zaključava C-pokrivač dobijen kao za operaciju čitanja. Nakon toga, zaključavaju se sve lokacije unutar nasumične kolone i to u nasumičnom redosledu (određenim nasumičnom permutacijom lokacija te kolone). Ovakav pristup garantuje da, ukoliko se pokuša sa dve konkurentne operacije pisanja ili konkurentnim operacijama pisanja i čitanja, sigurno postoji presek u C-pokrivačima katanaca dobijenim iz te dve operacije, što osigurava da će konflikt biti detektovan i sprečen jer druga operacija neće moći da dobije katanac na konfliktnoj lokaciji. Slično kao i kod čitanja, nasumičan odabir kolone i redosleda služi za raspodelu opterećenja.

Ovaj protokol karakteriše relativno mala veličina kvoruma (asimptotski $O(\sqrt{n})$ jer se iz svake kolone uzima samo po jedan element za kvorum čitanja, a za kvorum pisanja se na to dodaje i jedna čitava kolona), dobra raspodela opterećenja po čvorovima usled nasumičnog odabira pri zaključavanju, ali isto tako i pretpostavka da ne može doći do greške u mrežnoj komunikaciji, jer bi particionisanje mreže prouzrokovalo potpuni kolaps u kvorumima dobijenim ovim algoritmom.

287. Šta su sistemi za podršku odlučivanju?

Sistemi za podršku odlučivanju su sistemi koji imaju mogućnost pružanja informacija od strateškog značaja (npr. izbor najbolje poslovne odluke, koje proizvode ukloniti iz prodavnice, koje dodati, ...) na osnovu analize velikog broja prikupljenih kolekcija podataka.

288. Šta su skladišta podataka? Po čemu se razlikuju od uobičajenih baza podatka za obradu transakcija?

Skladišta podataka su baze podataka specijalno projektovane i namenjene u svrhe upotrebe za podršku odlučivanju. Kao takve, ne koriste se za transakcionu i operativnu upotrebu, već za analitičku obradu (većinski) istorijskih podataka (za razliku od operativnih kod standardnih transakcionih baza).

Ona su primarno namenjena za čitanje i (intenzivnu) analizu podataka koju sadrže, ali i retko ažuriranje (periodično ažuriranje i dopunjavanje skupa podataka), pa su iz tog razloga podaci najčešće denormalizovani, što uvodi redundantnost, ali smanjuje potrebu za njihovim spajanjem, čime se povećava efikasnost čitanja.

Zbog kombinacije čuvanja istorijskih i redundantnih podataka, veličina (u kontekstu memorijskog zauzeća) ovih baza je najčešće ogromna – i do više redova jedinica veća od standardnih transakcionih baza podataka (petabajti i terabajti naspram gigabajta).

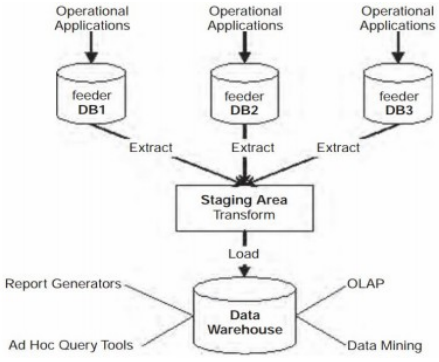
Pored ovoga, ovim bazama najčešće pristupa samo nekolicina korisnika i nad njom postavljaju poprilično složene i obimne upite (za razliku od hiljada korisnika sa jednostavnijim upitima u slučaju standardnih baza).

289. Navesti i objasniti specifične zahteve pri projektovanju skladišta podataka.

Sami koraci projektovanja skladišta podataka dosta podsećaju na standardne korake pri projektovanju baze podataka (prikupljanje podataka, logičko i fizičko projektovanje), međutim, pri izvođenju ovih koraka je potrebno uzeti neke specifične zahteve u obzir:

- Podaci se organizuju tematski – ove teme najčešće odgovaraju funkcijama informacionog sistema (npr. polagani ispiti, upisani predmeti, ...). Svaka od ovih tematskih oblasti se najčešće zasebno projektuje i implementira, mada, i pored ovoga, neke teme mogu imati deljene podatke u vidu dimenzionih tabela (ali centralni podaci za temu su po pravilu nedeljeni).
- Mogućnost integrisanja podataka – iako su tematske oblasti najčešće odvojene, često je korisno imati mogućnost integrisanja, kako tema (radi izvođenja složenijih analiza koje zahtevaju podatke iz više tema), tako i izvora (podaci se sakupljaju iz više različitih izvora i integrišu u celinu). Ovo se najčešće izvodi ili implementacijom svih celina u jednoj, centralnoj bazi podataka, ili se vrši federativno integrisanje.
- Podaci su nepromenljivi tokom obrade – podaci se, van periodičnih i masovnih ažuriranja, ne menjaju. Iz tog razloga, tokom obrade u skladištima podataka najčešće nije potrebno voditi računa o transakcijama (ACID osobine) kao u standardnim bazama.
- Ažuriranje se odvija periodično i u masovnim paketima.
- Podaci su često potrebni na različitim nivoima granularnosti – radi veće efikasnosti analiza, često se čuvaju redundantni podaci na više nivoa granularnosti (npr. zasebno se čuvaju timestamp, dan, mesec i godina čak iako bi se svi oni mogli, po nekoj ceni, izvući iz timestamp-a).
- Fleksibilnost u odnosu na zahteve i ciljeve – zbog dinamične prirode poslovnih okruženja, poželjno je da postoji određeni nivo fleksibilnosti skladišta podataka. Ovo najčešće podrazumeva mogućnost dodavanja novih tema, a ređe i dodavanje novih kolona u postojeće tabele (ovo je često veoma kompleksan zadatak jer je za sve već postojeće podatke, tj. redove, potrebno pronaći i odgovarajće podatke za novu kolonu).
- Mogućnost "menjanja istorije" i "buduće istorije" radi analiza tipa "šta ako" – izuzetno koristan vid analize u poslovnim okruženjima je analiza koja odgovara na pitanje "šta ako". Potrebna je mogućnost zamene već postojećih podataka ili dodavanje novih "lažnih" podataka radi sprovođenja takve analize. Ovo se često postiže horizontalnim particionisanjem, npr. po vremenskom periodu (npr. za svaki mesec).
- Specifične korisničke aplikacije sa odgovarajućim korisničkim interfejsom.

290. Objasniti osnovnu arhitekturu skladišta podataka.

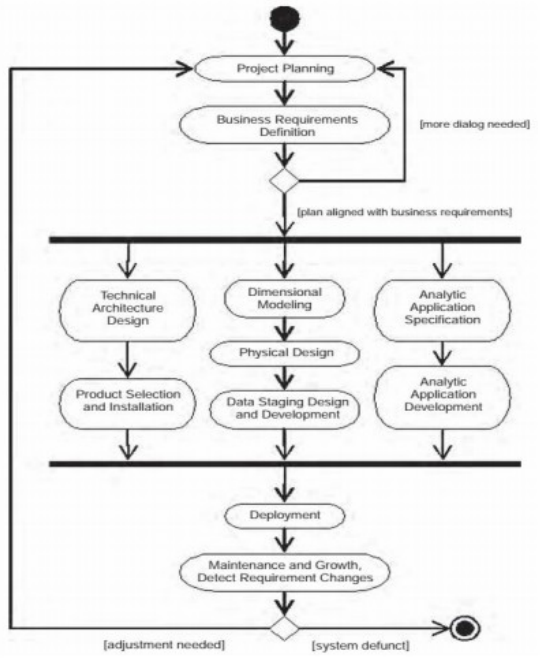


Skladište podataka se obično "puni" iz više standardnih (transakcionih) baza podataka. Podaci iz tih baza se prvo prečišćavaju od nepouzdanih i nerelevantnih podataka, nakon čega se po potrebi spajaju i transformišu u odgovarajući oblik pre punjenja samog skladišta podataka.

Nakon punjenja, ažuriraju se pomoćne tabele (npr. materijalizovani pogledi) u okviru skladišta podataka. Punjenje i ažuriranje skladišta se izvršava periodično i sa velikim količinama podataka, veoma često uz pomoć mehanizama za osvežavanje koji prate stanje u transakcionim bazama podataka i iniciraju ceo ovaj proces po potrebi (npr. u određenim vremenskim intervalima ili nakon određenog obima podataka).

Nad podacima u skladištu se zatim koristi onlajn analitička obrada (u suštini, polu-interaktivno postavljanje složenih upita), kao i tehnike istraživanja podataka. Sve to je podržano alatima koji pomažu u vršenju analize podataka i generisanju izveštaja.

291. Opisati i objasniti životni ciklus skladišta podataka.



Životni ciklus jednog skladišta podataka je zapravo dosta sličan životnom ciklusu standardnih transakcionih baza podataka. On se može podeliti u naredne osnovne korake:

- Prikupljanje i analiza zahteva – ovde je možda i glavna razlika u odnosu na standardni pristup kod baza podataka. Naime, korak konceptualnog projektovanja se suštinski preskače, tj. utapa se u korak logičkog projektovanja, iz razloga što su skladišta podataka organizovana po tematskim celinama, a ne po spoljnim shemama koje će određene vrste korisnika upotrebljavati. Umesto njega imamo korak prikupljanja i analize zahteva.
- Logičko projektovanje
- Fizičko projektovanje
- Distribuiranje podataka
- Implementacija, praćenje i menjanje

292. Šta je zvezdasta shema? Kada se, zašto i kako koristi?

Zvezdasta shema predstavlja osnovni oblik logičkog modela skladišta podataka dobijen u koraku dimenzionog modeliranja. Naime, ideja je da se svi podaci koji su neophodni za analize nad nekom tematskom celinom izdvoje u jednu veliku, centralnu tabelu, tzv. "tabelu činjenica", dok se svi ostali podaci, oni koji nisu bitni za direktnu analizu, ali jesu za konačni izveštaj, izmeštaju u tzv. "dimenzione tabele", gde će se na odgovarajuće redove iz nje referisati iz tabele činjenica. Ni tabela činjenica, ni dimenzione tabele u ovom slučaju nisu normalizovane.

Jedan od glavnih razloga za ovaj pristup je ušteda u memorijskom prostoru. Naime, ukoliko se neki podaci neće koristiti u okviru samih analiza, već samo na kraju, prilikom pravljenja izveštaja, zašto ih redundantno čuvati u stotinama hiljada ili milionima redova u tabeli činjenica? Ti podaci neće omogućiti efikasniju analizu, već će samo nepotrebno zauzimati prostor. Pored ovoga, prostor će se uštedeti i u slučajevima kada više različitih tabela činjenica u okviru istog skladišta podataka može da referiše na deljene dimenzione tabele.

Primer bi bio analiza polaganih ispita na nekom univerzitetu (sačinjenom od više fakulteta). Centralna tabela bi definitivno trebala da sadrži podatke o predmetu, ispitu, fakultetu, studentu. Međutim, da li su ime fakulteta, njegova adresa i neki ostali, sekundardni podaci bitni za ovakve analize, ili je dovoljan sam identifikator fakulteta? Verovatno važi ono drugo, te je poželjno izdvojiti te sekundarne podatke u dimenzione tabele. Slično važi i za podatke o predmetu i studentu.

293. Šta je shema pahuljice? Kada se, zašto i kako koristi?

Potpuno isto kao pitanje 292. (zvezdasta shema), samo što su dimenzione tabele u ovom slučaju normalizovane (ne moraju baš sve biti).

294. Objasniti specifičnosti fizičkog modeliranja skladišta podataka.

Specifičnosti fizičkog modeliranja skladišta podataka se, između ostalog, velikim delom ogledaju u naredna tri aspekta:

- Projektovanje indeksa
  - Za dimenzione tabele se najčešće koriste heš indeksi zbog izuzetno efikasnog spajanja tabela sa njihovom upotrebom (dimenzione tabele se uglavnom ne koriste u samoj analizi, već samo za pravljenje krajnjeg izveštaja, pa je zato najvažnija brzina njihovog spajanja sa tabelom činjenica).
  - Za tabele činjenica se najčešće koriste bit-mapirani indeksi iz razloga što omogućavaju izuzetno efikasno spajanje indeksa (npr. konjukcijom ili disjunkcijom, zavisno od uslova u upitu).
  - Poželjan broj indeksa je suštinski neograničen (za razliku od transakcionih baza podataka gde je ovaj broj obično 3 do 5 indeksa) zato što su čitanja izuzetno česta, a ažuriranja retka i periodična, pa ažuriranje indeksa ne pravi veliki problem.
- Projektovanje materijalizovanih pogleda
  - Materijalizovani pogled predstavlja tabelu čija su struktura i sadržaj definisani nekim upitom. U skladu sa tim, materijalizovani pogledi se izuzetno često koriste u skladištima podataka, najčešće u obliku tzv. "zbirnih tabela". U pitanju su tabele, implementirane preko materijalizovanih pogleda, koje na različitim nivoima granularnosti održavaju određene statistike podataka.
  - Materijalizovani pogledi se najčešće izračunavaju ili po prvom izvršavanju nekog upita, ili automatski po punjenju skladišta podataka.
- Projektovanje particija
  - Horizontalno particionisanje se najčešće vrši po grupama redova, npr. po lokacijama ili vremenskim periodima (ispiti polagani 2018., 2019., ...). Čak i ukoliko se ne vrši distrubiranje podataka, horizontalno particionisanje je izuzetno poželjno kod skladišta podataka radi lakog "uključivanja i isključivanja" particija i njihove zamene sa alternativnim podacima (u svrhe "šta ako" analize).
  - Vertikalno particionisanje se u skladištima podataka relativno retko sprovodi, i to u slučajevima kada je veličina podataka preobimna za jednu celovitu, neparticionisanu tabelu. Kada se ono ipak radi, to je obično po grupama kolona sličnih karakteristika.

Dodatno, što se tiče distribuiranja skladišta podataka – težnja je da skladište ne bude distribuirano, već centralizovano radi što veće efikasnosti čitanja. Ukoliko je distribuiranje ipak neophodno zbog veličine skladišta podataka, poželjno je da to bude na čvorovima (računarima) u jednom centru (zbog brže komunikacije).

295. Šta je onlajn analitička obrada? Objasniti specifičnosti.

Onlajn analitička obrada, tj. OLAP, u širem smislu označava "polu-interaktivno" (ukoliko je potrebno nezanemarljivo mnogo vremena za izvršavanje upita, to ne možemo baš smatrati potpunom interaktivnošću) zadavanje upita ili upotrebu alata radi vršenja analitičke obrade podataka (donošenje zaključaka na osnovu "sirovog" skupa podataka). OLAP se može posmatrati i u užem smislu gde označava analitičku obradu koja se izvodi nad "živim" podacima, tj. nad bazom u produkciji.

Važno je naglasiti i specifičnosti OLAP-a nad skladištima podataka u odnosu na standardne transakcione baze podataka. Naime, OLAP u skladištima podataka se može vršiti bez ikakvog izolovanja upita zbog pretpostavke retkog i periodičnog ažuriranja podataka, vrši se nad istorijskim (za razliku od samo operativnih) podacima i to na mnogo većem skupu njih. Dodatno, struktura samog skladišta podataka je, uvođenjem velikog stepena redundantnosti, prilagođena OLAP-u.

296. Šta su "automatske zbirne tabele"? Objasniti namenu i način funkcionisanja.

Automatske zbirne tabele su materijalizovani pogledi različitih nivoa granularnosti koji služe za čuvanje nekih čestih statistika u svrhe povećane efikasnosti prilikom analize podataka (ukoliko neke analize zahtevaju određene statistike, zašto ih računati iznova svaki put?).

Pošto automatske zbirne tabele najčešće postaju neažurne nakon (retke i periodične) promene podataka u skladištu, potrebno ih je ažurirati – ili prilikom punjenja skladišta podacima, ili prilikom prve naredne upotrebe automatske zbirne tabele.

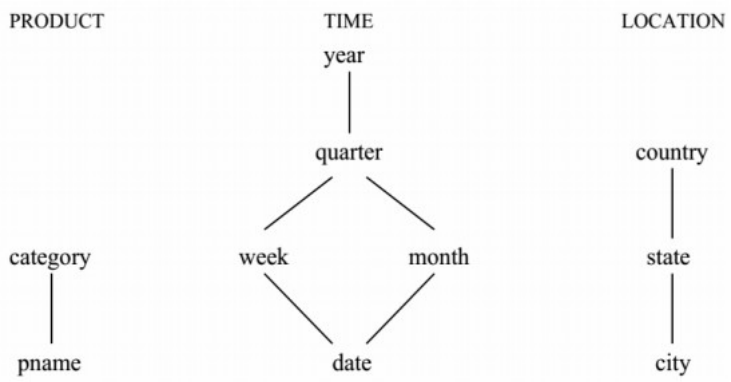
Zbog ogromnog broja mogućih pogleda, tj. automatskih zbirnih tabela koje se mogu napraviti usled većih nivoa granularnosti i većeg broja elemenata nad kojima se može vršiti grupisanje (eksplozija broja pogleda), najčešće se koriste heuristički algoritmi, kao npr. HRU ili PGA, za automatski odabir pogodnih zbirnih tabela.

Sledi primer AZT – neka je dat skup podataka o prodatim proizvodima u nekoj onlajn prodavnici, zajedno sa tačnim datumima prodaje. Može se napraviti zbirna tabela vrednosti prodaje po danima, pa na osnovu nje po mesecima, pa na osnovu nje po godinama, itd.

297. Objasniti eksploziju broja pogleda kod OLAP sistema.

Eksplozija broja pogleda se odnosi na kombinatornu eksploziju svih mogućih grupa atributa (dimenzija) nad kojima se mogu kreirati agregirani materijalizovani pogledi.

Naime, ukoliko je broj dimenzija jednak d (npr. proizvod, vreme, lokacija), a h_i broj nivoa i-te dimenzije (npr. za vreme datum, nedelja, mesec, godina), onda je ukupan broj mogućih pogleda jednak $N = \prod_{i=1}^{d} h_i$.



Zbog potencijalno ogromnog broja N za veće tabele činjenica (centralne tabele u skladištima podataka), često je nemoguće praviti i održavati sve moguće materijalizovane poglede, pa se zato koriste heuristički algoritmi (npr. HRU ili PGA) za izbor podskupa njih koji donose što veću uštedu u broju redova materijalizovanih pogleda.

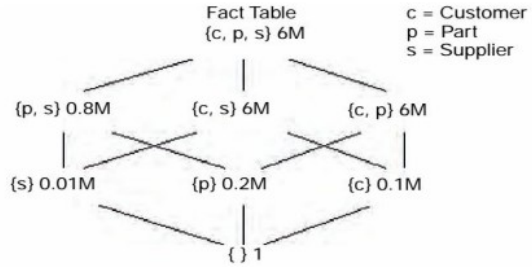
298. Algoritam HRU odabira materijalizovanih pogleda.

Algoritam HRU je heuristički algoritam odabira materijalizovanih pogleda koji radi dobro za mali broj dimenzija (npr. vreme, lokacija, ...).

Ideja iza algoritma je sledeća – pravi se struktura rešetke hiperkocke gde su njeni čvorovi različiti materijalizovani pogledi. Postoji veza između čvorova, tj. pogleda, koji se razlikuju za samo po jednu agregaciju. Svaki čvor se označava procenjenim brojem redova (postoje različite formule za ovu procenu, primer jedne je Kardenova formula $N = v\left(1-\left(1-\frac{1}{v}\right)^{n}\right)$, gde je N procenjeni broj redova pogleda, n broj redova u tabeli činjenica i v teorijski najveći mogući broj redova u pogledu (broj mogućih različitih ključeva u prostoru pogleda)). U svakoj iteraciji algoritma (broj iteracija jednak broju materijalizovanih pogleda koji želimo) se polazi od čvora tabele činjenica (početnog, neagregiranog) i traži se čvor, tj. pogled, kojim bi se najviše smanjio broj redova koji se obrađuje (ušteda u redovima se računa oduzimanjem broja redova datog čvora od broja redova čvora u odnosu na kojeg se računa ušteda, pomnoženo sa brojem čvorova na koje bi ta ušteda imala efekta). U svakoj narednoj iteraciji se postupak ponavlja, samo bez provere prethodno izabranog čvora.

Dodatno, u opštem slučaju, sa dimenzijama koje imaju više nivoa se dodaje i više čvorova. Takođe, čvorovima se mogu pridodati i težine u zavisnosti od učestalosti upotrebe (u upitima, tj. pri analizama).

Mana ovog algoritma je što se u svakom koraku računa ušteda za sve preostale čvorove rešetke, pa je njegova vremenska složenost $O(k*2^{2d})$, gde je k broj iteracija, tj. željenih pogleda, a d broj dimenzija.



	Iteration 1 Benefit	Iteration 2 Benefit
{p, s}	5.2M × 4 = 20.8M	
{c, s}	0 × 4 = 0	0 × 2 = 0
{c, p}	0 × 4 = 0	0 × 2 = 0
{s}	5.99M × 2 = 11.98M	0.79M × 2 = 1.58M
{p}	5.8M × 2 = 11.6M	0.6M × 2 = 1.2M
{c}	5.9M × 2 = 11.8M	5.9M × 2 = 11.8M
{}	6M − 1	0.8M − 1

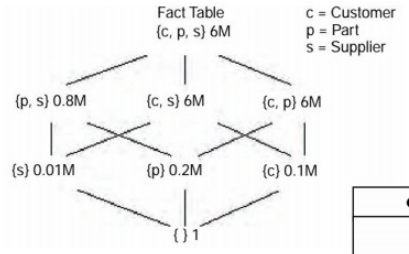
299. Algoritam PGA odabira materijalizovanih pogleda.

Algoritam PGA je heuristički algoritam odabira materijalizovanih pogleda. Slično kao HRU, izvršava se u iteracijama nad rešetkom hiperkocke sa čvorovima koji predstavljaju poglede.

Ideja iza algoritma je sledeća - pravi se struktura rešetke hiperkocke gde su njeni čvorovi različiti materijalizovani pogledi. Postoji veza između čvorova, tj. pogleda, koji se razlikuju za samo po jednu agregaciju. Svaki čvor se označava procenjenim brojem redova (postoje različite formule za ovu procenu, primer jedne je Kardenova formula $N = v\left(1-\left(1-\frac{1}{v}\right)^{n}\right)$, gde je *N* procenjeni broj redova pogleda, *n* broj redova u tabeli činjenica i *v* teorijski najveći mogući broj redova u pogledu (broj mogućih različitih ključeva u prostoru pogleda)). Svaka iteracija algoritma se deli na dve faze:

- Faza nominacije u kojoj se nalaze dobri kandidati tako što se na svakom "nivou" rešetke nominuje najmanji pogled, s tim da svaki naredni mora biti "dete čvor" prethodno izabranog. Dodatno, ne mogu se nominovati oni kandidati (osim potpune agregacije) koji su nominovani u prethodnoj iteraciji.
- Faza izbora u kojoj se za sve kandidate izračunava dobit oduzimanjem broja redova datog čvora od broja redova čvora u odnosu na kojeg se računa ušteda, pomnoženo sa brojem čvorova na koje bi ta ušteda imala efekta. Nakon ovoga se bira onaj čvor, tj. materijalizovani pogled, koji donosi najveću dobit.

Prednost ovog algoritma u odnosu na HRU je u mnogo efikasnijim iteracijama usled manjeg broja čvorova za koje se računa ušteda, tj. dobit (ne obilaze se svi čvorovi kao u HRU).



Candidates	Iteration 1 Benefit
{p,s}	**5.2M x 4 = 20.8M**
{s}	5.99M x 2 = 11.98M
{}	6M - 1

Candidates	Iteration 2 Benefit
{c,s}	0 x 2 = 0
{c}	**5.9M x 2 = 11.8M**
{}	6M - 1

300. Šta je istraživanje podataka?

Postoji više različitih prihvaćenih definicija, ali jedna od njih glasi – netrivijalno izdvajanje implicitnih, prethodno nepoznatih korisnih informacija iz baza podataka.

Jedan primer primene istraživanja podataka bio bi izvođenje pravila dodeljivanja (*association rules*) nad podacima, sadržanim u skladištu podataka, o prodatim proizvodima u nekom lancu marketa. Na osnovu tih pravila bi imali uvid u to koji proizvodi se često, a koji retko, prodaju zajedno.

301. Šta je plan izvršavanja upita? Na osnovu čega se pravi?

Plan izvršavanja upita predstavlja graf (tačnije stablo) koji odgovara nekom upitu nad bazom podataka (napomena – isti upit može imati ogroman broj planova izvršavanja različitog kvaliteta). Ovaj graf, tj. stablo, predstavlja redosled koraka pri izvršavanju upita, gde svaki korak (čvor grafa) čine operacije koje se izvršavaju u sklopu tog koraka. Dodatno, plan izvršavanja takođe sadrži i informacije o strukturama podataka (indeksima) koji se upotrebljavaju prilikom sprovođenja operacija. Svakom koraku, kao i celokupnom planu izvršavanja, je pridodata cena izvršavanja (koja se može odnositi na procenjeno potrebno procesorsko vreme, IO operacije, memorijsko zauzeće i slično.).

Kako će ovaj plan upita izgledati zavisi od mnogo faktora – same strukture upita, statistika nad tabelama koje učestvuju u upitu (broj redova, selektivnost upita, ...), napravljenih indeksa i slično. Za jedan upit se često pravi veći broj potencijalnih planova, od kojih se bira onaj sa najmanjom procenjenom cenom.